Mälardalen University

Using Deep Learning To Optimize Next Day's Portfolio

Authors:
Tom Karlsson
Gottfrid Bylund
Haimanot Ayana
Anton Erlandsson

Supervisor:
Ying Ni

January 8, 2021

Contents

1 Introduction

Deep learning is a part of the machine learning umbrella and is based on artificial neural networks. Deep-learning have mostly been applied fields such as computer vision, speech recognition and natural language processing. However, in latter years, deep-learning has started to expand into the financial area as-well. Areas such as price forecasting, portfolio management, algorithmic trading and high performance computing are now subject to the deep-learning regime e.g. Heaton, Polson and Witte, (2016); Zhengyao et al. (2017).

In this case, price forecasting is of interest, because its output becomes input in a rather complex model, namely the Markowitz model, which needs the expected return of an asset/portfolio. These returns are usually computed pretty straight forward, often by deriving historical returns. However, using historical returns, they usually have no real predictive power of the future. By using a price prediction model called Long Short Term Memory (LSTM), inputs of historical data can be converted into a single next day price prediction which is based on the short term trend in which the price has moved. LSTM is an artificial recurrent neural network which is well-suited to classify, process and make predictions based on time series data (Ta et al., 2020). This should theoretically give an investor the possibility to compute next days stock prices for n number of assets within a portfolio and optimize their asset allocations in accordance with this knowledge.

By using this approach this paper aims to firstly predict next days stock prices for a given amount of assets within a portfolio based on historical data. This predicted stock price will then be used for calculating an expected return (mean) which will be fed into an optimizing model that maximizes the portfolios Sharpe for the next day. The deep learning driven allocation will then be evaluated by comparing it with how stocks actually performed. It will also be compared with a simple 1overN allocation and a random allocation to seek how those portfolios performs.

2 Background

2.1 Machine Learning

Machine learning is a computing system where a computer system learn from data through algorithm rather than explicit programming. A predictive algorithm will create a predictive model. A predictive model with financial data will give a predication based on the fed data that trained the model. There are different approaches/techniques to machine learning:

1. *Supervised learning* - The financial data in supervised learning has labeled features that define the data. It typically begins with an established set of data and a certain understanding of how that data is classified.

2. *Unsupervised learning* - When the data is massive and unlabeled like data in social medias as in twitter and Instagram, unsupervised data is usually well-suited for task. The objective of this technique is to analyze the data without human intervention. It uses algorithms to classify the data based on the patterns and based on the relationship between the data points themselves.

3. *Reinforcement learning* - Reinforcement learning is a behavioural learning model. It consist of an agent and an environment where the system is not trained with sample data set like other machine learning models. Rather, it learns through trial and error. Sequence of successful decisions will reinforce the process. The algorithm receives feedback from the data analysis and guides the agent to the best outcome.

4. *Deep learning* - Deep learning incorporates neutral networks in successive layers to learn from data in an iterative manner. This approach is useful in learning patterns from unstructured data. It emulates human brain (the deep learning complex neural networks design). Deep learning is trained to deal with poorly defined abstractions and problems. It is widely used in image and speech recognition and computer vision applications.

2.2 Deep Learning

Within deep learning there is an sub field named Long Short Term Memory (LSTM) which can be described an artificial Recurrent Neural Network (RNN). LSTM which can process entire sequences of data and has the ability to learn long term sequences of observations. RNN have short term memory in that they use persistent previous information to be used in the current neutral network. Thus it can experience a loss of information which is known as the vanishing gradient problem. This is caused by the repeated used of the recurrent weight matrix in RNN. In an LSTM model, the recurrent weight matrix is replaced by an identity function and it is controlled by a series of gates. The input gate, output gate and forget gate acts like a switch that controls the weights and creates the long term memory function (Yu, Y, et al., 2019).

2.2.1 Long Short Term Memory (LSTM)

LSTM networks are one of the most advanced deep learning architectures for sequence learning tasks such as handwriting recognition, speech recognition or time series prediction. LSTM is an artificial RNN architecture used in the field of deep learning. Unlike standard feed forward neural networks, LSTM has feedback connections. LSTM is widely used for sequence prediction problems and have proven to be extremely effective.The reason it works so well is because LSTM has the ability to store past information that is important and forget otherwise. This is important in our case because the previous price of a stock is crucial in predicting its future price (Gers, Schmidhuber and Cummins, 1999).

3 Method

3.1 Recurrent Neural Networks (RNN)

Consider an hidden Markov model (HMM) where h_t is a hidden Markov chain with transition probability p$(h_t|h_{t-1})$. Assume y_t as the observations from p$(y_t|h_t)$ where the conditionally independent are $y_1, ..., y_T$ given by $h_1, ..., h_T$.

Then let the hidden states (as parameters) from its own parameters θ, ψ as

$$\begin{aligned} h_t|h_{t-1} &\sim p(h_t|h_{t-1}), \theta), \\ y_t|h_t &\sim p(h_t|h_t, \psi). \end{aligned}$$

By introducing a complementary layer of observed input x_t into the dynamics of the hidden states, the result is

$$\begin{aligned} h_t|h_{t-1}, x_t &\sim p(h_t|h_{t-1}, x_t, \theta) \\ y_t|h_t &\sim p(y_t|h_t, \psi) \end{aligned}$$

We aim to predict y_{T+1} based on y_{ttT}. In general parameters of θ and ψ can determined by Markov chain, Monte Carlo, EM-algorithm, etc. Let f be a deterministic function transiting state h_{t-1} to h_t, a traditional option for f is then the hyperbolic tangent function. What follows is the explicit relationship of:

$$\begin{aligned} h_t &= f(h_{t-1}, x_t; \theta \\ y_t|h_t &\sim p(y_t|, \psi) \end{aligned}$$

Denote the parameters $\theta = (b, W, U)$ and $\psi = (c, V)$ where b, c are vectors and W,U,V are matrices. Of a initial state h_0, which is given, the forward propagation can then be described as;

$$\begin{aligned} h_t &= tanh(b + Wh_{t-1} + Ux_t), \\ y_t &= \sigma(c + Vh_t). \end{aligned}$$

σ is the activation function, but can be changed due to which RNN model is chosen. Furthermore, the likelihood of h_t are x_t measurable. We can then describe the likelihood as;

$$L(y_t|x_t, \theta, \psi) = \sum_{t=1}^{T} L_t(y_t, g(h_t; \psi)),$$

where L can be translated to some error function. For examples, when dealing with regression problems, L can be mean absolute error or mean square error. To minimize the loss function, some stochastic gradient (SGD) method will be used as a foundation. For computation, SGD requires some gradients with respect to θ and ψ . For an RNN, this method has been named back propagation through time (BPTT). As an initial guess,we compute h_t, g$(h_t;\psi)$ for all time step t as follows,

$$\frac{\partial}{\partial\psi}L(x_t, y_t; \theta, \psi) = \sum_{t=1}^{T} \frac{\partial L_t(y_t, g(h_t;\psi)}{\partial g}\frac{\partial g(h_t;\psi)}{\partial\psi}$$

for the next being;

$$\frac{\partial}{\partial\psi}L(x_t, y_t; \theta, \psi) = \sum_{t=1}^{T} \frac{\partial L_t(y_t, g(h_t;\psi)}{\partial g}\frac{\partial g(h_t;\psi)}{\partial\psi}\frac{\partial h_t}{\partial_\theta}$$

where

$$\frac{h_t}{\partial\psi} = \frac{\partial f(h_{t-1}, x_t; \theta_t)}{\partial\theta_t}|_{\theta_t=\theta} + \frac{\partial f(h_{t-1}, x_t; \theta_t)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial\theta}$$

and $\theta_t = \theta$, $\forall$t is dummy variable for notation convenience.

To quest with minimizing the loss function, the Adam optimization algorithm was picked to be used as the optimizer for the model. The algorithm is an extension of the stochastic gradient descent algorithm and has great advantages in solving the non-convex optimization problem (Kingma and Ba, 2014). During the training phase of the model, the Adam optimization is used, and combined with MSE as the loss function.

3.2 How is a LSTM-model set up?

Before fitting an LSTM model to the data set, the data must be transformed. These usually occurs in three steps. The first step is to transform the time series into a supervised learning program, the second one is to transform the time series data so that it is stationary, the last one is to transform the observations to have a specific scale.

For a time series problem, we can achieve this by using the observation from the last time step ($t-1$) as the input and the observation at the current time step (t) as the output. Stock prices are non stationary. This means that there is a structure in the data that is dependent on the time with an increasing trend in the data. Stationary data is easier to model and will very likely result in more skillful forecasts. The trend can be removed from the observations, then added back to forecasts later to return the prediction to the original scale and calculate a comparable error score.

A standard way to remove a trend is by differentiating the data. That is the observation from the previous time step $(t-1)$ is subtracted from the current observation (t). This removes the trend and we are left with a difference series, or the changes to the observations from one time step to the next. Like other neural networks, LSTM expects data to be within the scale of the activation function used by the network. The default activation function for LSTM is the hyperbolic tangent (tanh), which outputs values between -1 and 1. This is the preferred range for the time series data.

We can transform the data set to the range [-1, 1] using the MinMaxScaler class. Like other Scikit-learn transform classes, it requires data provided in a matrix format with rows and columns. Therefore, we must reshape our NumPy arrays before transforming.

3.2.1 Equations of the LSTM cell

Through out our paper, we depicted matrices with upper case letters and vectors with lower case letters. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. Below we will discuss the equations for each unit. For $x_t \in \mathrm{R}^N$, where N is the feature length of each time step, while i_t, f_t, o_t, h_t, h_{t-1}, c_t, c_{t-1}, $b \in \mathrm{R}^H$, where H is the hidden state dimension. The LSTM equations are the following:

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_0) \\ h_t &= o_t \circ \tanh(c_t) \end{aligned}$$

Equation 1: the input gate

$$i_t = \sigma(W_{xi}xx_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i)$$

The weight matrices represent the memory of the cell. The input x_t is in the current input time step, while h and c are indexed with the previous time step. Every matrix W is a linear (or simply a matrix multiplication). This function enables us to take multiple linear combinations of x, h , c and match the dimensionality of input x to the one of h and c.

The dimensionalities of h and c are basically the hidden states parameter in a deep learning framework.

The bias term is part of the linear layer and is simply a trainable vector that is added. The output is also in the dimensionality of the hidden and context/cell vector. Finally, after the 3 linear layers from different inputs, we have a non-linear activation function to introduce non-linearity, which enables the learning of more complex representations. In this case, the sigmoid function is usually used.

Equation 2: the forget gate

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

Equation 2 is similar with equation 1. However, the weight matrices are different this time. This mans that we get a different set of linear combinations, that represent different things to model different things.

Equation 3: the new cell / context vector

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1}) + b_c)$$

Here, we have another linear combination of the input and hidden vector, which is totally different from the previous two! This term is the new cell information, passed by the tanh function so as to introduce non-linearity and stabilize training.

Besides updating the cell with the new states, we want to consider previous states; that's why we designed RNN's anyway! This is where the calculated input gate vector i comes into play. We filter the new cell info by applying an element-wise multiplication with the input gate vector i.

The forget gate vector comes into play now. Instead of just adding the filtered input info, we first perform an element$-$wise vector multiplication with the previous context vector. To this end, we would like the model to mimic the forgetting notion of humans as a multiplication filter.

By adding the previously described term in the tanh parenthesis, we get the new cell state, as shown in Equation 3.

Equation 4: the forget gate

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_0)$$

Let's just take another linear combination! This time, of our 3 vectors x_t, h_{t-1}, c_t while we add another non-linear function in the end. Note that, we will now use the calculated new cell state (c_t) as opposed to equations 1 and 2. We have almost calculated the desired output vector o_t of a single cell in a particular time step.

Equation 5: the new context

$$h_t = o_t \circ \tanh(c_t)$$

The title in equation 4 was the almost new output. Thus, one can calculate the new output (literally the new hidden state) based on equation 5. Instead of producing an output as shown in equation 4, we further inject the vector called context. Looking back in equations 1 and 2, one can observe that the previous context was involved. In this way, information based on previous time steps is involved. This notion of context enabled the modeling of temporal correlations in long term sequences.

Basically, a single cell receives as input the cell and hidden state from the previous time step, as well as the input vector from the current time step.

Each LSTM cell outputs the new cell state and a hidden state, which will be used for processing the next time step. The output of the cell, if needed for example in the next layer, is its hidden state.

The new output (the new hidden state can be calculated) based on equation 5.

3.3 Markowitz portfolio optimization

A portfolio with N assets, can be considered by letting R_j illustrates the return of assets i during time. Time is used in intervals, given t_{-1}, t and $R = (R_1, ..., R_N)'$, where $\mu = E[R]$, and $\Sigma = Cov(R, R')$. Hence, optimal weights of the portfolio can be found by w_i for asset i,j $= 1, ..., N$ given that the portfolio return is maximized. The boundary is set to $w_i \in [0, 1]$., meaning that the portfolio will not initiate any short positions (otherwise; constraints would be [-1, 1]).

By Markowitz optimization theory, the return of portfolio R_p is also a random variable with mean $E[R_p] = \mu' w$, followed that variance equals $Var(R_p) w' \Sigma w$. Expected portfolio return then become $\mu^* \in R$, which solves weights by using a quadratic equation as follows;

$$\begin{cases} \mathbf{w} = \mathbf{arg}\, min\ \mathbf{w'\Sigma w} \\ subject\ to\ \mu'\mathbf{w} = \mu\mathbf{w} \geq \mathbf{0}\ and\ \mathbf{1'w} = \mathbf{1}, \end{cases}$$

where the condition $w \geq 0$ applies to no short-selling portfolios. From these equations, it is possible to identify the μ^* two components can be distinguished, the co-variance matrix Σ and μ, the expected of the return asset vector.

In order to solve the quadratic equation, Scipy's minimize function within its optimization class has been used.In this instance, the method used for the minimization is Sequential Least Square Quadratic Programming (SLSQP). This method which is build on the Han–Powell quasi–Newton method, is often used for similar minimization problems.

3.3.1 Calculating the mean

Usually, when one is using the Markowitz's Framework, the mean μ is computed by using many years of historical price data. In this paper however, the mean is calculated as the return between P_t and PP_{t+1} . This creates the possibility to make the following function:

$$\mu = E(R_A)_{t+1} = \frac{PP^A_{t+1} - P^A_t}{P^A_t}$$

where $E(R_A)_{t+1}$ is the expected return of an asset at time t+1, PP^A_{t+1} is the predicted price for an specific asset price at t+1, P^A_t is the real price of asset at time t.

3.3.2 Risk factors

To decompose the risk factors aligned with the portfolio theory, this section will illustrate the difference factors for model on risk. However, due to the limitation of space, this section will not address the *systematic* and *unsystematic risk*, e.g. macro-economical- and fundamental factors. Moreover, the risk co-variance Σ can decomposed, using a number of arbitrary factors. Denoting M as a factor model with $f_m, m \in 1, ..., N$, the return of the asset i can be described as;

$$R_i = \alpha_i + \sum_{m=1}^{M} \beta_{i,m} f_m + \epsilon,\ i \in\ 1, ..., N,$$

where the $\beta_{i,m}$ is the sensitivity of asset i to factor f_m and $\int_i$, is equal to the idiosyncratic risk of the asset. Returns for all assets can be described as a matrix;

$$\begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} = \begin{bmatrix} \beta_{1,1} & \dots & \dots & \beta_{1,M} \\ \vdots & \ddots & \ddots & \vdots \\ \beta_{N,1} & \dots & \dots & \beta_{N,M} \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

or in the compact form of R = + f + ϵ. Hence, if V can be seen as the co-variance matrix of the factors, and thus, β is the least square projection. Where $\beta = Cov(R, f)V^{-1}$ it leads to that $Cov(F, \epsilon) = 0$. The co-variance of the return can then be assigned to

$$\sum = Cov(R, R') = \beta V \beta' + D.$$

where D is diagonal matrix, V is the co-variance matrix of the factors and β can be viewed as an indicator of an assets vulnerability to systematic risk. In other words, the coefficient of the β indicates the degree to which the asset's return is correlated with market. If β =1 for an asset, it is said to be in phase with the overall market. With a β ¿ 1, the asset act *more* volatile than the overall market, conversely, if β ¡ 1 the asset have *less* volatility than the market overall.

3.4 Measuring error

Since we are working with both deep learning and time-series prediction, measuring errors is of interest. For the model-training itself, a loss (error) is measured in order to evaluate how well a model during training is fitting to the actual values. In our case, Mean Square Error (MSE) has been chosen as the loss function and RMSE for validating the model.

Since we are squaring the errors, larger mistakes result in larger errors which. This means that we are punishing the models learning more when it makes larger mistakes. Using a loss-function also makes it possible to implement an early stop so that we can avoid overfitting-underfitting the model. This means that if the model has not improved for a certain amount of epochs, the training is terminated. MSE can be calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y_i)$$

Another widely used error measurement is Root Mean Square Error (RMSE). It is just the square root of the mean square error. This is more used for overall evaluation of a trained model as it is directly interpretable in terms of measurement units. RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y_i)}$$

To not only evaluate the models total error and performance, we also measure the out of sample error for the t+1 predicted stock price. This is important because we want to understand if the t+1 predicted price has an error that is in line with the overall models error. If the t+1 predicted price error is much larger than the model's overall error, our out-of-sample prediction could be inferred to not be that reliable/accurate. To measure these errors we use absolute and relative error. Relative error is often used for measuring the accuracy of a prediction. Absolute and relative error is calculated as:

$$Absolute\ Error = x_i - x$$

$$Relative\ Error = \frac{|x_i - x|}{x}$$

4 Results

As earlier mentioned, the paper is supposed to optimize an portfolio with an ML and DL approach through LSTM. To be able to do this, stocks from Nasdaq Large Cap Stockholm Stock Exchange will firstly be filtered and thereafter be selected to create the portfolio that will be used in the paper and in the end become optimized. Secondly, the portfolio and its data will be processed, prepared for LSTM and presented. Thirdly, by using the predicted data, the selected portfolio will be optimized with help of Markowitz's work through an Scipy-approach.

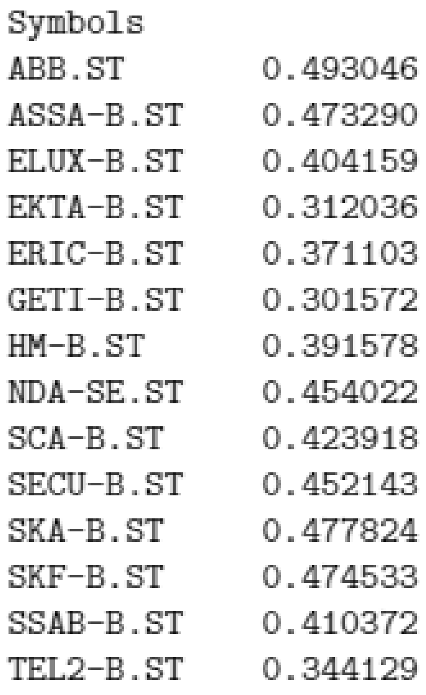
4.1 Selection of the portfolio

The selection of the portfolio is based on a way of filtering companies to achieve an uncorrelated combination of several stocks. The companies that is initially chosen in the filtering phase came from the Nasdaq Large Cap Stockholm Stock Exchange. These companies then became filtered in regards to their listing date and the 100-day rolling trading volume. Companies that was listed before 2000-01-01 was excluded from the data set, as well as companies that respectively had a lower 100-day rolling trading volume than the average 100-day rolling trading volume of the companies combined. This first phase of the filtering resulted in a total of 21 companies.

21 companies is still quite a large sample which the authors wanted to minimize in a relevant way. The second phase of the filtering was based

on sector and industry, in specific the industry that the companies operates in since this describes more than just the sector. By coding, the outcome is estimated that there were 14 unique industries in the data set, with duplicates taken away. To receive a nice diversification, we used companies from these 14 unique industries and retrieved data from 2000-01-01 to 2020-12-10.

To filter the last batch of stocks, to end up with the final stocks that created our selected portfolio, the respectively log-return was calculated and shown as an correlation matrix. Further, the mean of the correlation of each company was calculated. To receive an as uncorrelated portfolio as possible, the six stocks with the lowest mean correlation between each other was chosen. The result of the selected portfolio that was going to be optimized were: **SKA-B.ST**, **EKTA-B.ST**, **GETI-B.ST**, **ERIC-B.ST**, **TEL2-B.ST** and **HM-B.ST**.



```
Symbols
ABB.ST       0.493046
ASSA-B.ST    0.473290
ELUX-B.ST    0.404159
EKTA-B.ST    0.312036
ERIC-B.ST    0.371103
GETI-B.ST    0.301572
HM-B.ST      0.391578
NDA-SE.ST    0.454022
SCA-B.ST     0.423918
SECU-B.ST    0.452143
SKA-B.ST     0.477824
SKF-B.ST     0.474533
SSAB-B.ST    0.410372
TEL2-B.ST    0.344129
```

Figure 1: Correlation mean

4.2 Results from LSTM model

After the portfolio was selected, it was time to train the data via our LSTM model. Below we see the models performance of predicting the stock price of **TEL2-B.ST**. We can clearly see that the model is not perfect, and it almost never predicts the correct price. It however does quite a good job of showing the overall trend the stock is moving at. Still, we believe that the model may be too accurate given the nature of the time-series. We would like to have seen larger lags whenever the price turned reversed. Because of this, be believe that the model/s may be over fitted. The graph of the loss also shows some signs of this as the validation-loss is quite larger than the training loss. We would like to see a relationship that is more linear between the two loss-functions.

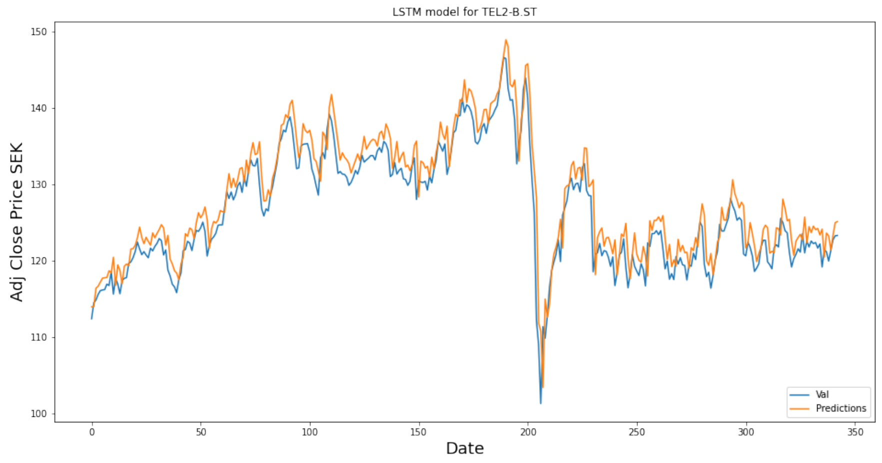


Figure 2: Predictions of stock prices

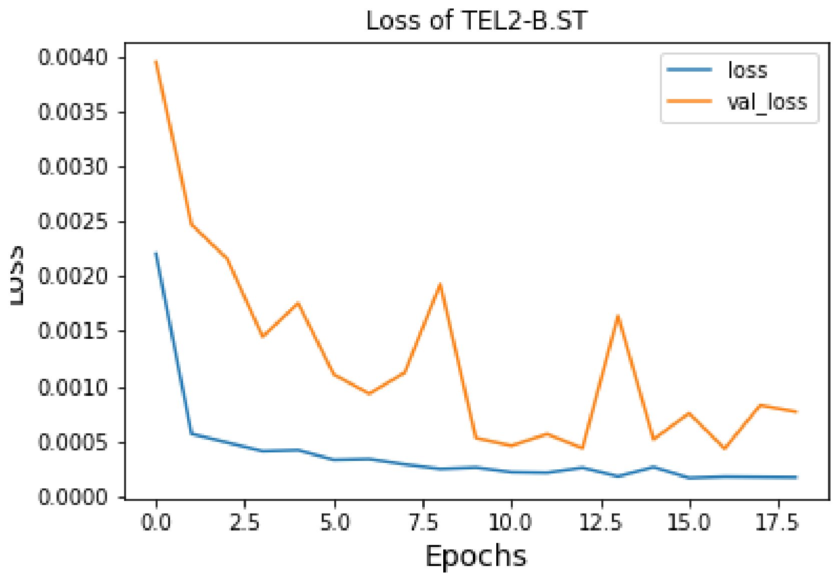


Figure 3: Model loss of TELE2

We can also use the Root Mean Squared Error (RMSE) function to understand how well our model performed.

In the case of **TEL2-B.ST**, the RMSE was around 2.86, which can be interpreted as an average stock price prediction error of 2.86 SEK. Since the stock price of **TEL2-B.ST** is around 100 SEK, an error of 2.86 SEK should be seen as modest. This could be seen in the graph below as well. **TEL2-B.ST** had the lowest relative error within the portfolio. By using absolute and relative error, we can also visualize the size of the errors.

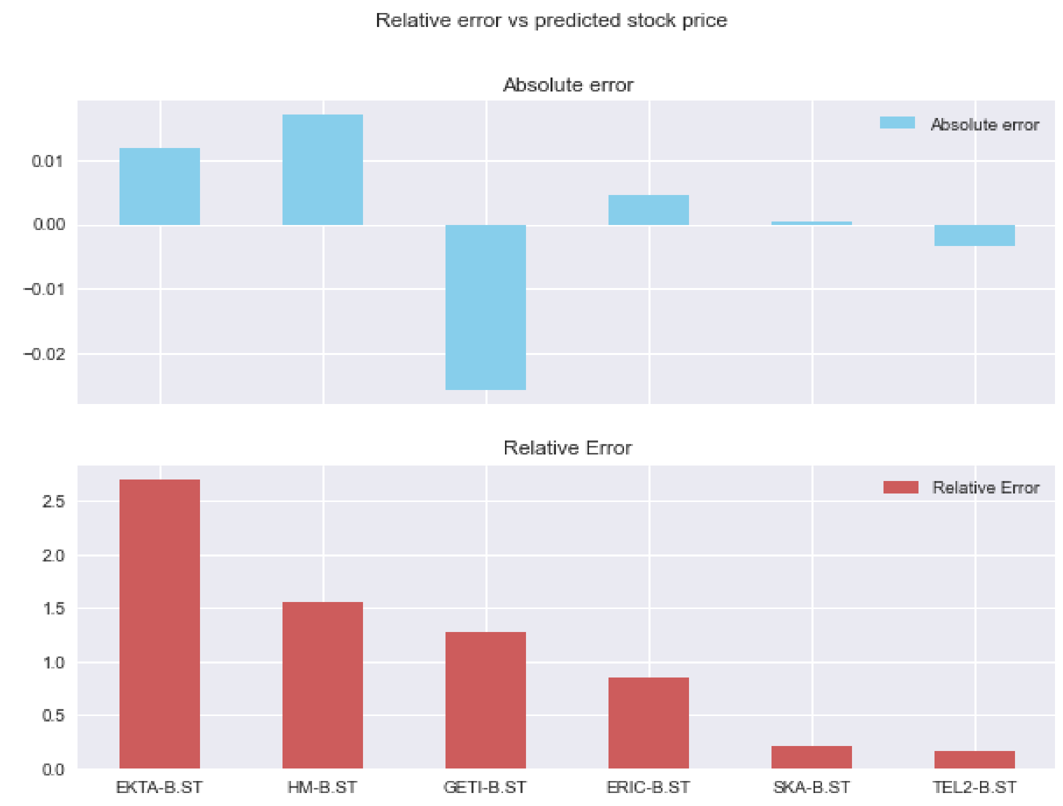


Figure 4: Error of predictions vs real price

In figure 4 the absolute error and relative error is shown. **GET-B.ST** has an negative error of 0.02 units, while **HM-B.ST** has a positive error of slightly above 0.01. As illustrated in the stock return diagram, **SKA-B.ST** has the lowest error for the model and the predicted stock price comes very close to the actual price. Hence, **GET-B.ST** and **TEL2-B.ST** has the highest actual return while **HM-B.ST** drops in return to the lowest level of stocks compared. Only **GET-B.ST** and **HM-B.ST** have diametrical predictions compared to actuals, meaning that the model succeeds with predicting the right direction (e.g. below or above zero) of the returns in four out of six cases.

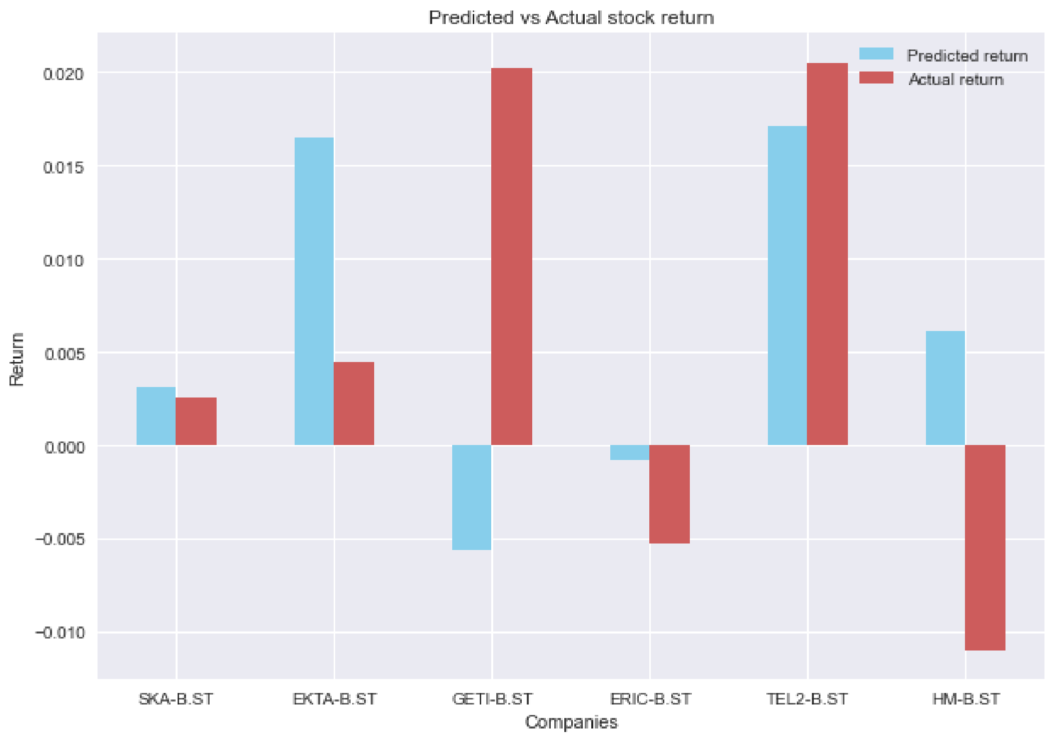


Figure 5: Absolute values prediction vs actual

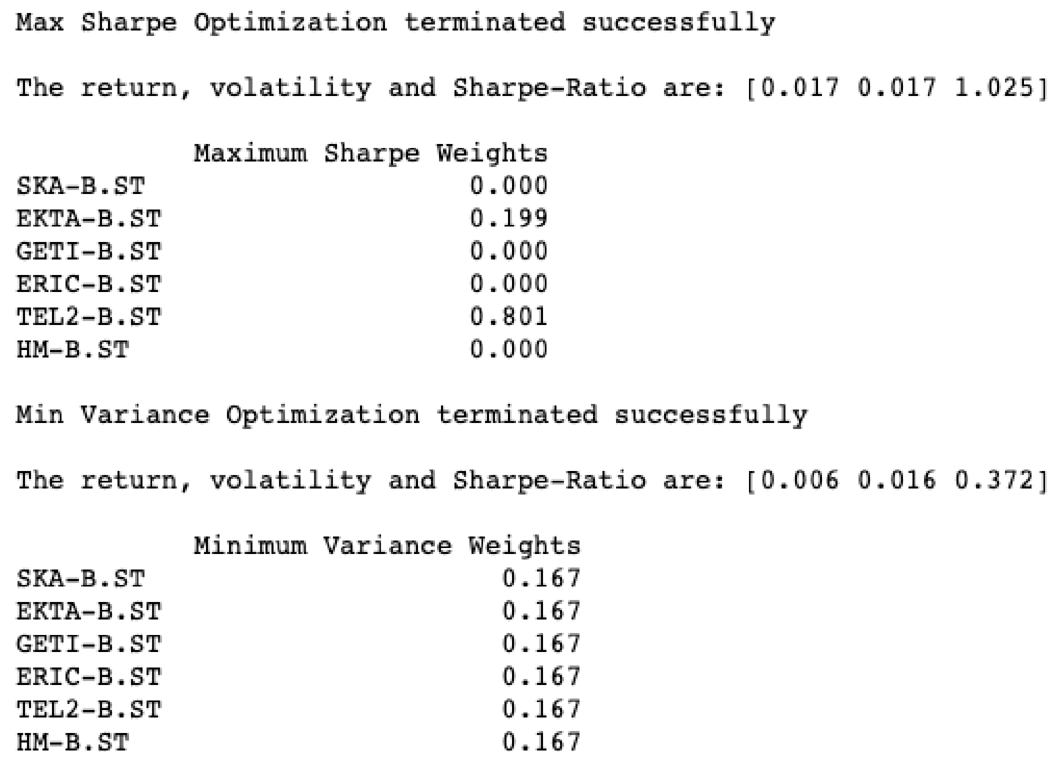
4.3 Optimizing the portfolio

As described earlier, the selected portfolio will be optimized with help of Markowitz's framework and in particular with an Scipy-approach. The LSTM model has earlier been trained with our selected portfolio and its respective stocks. In addition, the covariance matrix of the portfolio has been calculated. By moving forward with the Scipy-approach, the portfolio can be optimized to find the portfolio with the specific allocation that contributes to the portfolio with the highest Sharpe ratio or the portfolio with the minimum variance. Receiving weights or the allocation of the optimized portfolio can be done in two ways: by using the 1overN method to get equally distribution of weights between the assets or use a random optimizer that randomly chooses the weights so that the sum add up to 1.

Further, both these weights approaches and the statistics of the optimized selected portfolio, that has been trained through LSTM, can be seen below.

What the optimization shows is that the portfolio that would contribute to the highest Sharpe ratio would only consist of **EKTA-B.ST** and **TEL2-B.ST**, wheres the latter has an allocation size of 0.801. This portfolio would have an expected return of 0.017, and the volatility of 0.017 which ends up with the maximum Sharpe ratio of 1.025.

The lower part of the information below that is contributed to the minimum variance weights is somehow incorrect. The weights in itself is correct, but we did not manage to compute nor visualize the minimum variance portfolio in the end. The Monte-Carlo simulation, that will be used in the end to visualize the results, became incorrect when assessing the minimum variance portfolio. The weights was always chosen based on our initial guess. Since the LSTM model was based on a one-day basis the portfolio volatility and returns was not scaled to 256 trading days (1 year). This makes the covariances extremely small and there is probably a very small difference in volatility between different portfolios.



```
Max Sharpe Optimization terminated successfully

The return, volatility and Sharpe-Ratio are: [0.017 0.017 1.025]

           Maximum Sharpe Weights
SKA-B.ST                    0.000
EKTA-B.ST                   0.199
GETI-B.ST                   0.000
ERIC-B.ST                   0.000
TEL2-B.ST                   0.801
HM-B.ST                     0.000

Min Variance Optimization terminated successfully

The return, volatility and Sharpe-Ratio are: [0.006 0.016 0.372]

           Minimum Variance Weights
SKA-B.ST                      0.167
EKTA-B.ST                     0.167
GETI-B.ST                     0.167
ERIC-B.ST                     0.167
TEL2-B.ST                     0.167
HM-B.ST                       0.167
```

Figure 6: Optimization results

As we can see, the highest Sharpe ratio is achieved by Known prices maximize Sharpe, which stretches to above 1.2. On the other hand, the equal distribution of weights, 1overN acheives the lowest Sharpe with 0.3. LSTM achieves the best performing statistic when paired with maximizing Sharpe, in favor for random weights and 1overN.

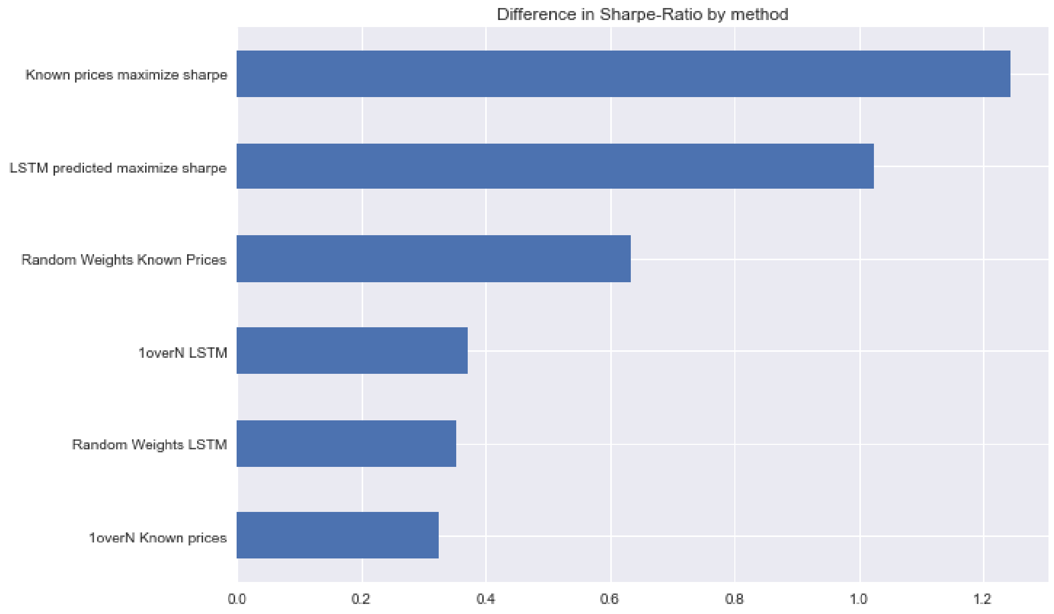


Figure 7: Difference in Sharpe Ratio

In order to visualize and demonstrate the efficient frontier, the following plot was generated. The efficient frontier is the different x notations, which summarize to a bow shaped line. The portfolios, which is represented by each individual point, was constructed by conducting a Monte Carlo simulation. As we can see our portfolio with maximized Sharpe ratio is denoted by the red star. In this attempt, we did not manage to maintain a portfolio with the lowest variance, however we discussed the details about this issue previous in this section.

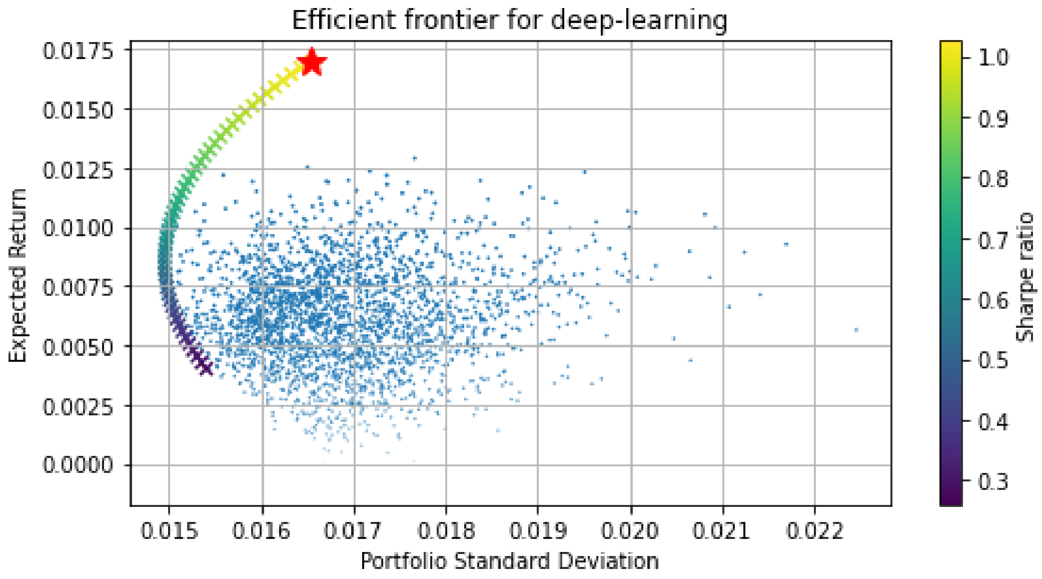


Figure 8: Illustration of efficient frontier

5 Conclusion

This paper had the interest of understanding how deep-learning within finance works and how a technique such as the Long Short Term Memory could be used to maximize the Sharpe ratio for tomorrows portfolio by predicting next days stock price. We selected a number of stocks and trained a LSTM model to learn the price movements for each stock. These trained models were then used to predict the next days stock price for every asset that were in the portfolio. The average RMSE for all the trained models were around 3.46. This means that the models was on average 3.46 SEK away from the actual value. All the stocks differ in price ranges, which makes the absolute value of 3.46 insipid. However, from the graphs presented in the result, we can see that the relative error between actual price and predicted price can be quite large.

When we translated the predicted prices into expected returns, we created the possibility to create a maximized Sharpe portfolio. The deep-learning Sharpe ratio were fairly close to the Sharpe ratio that the real optimized portfolio yielded. With deep-learning we got a Sharpe ratio of 1.02 while the actual Sharpe ratio was 1.244. The actual Sharpe were therefore 21 percent higher than our deep-learning Sharpe. Although neither Sharpe is considered to be astonishing, they are over the acceptable limit of 1. In order to understand the performance further, we also compared the deep-learning model with some other benchmarks. Compared with the 1overN and random weight allocation, the deep-learning yielded 50 percent higher Sharpe ratio than random weight and almost 70 percent better than 1overN.

Even though LSTM alludes some form of confidence in price prediction and better investment decision information compared to the benchmarks, there is still some big uncertainties in this approach. Firstly, stock prices are non-stationary, which means that the LSTM may have a hard time finding patterns compared to for example weather time series. Secondly, the models has to be retrained every single day in order to predict the next days stock price. This may be a very time-consuming and tedious process. Thirdly, we have not included risk-free rate because of not complicating the project work more than necessary. In order for the portfolio to be "complete", risk-free rate should be included. However, at the time as this paper was written, the risk free rate was negative (PwC, 2020). Thus it would be less attractive with such a asset in the portfolio.

6 References

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

Heaton, J., Polson, N. and Witte, J. (2016). Deep learning for finance: deep portfolios. Applied Stochastic Models in Business and Industry, 33(1), pp.3-12.

Hilpisch, Y. (2015). Derivatives Analytics With Python: Data Analysis, Models, Simulation, Calibration And Hedging. Wiley.

Hilpisch, Y. (2018). Python For Finance: Master Data-Driven Finance. 2nd ed. O'Reilly Media.

Jiang, Z.Y., Xu, D.X. and Liang, J.J. (2017). A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem.

Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Ta, VD., Liu, CM., and Addis Tadeese, D. (2020). Portfolio Optimization-Based Stock Prediction Using Long-Short Term Memory Network in Quantitative Trading.

Yu, Y., Si, X., Hu,C., and Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures.

PWC Report. 2020. https://www.pwc.se/sv/corporate-finance/riskpremiestudien.html (Accessed 2020-12-20).

7 Python source code

Project work

January 6, 2021

```
[3]: #*****Imports for dataprocessing********
import pandas as pd
import numpy as np
from numpy import array
import math
import pandas_datareader as pdr
import matplotlib.pyplot as plt
from heapq import nsmallest
import scipy.optimize as sco
save_path = '/Users/antonerlandsson/Documents/Models'

#*****Imports for designing the LSTM model.******
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.callbacks import EarlyStopping
```

```
[51]: plt.style.use('seaborn')
```

The code is splitted into multiple steps

1. Filtering and finding initial stocks for the portfolio.
2. Writing a portfolio class and training a LSTM model with the selected stocks.
3. Calculating returns and covariances
4. Optimizing the portfolio
5. Results

1 Filtering and finding initial stocks for the portfolio.

```
[26]: # Reading in a list of stocks from the NASDAQ largecap exchange. These are then␣
 ↪filtered for time on the stoch exchange.
stocks_raw = pd.read_csv('/mnt/c/Users/Anton/Dropbox/PythonProjects/Python␣
 ↪Projects/Python in Financial Engineering/Project Work/largecap.csv', sep=';
 ↪', encoding='latin-1', parse_dates = True, infer_datetime_format = True,␣
 ↪index_col = 'listingdate')
stocks_filtered = stocks_raw[stocks_raw.index <= '2000-01-01'].copy()
stocks_filtered
```

[26]:
```
                        company   volume100d       sector  \
listingdate
1999-06-22                  ABB  1411425,625  Industrials
1994-11-08           Assa Abloy  2023678,875  Industrials
1999-04-06          AstraZeneca  501995,8438  Health Care
1973-01-01       Atlas Copco A     1111550,5  Industrials
1994-07-14   Atrium Ljungberg  179766,4063   Financials
…                           …            …            …
1989-07-03               SSAB B    3446860,5    Materials
1997-01-02         Stora Enso A  13966,51953    Materials
1998-08-24              Sweco A  2118,939941  Industrials
1996-05-14              Tele2 A  2005,920044      Telecom
1982-01-01              Volvo A   275950,375  Industrials

                               industry
listingdate
1999-06-22          Industrial Machinery
1994-11-08        Construction Supplies
1999-04-06              Pharmaceuticals
1973-01-01          Industrial Machinery
1994-07-14                  Real Estate
…                                     …
1989-07-03   Mining - Steel & Aluminum
1997-01-02       Forest & Wood Products
1998-08-24         Business Consultants
1996-05-14           Telecommunications
1982-01-01          Industrial Machinery

[77 rows x 4 columns]
```

[27]:
```
# Filtering for stocks that have a volume that is higher than the mean. This is␣
↪to only get those stocks that are liquid
stocks_filtered['volume100d'] = stocks_filtered['volume100d'].str.replace(',',␣
↪'.').astype(float)
stocks_filtered = stocks_filtered[stocks_filtered['volume100d'] >=␣
↪stocks_filtered['volume100d'].mean()]
stocks_filtered
```

[27]:
```
                        company   volume100d             sector  \
listingdate
1999-06-22                  ABB  1411425.625        Industrials
1994-11-08           Assa Abloy  2023678.875        Industrials
1973-01-01        Atlas Copco A  1111550.500        Industrials
1985-01-01         Electrolux B  1247480.375  Consumer Durables
1994-03-01               Elekta  1535047.875        Health Care
1976-01-01           Ericsson B  7240196.500         Technology
1995-06-08              Getinge  1074312.375        Health Care
```

```
1974-06-17   Hennes & Mauritz  4813372.000  Consumer Durables
1997-12-15        Nordea Bank  7857396.500         Financials
1989-01-01            Sandvik  2624841.500        Industrials
1987-01-01              SCA B  1765903.125          Materials
1975-01-01              SEB A  4364754.000         Financials
1991-07-09          Securitas  1292955.250        Industrials
1987-01-01    Handelsbanken A  5293509.000         Financials
1965-01-01            Skanska  1079145.125        Industrials
1982-01-01              SKF B  1549650.250        Industrials
1989-07-03             SSAB A  4616178.500          Materials
1995-06-09           Swedbank  3557981.500         Financials
1996-05-14            Tele2 B  2666959.500            Telecom
1982-01-01            Volvo B  4372945.000        Industrials
1989-07-03             SSAB B  3446860.500          Materials

                                  industry
listingdate
1999-06-22             Industrial Machinery
1994-11-08            Construction Supplies
1973-01-01             Industrial Machinery
1985-01-01             Consumer Electronics
1994-03-01                Medical Equipment
1976-01-01                   Communications
1995-06-08                 Medical Supplies
1974-06-17               Clothing & Footwear
1997-12-15                            Banks
1989-01-01             Industrial Machinery
1987-01-01           Forest & Wood Products
1975-01-01                            Banks
1991-07-09                         Security
1987-01-01                            Banks
1965-01-01   Construction & Infrastructure
1982-01-01            Industrial Components
1989-07-03          Mining - Steel & Aluminum
1995-06-09                            Banks
1996-05-14               Telecommunications
1982-01-01             Industrial Machinery
1989-07-03          Mining - Steel & Aluminum
```

[18]:
```python
# Looking how many unique sectors there are.

print(len(stocks_raw['sector'].unique()))
print(len(stocks_filtered['sector'].unique()))
len(stocks_filtered['sector'].unique()) / len(stocks_raw['sector'].unique())
```

```
9
7
```

```
[18]: 0.7777777777777778
```

```
[22]: # Dropping the duplicates within industry and sector.

      stocks_filtered_industry = stocks_filtered.drop_duplicates('industry')
      stocks_filtered_sector = stocks_filtered.drop_duplicates('sector')
      print(len(stocks_filtered_industry))
      print(len(stocks_filtered_sector))
```

```
14
7
```

```
[23]: # Printing the results after removing duplicates.

      print(stocks_filtered_industry)
      print(stocks_filtered_sector)
```

```
                       company   volume100d            sector  \
listingdate
1999-06-22                 ABB  1411425.625       Industrials
1994-11-08         Assa Abloy  2023678.875       Industrials
1985-01-01        Electrolux B  1247480.375  Consumer Durables
1994-03-01              Elekta  1535047.875        Health Care
1976-01-01          Ericsson B  7240196.500         Technology
1995-06-08             Getinge  1074312.375        Health Care
1974-06-17    Hennes & Mauritz  4813372.000  Consumer Durables
1997-12-15         Nordea Bank  7857396.500         Financials
1987-01-01               SCA B  1765903.125          Materials
1991-07-09           Securitas  1292955.250        Industrials
1965-01-01             Skanska  1079145.125        Industrials
1982-01-01               SKF B  1549650.250        Industrials
1989-07-03              SSAB A  4616178.500          Materials
1996-05-14             Tele2 B  2666959.500            Telecom

                                    industry
listingdate
1999-06-22           Industrial Machinery
1994-11-08          Construction Supplies
1985-01-01           Consumer Electronics
1994-03-01              Medical Equipment
1976-01-01                 Communications
1995-06-08               Medical Supplies
1974-06-17             Clothing & Footwear
1997-12-15                          Banks
1987-01-01          Forest & Wood Products
1991-07-09                       Security
1965-01-01    Construction & Infrastructure
1982-01-01          Industrial Components
```

```
1989-07-03       Mining - Steel & Aluminum
1996-05-14               Telecommunications
                  company   volume100d             sector  \
listingdate
1999-06-22            ABB  1411425.625        Industrials
1985-01-01   Electrolux B  1247480.375  Consumer Durables
1994-03-01         Elekta  1535047.875        Health Care
1976-01-01     Ericsson B  7240196.500         Technology
1997-12-15    Nordea Bank  7857396.500         Financials
1987-01-01          SCA B  1765903.125          Materials
1996-05-14        Tele2 B  2666959.500            Telecom

                             industry
listingdate
1999-06-22     Industrial Machinery
1985-01-01     Consumer Electronics
1994-03-01        Medical Equipment
1976-01-01           Communications
1997-12-15                    Banks
1987-01-01   Forest & Wood Products
1996-05-14       Telecommunications
```

[5]:
```
# Picking the stocks from the unique industry list as it gives a nice␣
↪diversification.
tickers = ['ABB.ST', 'ASSA-B.ST', 'ELUX-B.ST', 'EKTA-B.ST', 'ERIC-B.ST',␣
↪'GETI-B.ST', 'HM-B.ST', 'NDA-SE.ST', 'SCA-B.ST', 'SECU-B.ST', 'SKA-B.ST',␣
↪'SKF-B.ST', 'SSAB-B.ST', 'TEL2-B.ST']
```

[45]:
```
df = pdr.get_data_yahoo(tickers, start = '2000-01-01', end =␣
↪'2020-12-10')['Close']
df = df.dropna()
df.head()
```

[45]:
```
Symbols         ABB.ST  ASSA-B.ST   ELUX-B.ST  EKTA-B.ST  ERIC-B.ST  \
Date
2012-06-18  112.599998  62.400002  127.900002  81.449997  63.299999
2012-06-19  114.900002  62.866699  129.800003  81.050003  64.500000
2012-06-20  115.099998  63.466702  136.300003  81.599998  64.800003
2012-06-21  114.800003  63.366699  137.199997  81.849998  63.599998
2012-06-25  109.900002  62.400002  133.000000  79.500000  62.000000

Symbols      GETI-B.ST     HM-B.ST  NDA-SE.ST   SCA-B.ST  SECU-B.ST  \
Date
2012-06-18  143.835999  226.399994  56.250000  21.233601  53.599998
2012-06-19  143.673996  230.399994  57.799999  21.233601  54.049999
2012-06-20  142.298004  241.500000  58.150002  21.233601  54.349998
2012-06-21  141.246002  246.199997  57.650002  21.090900  53.849998
```

```
2012-06-25  140.113007  242.300003  56.400002  20.601299  52.450001

Symbols       SKA-B.ST    SKF-B.ST  SSAB-B.ST   TEL2-B.ST
Date
2012-06-18   99.699997  132.199997  34.571098  101.084000
2012-06-19  100.500000  134.500000  35.603901  102.625999
2012-06-20  101.900002  137.300003  36.622002  102.722000
2012-06-21  101.400002  135.100006  37.098000   99.734802
2012-06-25   98.900002  129.500000  34.651699   99.445702
```

[62]:
```
# We compute the log-return correlation matrix.

logReturn = np.log(df/df.shift(1))
corr_matrix = logReturn.corr()
corr_matrix_box_mean = corr_matrix.mean()
print(corr_matrix)
```

```
Symbols       ABB.ST  ASSA-B.ST  ELUX-B.ST  EKTA-B.ST  ERIC-B.ST  GETI-B.ST  \
Symbols
ABB.ST      1.000000   0.551133   0.450056   0.309828   0.408138   0.278441
ASSA-B.ST   0.551133   1.000000   0.412301   0.291934   0.365384   0.330482
ELUX-B.ST   0.450056   0.412301   1.000000   0.230915   0.287909   0.251236
EKTA-B.ST   0.309828   0.291934   0.230915   1.000000   0.215705   0.266704
ERIC-B.ST   0.408138   0.365384   0.287909   0.215705   1.000000   0.250783
GETI-B.ST   0.278441   0.330482   0.251236   0.266704   0.250783   1.000000
HM-B.ST     0.395156   0.406861   0.361358   0.242273   0.270380   0.154107
NDA-SE.ST   0.536900   0.473044   0.415614   0.242199   0.367478   0.229671
SCA-B.ST    0.466167   0.444881   0.380658   0.296256   0.363928   0.288879
SECU-B.ST   0.501598   0.538117   0.391645   0.280789   0.345475   0.254558
SKA-B.ST    0.557000   0.538289   0.413516   0.285710   0.353931   0.256461
SKF-B.ST    0.612646   0.545771   0.447416   0.281339   0.328304   0.264776
SSAB-B.ST   0.483609   0.390880   0.352877   0.235829   0.302891   0.174476
TEL2-B.ST   0.351977   0.336978   0.262719   0.189015   0.335138   0.221433

Symbols      HM-B.ST  NDA-SE.ST  SCA-B.ST  SECU-B.ST  SKA-B.ST  SKF-B.ST  \
Symbols
ABB.ST      0.395156   0.536900  0.466167   0.501598  0.557000  0.612646
ASSA-B.ST   0.406861   0.473044  0.444881   0.538117  0.538289  0.545771
ELUX-B.ST   0.361358   0.415614  0.380658   0.391645  0.413516  0.447416
EKTA-B.ST   0.242273   0.242199  0.296256   0.280789  0.285710  0.281339
ERIC-B.ST   0.270380   0.367478  0.363928   0.345475  0.353931  0.328304
GETI-B.ST   0.154107   0.229671  0.288879   0.254558  0.256461  0.264776
HM-B.ST     1.000000   0.423641  0.388743   0.411571  0.442182  0.367955
NDA-SE.ST   0.423641   1.000000  0.401215   0.450142  0.540480  0.512431
SCA-B.ST    0.388743   0.401215  1.000000   0.412885  0.428717  0.422285
SECU-B.ST   0.411571   0.450142  0.412885   1.000000  0.516212  0.498442
SKA-B.ST    0.442182   0.540480  0.428717   0.516212  1.000000  0.536135
```

```
SKF-B.ST   0.367955   0.512431  0.422285   0.498442  0.536135  1.000000
SSAB-B.ST  0.354173   0.452063  0.350681   0.411119  0.459950  0.512576
TEL2-B.ST  0.263693   0.311428  0.289564   0.317447  0.360949  0.313388

Symbols    SSAB-B.ST  TEL2-B.ST
Symbols
ABB.ST      0.483609   0.351977
ASSA-B.ST   0.390880   0.336978
ELUX-B.ST   0.352877   0.262719
EKTA-B.ST   0.235829   0.189015
ERIC-B.ST   0.302891   0.335138
GETI-B.ST   0.174476   0.221433
HM-B.ST     0.354173   0.263693
NDA-SE.ST   0.452063   0.311428
SCA-B.ST    0.350681   0.289564
SECU-B.ST   0.411119   0.317447
SKA-B.ST    0.459950   0.360949
SKF-B.ST    0.512576   0.313388
SSAB-B.ST   1.000000   0.264081
TEL2-B.ST   0.264081   1.000000
```

[50]:
```
# We look at the mean correlation between the assets
print(corr_matrix_box_mean)
```

```
Symbols
ABB.ST       0.493046
ASSA-B.ST    0.473290
ELUX-B.ST    0.404159
EKTA-B.ST    0.312036
ERIC-B.ST    0.371103
GETI-B.ST    0.301572
HM-B.ST      0.391578
NDA-SE.ST    0.454022
SCA-B.ST     0.423918
SECU-B.ST    0.452143
SKA-B.ST     0.477824
SKF-B.ST     0.474533
SSAB-B.ST    0.410372
TEL2-B.ST    0.344129
dtype: float64
```

[51]:
```
# We look at the assets which had the smallest mean correlation between␣
↪eachother.

print(nsmallest(6, corr_matrix_box_mean))
print('[SKA-B.ST,            EKTA-B.ST,           GETI-B.ST,            ERIC-B.
↪ST,         TEL2-B.ST,          HM-B.st            ]')
```

```
[0.3015719006389189, 0.31203558161164124, 0.344129426140454, 0.3711031739792328,
0.3915780588071901, 0.404158515591401]
[SKA-B.ST,            EKTA-B.ST,           GETI-B.ST,           ERIC-B.ST,
TELE2-B.ST,         HM-B.st              ]
```

[15]:
```
tickers = ['SKA-B.ST', 'EKTA-B.ST', 'GETI-B.ST', 'ERIC-B.ST', 'TEL2-B.ST',␣
↪'HM-B.ST']
#stock_data = ext_datareader(tickers, start='2014-10-01', end='2020-10-01')
```

[474]:
```
stock_data.head()
```

[474]:
```
Asset              SKA       EKTA        GETI       ERIC       TEL2  \
2014-10-01  145.800003  70.550003  146.425995  90.400002  83.064201
2014-10-02  144.600006  71.250000  145.455002  89.250000  81.811501
2014-10-03  147.699997  71.050003  147.559998  90.599998  82.486000
2014-10-06  146.600006  71.000000  148.044998  89.500000  83.256897
2014-10-07  144.800003  70.500000  147.317001  86.849998  82.486000

Asset               HM
2014-10-01  294.500000
2014-10-02  288.399994
2014-10-03  291.000000
2014-10-06  290.500000
2014-10-07  286.200012
```

2 Writing a portfolio class and training a LSTM model with the selected stocks.

We create a class because we need the stock-data from every asset within our portfolio. By storing the dowloaded data within the object, we only need to specify our portfolio once, then we can train a model on every individual asset by iterating over the TrainModel method.

[6]:
```
class Portfolio(object):
    def __init__(self, tickers, start, end):
        self.tickers = tickers
        self.start = start
        self.end = end
        self._data = self.ext_datareader() # Runs the ext_datareader when we␣
↪initialize the data_reader


    def ext_datareader(self):
        """Converts pandas-datareader yahoo data to a cleaner format.
        Parameters
        ----------
        raw_data : pd.DataFrame
```

```
            Rows represents different timestamps stored in index. Note that␣
↪there can be gaps. Columns are pd.MultiIndex
            with the zero level being assets and the first level indicator.
        Returns
        --------
        df : pd.DataFrame
            A cleaned pd.DataFrame.

        """
        df = pdr.get_data_yahoo(self.tickers, self.start, self.end)
        df = df.drop(['High', 'Low', 'Open', 'Volume'], axis=1, level=0) #␣
↪drops the columns we don't need
        df.index.name = None # removes the date index
        df = df.swaplevel(0, 1, axis=1) # swaps the multiindex
        df.columns.rename(names = ['Asset', 'Channel'], inplace = True)

        df.replace(0, np.nan, inplace=True)
        df.to_csv('/mnt/c/Users/Anton/Documents/Models/portfolio.csv')
        print('Tickers read...')
        print(f'You have {df.columns.levels[0].tolist()} in your portfolio')
        return df.interpolate().dropna()

    def TrainModel(self, stock, split_ratio = 0.8, batch_size = 5, look_back =␣
↪40, epochs = 50, save_graph=True):
        """Trains a univariate LSTM model with the input given.
        Parameters
        ----------
        stock : pd.DataFrame
            Pandas DataFrame containing stock prices ["Close"] and ['Adj␣
↪Close'].
        split_ratio : float
            How the data should be splitted for testing and training. Default␣
↪is 80 % training and 20 % testing.
        batch_size : integer
            How many batches the model should be trained with.
        look_back : integer
            How many days in the past the model should use to predict the next␣
↪days stock price.
        epochs : integer
            How many times the model will iterate every batch_size.
        save_graph : bool
            True if we want to save the loss and validation graphs for every␣
↪trained model. False if we don't want to save.

        Returns
        --------
        model : tf.keras.Model
            Returns a tf/keras object which has been trained and can be used␣
↪for predictions.
        X_test : ndarray shape (Sample,Timestep,Features)
            Contains the test data for prediction.
        y_test : ndarray shape (Sample,Timestep,Features)
            Validation data
        scaler : sklearn.MinMaxScaler object
            A scaler that is unique for every inputted stock. Used to reverse␣
↪MinMax scaling after prediction.
        """

        # Load the data from the pd.DataFrame
        data = self._data[stock].filter(['Adj Close'])
        dataset = data.values
        training_data_len = math.ceil(len(dataset) * split_ratio)
        scaler = MinMaxScaler(feature_range=(0, 1)) # Transforms features by␣
↪scaling each feature to a given range
        scaled_data = scaler.fit_transform(dataset)

        # Divide the data into training and testing data. Default split ratio␣
↪is 0.8
        X, y = self.processData(scaled_data, look_back)
        X_train, X_test = X[:int(X.shape[0] * split_ratio)], X[int(X.shape[0] *␣
↪split_ratio):]
        y_train, y_test = y[:int(y.shape[0] * split_ratio)], y[int(y.shape[0] *␣
↪split_ratio):]
        pd.DataFrame(X_train).to_csv(f'/mnt/c/Users/Anton/Documents/Models/
↪Train Data/{stock}-X_train.csv', index=False)
        pd.DataFrame(X_test).to_csv(f'/mnt/c/Users/Anton/Documents/Models/Test␣
↪Data/{stock}-X_test.csv', index=False)
        pd.DataFrame(y_train).to_csv(f'/mnt/c/Users/Anton/Documents/Models/
↪Train Data/{stock}-y_train.csv', index=False)
        pd.DataFrame(y_test).to_csv(f'/mnt/c/Users/Anton/Documents/Models/Test␣
↪Data/{stock}-y_test.csv', index=False)

        #Reshape data for (Sample,Timestep,Features) Has to be done because the␣
↪models are very sensitive to input-shape
        X_train = X_train.reshape((X_train.shape[0], X_train.shape[1] ,1))
        X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

        # Create the model
        callback = EarlyStopping(monitor='loss', patience=3) # specifying that␣
↪the training should early stop when the loss has not decreased in three␣
↪ephocs.
        model = self.model(look_back = look_back)
        history = model.fit(X_train, y_train, epochs=epochs,
                            validation_data = (X_test,y_test),
                            shuffle=True, batch_size = batch_size, callbacks =␣
↪[callback]) # training/fitting the model with the previous data

        # ** Saving model-data **
        model.save(f'/mnt/c/Users/Anton/Documents/Models/{stock}')

        if save_graph == True:
            fig = plt.figure()
            plt.title(f'Loss of {stock}')
            plt.xlabel('Epochs', fontsize=14)
            plt.ylabel('Loss', fontsize=14)
            plt.plot(history.history['loss'])
            plt.plot(history.history['val_loss'])
            plt.legend(['loss', 'val_loss'], loc='upper right')
            plt.savefig(f'/mnt/c/Users/Anton/Documents/Models/{stock}/
↪{stock}-loss.png')
            plt.close(fig)
        return model, X_test, y_test, scaler

        #           ** Creating the LSTM network that is used for training. **

    def model(self, look_back):
        """Builds a model for every single stock. Gives the possibility to␣
↪tweak the architecture of the model for indivudual assets.

        LAYER 1 : LSTM: 52 neurons
        LAYER 2 : LSTM: 52 neurons
        LAYER 3 : DENSE : 25 neurons
        LAYER 4 ( OUTPUT ) : 1 neuron

        Optimizer for gradient decent is adam.
        Loss is measured with Mean Squared Error.

        Parameters
        ----------
        look_back : pd.DataFrame
            How many days in the past the model should use to predict the next␣
↪days stock price.
        Returns
        -------
        _model : tf.keras.Model
            An LSTM model that can be trained on given a certain look_back␣
↪period.
        """
        _model = Sequential() # calls on the basemodel Sequential() from the␣
↪keras library. Used for one input one output models.
        _model.add(LSTM(52, return_sequences=True, input_shape=(look_back ,␣
↪1))) # return_sequence has to be true when using multiple LSTM layers
        _model.add(LSTM(52, input_shape=(look_back , 1)))
        _model.add(Dense(25))
        _model.add(Dense(1))
        _model.compile(optimizer='adam', loss='mse')
        return _model

    def processData(self, data, look_back):
        """ Method that processes/splits data into according shape/size.
        Parameters
        ----------
        data : pd.DataFrame
            Cointains stockdata that has been MinMax scaled

        Returns
        -------
        array(X) : ndarray shape (Scaled data, look_back, 0)
            Array of data that has been splitted.
        array(Y) : ndarray shape (Scaled data, look_back, 0)
            Array of data that has been splitted.
        """
        X , Y = [] , []
        for i in range(len(data) - look_back - 1): # the range of the data that␣
↪will be used for the model. -look_back and -1 for offsetting it.
            X.append(data[i:(i + look_back),0])
            Y.append(data[(i + look_back),0])
        return array(X) , array(Y)

    @staticmethod
    def plotter(stock, model, x_test, y_test, scaler):
        """ Method that saves the loss-plots

        """
        Xt = model.predict(x_test)
        fig = plt.figure(figsize=(16,8))
        plt.title(f'LSTM model for {stock}')
        plt.xlabel('Date', fontsize=18)
        plt.ylabel('Adj Close Price SEK', fontsize=18)
        plt.plot(scaler.inverse_transform(y_test.reshape(-1,1)))
        plt.plot(scaler.inverse_transform(Xt))
        plt.legend(['Val', 'Predictions'], loc='lower right')
        plt.savefig(f'/mnt/c/Users/Anton/Documents/Models/{stock}/
↪{stock}-model_graph.png')
        plt.close(fig)
    @staticmethod
    def pred_act(x_test, scaler, y_test, model):
        """Method that saves the predicted and actual stockprices into csv␣
↪files. Index [-1] is the "real" predicted price.
        """
        act = []
        pred = []
        for i in range(294):
            Xt = model.predict(x_test[i].reshape(1,40,1)) # predict the␣
↪stockprices at every i
            pred.append(scaler.inverse_transform(Xt)) # using scaler.inverse to␣
↪reverse the previous scaled data back to stock prices
            act.append(scaler.inverse_transform(y_test[i].reshape(-1,1))) #␣
↪using scaler.inverse to reverse the previous scaled data back to stock prices
        pred = np.array(pred).flatten() # flattening the array to be␣
↪1-dimension.
        act = np.array(act).flatten()
        df_pred = pd.DataFrame(columns = tickers) # creating a pandas DataFrame␣
↪that has tickers as column-names.
        df_act = pd.DataFrame(columns = tickers)
        df_pred[stock] = pred # appending data to the given stock.
        df_act[stock] = act
        df_pred.to_csv(f'/mnt/c/Users/Anton/Documents/Models/{stock}/
↪{stock}-pred.csv', index=False)
        df_act.to_csv(f'/mnt/c/Users/Anton/Documents/Models/{stock}/
↪{stock}-actual.csv', index=False)
        return act, pred
```

[304]:
```
# constructs/initialize the portfolio. this loads all the data for the stocks␣
↪within the portfolio. tickers are given since before.
portfolio = Portfolio(tickers, start = '2013-10-01', end = '2020-10-01')
```

```
Tickers read…
You have ['SKA-B.ST', 'EKTA-B.ST', 'GETI-B.ST', 'ERIC-B.ST', 'TEL2-B.ST',
'HM-B.ST'] in your portfolio
```

[305]:
```
# Iterating through all of our stocks we have in our portfolio. each loop␣
↪trains the model and validates
# it to it's corresponding test-set. Loss, validation and fit is plotted in␣
↪graphs and saved in a
# directory. RMSE is also computed and saved as a text file.

for stock in tickers:
    model, x_test, y_test, scaler = portfolio.TrainModel(stock) # trains the␣
↪model for the specific stock.
```

```
    portfolio.plotter(stock, model, x_test, y_test, scaler) # calls on the␣
↪plotter method to save graohs
    act, prediction = portfolio.pred_act(x_test, scaler, y_test, model) # calls␣
↪on the pred_act method which saves the predicted and actual price into csv
    rmse = np.sqrt(np.mean(((prediction - act)**2))) # computes the rmse of␣
↪every validated model
    with open (f'/mnt/c/Users/Anton/Documents/Models/{stock}/{stock}-rmse.
↪txt','w') as f: # saves the rmse into .txt file.
            f.write(str(rmse)+'\n')
```

```
Epoch 1/50
275/275 [==============================] - 8s 27ms/step - loss: 0.0018 -
val_loss: 0.0033
Epoch 2/50
275/275 [==============================] - 7s 24ms/step - loss: 9.3786e-04 -
val_loss: 0.0039
Epoch 3/50
275/275 [==============================] - 7s 25ms/step - loss: 6.9316e-04 -
val_loss: 0.0025
Epoch 4/50
275/275 [==============================] - 7s 25ms/step - loss: 6.2064e-04 -
val_loss: 0.0016
Epoch 5/50
275/275 [==============================] - 7s 25ms/step - loss: 5.4043e-04 -
val_loss: 0.0013
Epoch 6/50
275/275 [==============================] - 7s 24ms/step - loss: 4.7188e-04 -
val_loss: 0.0012
Epoch 7/50
275/275 [==============================] - 7s 24ms/step - loss: 4.8699e-04 -
val_loss: 0.0014
Epoch 8/50
275/275 [==============================] - 7s 25ms/step - loss: 3.7723e-04 -
val_loss: 8.4694e-04
Epoch 9/50
275/275 [==============================] - 7s 25ms/step - loss: 3.6925e-04 -
val_loss: 0.0012
Epoch 10/50
275/275 [==============================] - 7s 25ms/step - loss: 3.0187e-04 -
val_loss: 7.7443e-04
Epoch 11/50
275/275 [==============================] - 7s 25ms/step - loss: 2.7186e-04 -
val_loss: 7.4171e-04
Epoch 12/50
275/275 [==============================] - 7s 24ms/step - loss: 3.2268e-04 -
val_loss: 7.3508e-04
Epoch 13/50
```

```
275/275 [==============================] - 7s 24ms/step - loss: 2.9433e-04 -
val_loss: 6.8587e-04
Epoch 14/50
275/275 [==============================] - 7s 25ms/step - loss: 2.8191e-04 -
val_loss: 6.6141e-04
INFO:tensorflow:Assets written to:
/mnt/c/Users/Anton/Documents/Models/SKA-B.ST/assets
Epoch 1/50
275/275 [==============================] - 10s 35ms/step - loss: 0.0028 -
val_loss: 0.0058
Epoch 2/50
275/275 [==============================] - 8s 29ms/step - loss: 0.0012 -
val_loss: 0.0031
Epoch 3/50
275/275 [==============================] - 8s 29ms/step - loss: 0.0010 -
val_loss: 0.0037
Epoch 4/50
275/275 [==============================] - 8s 28ms/step - loss: 8.1207e-04 -
val_loss: 0.0017
Epoch 5/50
275/275 [==============================] - 7s 26ms/step - loss: 6.6704e-04 -
val_loss: 0.0014
Epoch 6/50
275/275 [==============================] - 7s 27ms/step - loss: 6.5178e-04 -
val_loss: 0.0013
Epoch 7/50
275/275 [==============================] - 7s 27ms/step - loss: 5.3898e-04 -
val_loss: 0.0012
Epoch 8/50
275/275 [==============================] - 7s 26ms/step - loss: 4.6598e-04 -
val_loss: 0.0012
Epoch 9/50
275/275 [==============================] - 7s 26ms/step - loss: 4.6264e-04 -
val_loss: 0.0012
Epoch 10/50
275/275 [==============================] - 7s 25ms/step - loss: 4.9310e-04 -
val_loss: 0.0013
Epoch 11/50
275/275 [==============================] - 7s 25ms/step - loss: 5.1214e-04 -
val_loss: 0.0012
Epoch 12/50
275/275 [==============================] - 7s 26ms/step - loss: 4.0919e-04 -
val_loss: 0.0012
Epoch 13/50
275/275 [==============================] - 7s 27ms/step - loss: 4.1809e-04 -
val_loss: 0.0012
Epoch 14/50
275/275 [==============================] - 7s 27ms/step - loss: 4.7701e-04 -
```

```
val_loss: 0.0013
Epoch 15/50
275/275 [==============================] - 7s 26ms/step - loss: 3.8519e-04 -
val_loss: 0.0012
Epoch 16/50
275/275 [==============================] - 7s 26ms/step - loss: 4.2742e-04 -
val_loss: 0.0017
Epoch 17/50
275/275 [==============================] - 7s 26ms/step - loss: 4.1189e-04 -
val_loss: 0.0020
Epoch 18/50
275/275 [==============================] - 7s 26ms/step - loss: 3.8857e-04 -
val_loss: 0.0014
INFO:tensorflow:Assets written to:
/mnt/c/Users/Anton/Documents/Models/EKTA-B.ST/assets
Epoch 1/50
275/275 [==============================] - 8s 30ms/step - loss: 0.0051 -
val_loss: 0.0025
Epoch 2/50
275/275 [==============================] - 9s 32ms/step - loss: 0.0013 -
val_loss: 0.0044
Epoch 3/50
275/275 [==============================] - 8s 28ms/step - loss: 0.0011 -
val_loss: 0.0028
Epoch 4/50
275/275 [==============================] - 8s 27ms/step - loss: 9.0557e-04 -
val_loss: 0.0016
Epoch 5/50
275/275 [==============================] - 8s 28ms/step - loss: 7.0457e-04 -
val_loss: 0.0016
Epoch 6/50
275/275 [==============================] - 7s 27ms/step - loss: 6.6411e-04 -
val_loss: 0.0015
Epoch 7/50
275/275 [==============================] - 7s 26ms/step - loss: 6.1096e-04 -
val_loss: 0.0012
Epoch 8/50
275/275 [==============================] - 7s 25ms/step - loss: 5.7832e-04 -
val_loss: 0.0016
Epoch 9/50
275/275 [==============================] - 8s 27ms/step - loss: 6.1342e-04 -
val_loss: 0.0012
Epoch 10/50
275/275 [==============================] - 7s 26ms/step - loss: 5.8724e-04 -
val_loss: 9.3653e-04
Epoch 11/50
275/275 [==============================] - 7s 26ms/step - loss: 5.2465e-04 -
val_loss: 9.8365e-04
```

```
Epoch 12/50
275/275 [==============================] - 7s 26ms/step - loss: 5.1100e-04 -
val_loss: 0.0014
Epoch 13/50
275/275 [==============================] - 7s 25ms/step - loss: 4.9335e-04 -
val_loss: 9.4255e-04
Epoch 14/50
275/275 [==============================] - 8s 29ms/step - loss: 4.5666e-04 -
val_loss: 8.2874e-04
Epoch 15/50
275/275 [==============================] - 8s 28ms/step - loss: 4.2140e-04 -
val_loss: 0.0010
Epoch 16/50
275/275 [==============================] - 9s 33ms/step - loss: 4.5728e-04 -
val_loss: 9.8850e-04
Epoch 17/50
275/275 [==============================] - 10s 36ms/step - loss: 4.4245e-04 -
val_loss: 8.4596e-04
Epoch 18/50
275/275 [==============================] - 9s 32ms/step - loss: 4.2183e-04 -
val_loss: 8.4633e-04
INFO:tensorflow:Assets written to:
/mnt/c/Users/Anton/Documents/Models/GETI-B.ST/assets
Epoch 1/50
275/275 [==============================] - 8s 29ms/step - loss: 0.0057 -
val_loss: 0.0034
Epoch 2/50
275/275 [==============================] - 7s 27ms/step - loss: 0.0017 -
val_loss: 0.0058
Epoch 3/50
275/275 [==============================] - 8s 28ms/step - loss: 0.0013 -
val_loss: 0.0025
Epoch 4/50
275/275 [==============================] - 8s 28ms/step - loss: 0.0011 -
val_loss: 0.0016
Epoch 5/50
275/275 [==============================] - 8s 29ms/step - loss: 9.2067e-04 -
val_loss: 0.0015
Epoch 6/50
275/275 [==============================] - 7s 27ms/step - loss: 7.9414e-04 -
val_loss: 0.0028
Epoch 7/50
275/275 [==============================] - 7s 26ms/step - loss: 7.0193e-04 -
val_loss: 0.0024
Epoch 8/50
275/275 [==============================] - 7s 27ms/step - loss: 7.1975e-04 -
val_loss: 0.0015
Epoch 9/50
```

```
275/275 [==============================] - 8s 29ms/step - loss: 6.6426e-04 -
val_loss: 0.0017
Epoch 10/50
275/275 [==============================] - 7s 27ms/step - loss: 6.4111e-04 -
val_loss: 0.0011
Epoch 11/50
275/275 [==============================] - 7s 27ms/step - loss: 5.7573e-04 -
val_loss: 0.0017
Epoch 12/50
275/275 [==============================] - 7s 26ms/step - loss: 5.3531e-04 -
val_loss: 0.0018
Epoch 13/50
275/275 [==============================] - 7s 27ms/step - loss: 5.6650e-04 -
val_loss: 9.3250e-04
Epoch 14/50
275/275 [==============================] - 7s 26ms/step - loss: 5.1804e-04 -
val_loss: 9.3375e-04
Epoch 15/50
275/275 [==============================] - 7s 26ms/step - loss: 5.5759e-04 -
val_loss: 9.2727e-04
Epoch 16/50
275/275 [==============================] - 7s 26ms/step - loss: 6.6714e-04 -
val_loss: 9.4876e-04
Epoch 17/50
275/275 [==============================] - 7s 26ms/step - loss: 5.3791e-04 -
val_loss: 9.1310e-04
INFO:tensorflow:Assets written to:
/mnt/c/Users/Anton/Documents/Models/ERIC-B.ST/assets
Epoch 1/50
275/275 [==============================] - 8s 28ms/step - loss: 0.0022 -
val_loss: 0.0040
Epoch 2/50
275/275 [==============================] - 8s 28ms/step - loss: 5.6583e-04 -
val_loss: 0.0025
Epoch 3/50
275/275 [==============================] - 8s 28ms/step - loss: 4.8780e-04 -
val_loss: 0.0022
Epoch 4/50
275/275 [==============================] - 7s 26ms/step - loss: 4.0940e-04 -
val_loss: 0.0015
Epoch 5/50
275/275 [==============================] - 7s 26ms/step - loss: 4.1855e-04 -
val_loss: 0.0018
Epoch 6/50
275/275 [==============================] - 7s 26ms/step - loss: 3.3062e-04 -
val_loss: 0.0011
Epoch 7/50
275/275 [==============================] - 7s 25ms/step - loss: 3.3786e-04 -
```

```
val_loss: 9.3202e-04
Epoch 8/50
275/275 [==============================] - 7s 25ms/step - loss: 2.8845e-04 -
val_loss: 0.0011
Epoch 9/50
275/275 [==============================] - 7s 25ms/step - loss: 2.4722e-04 -
val_loss: 0.0019
Epoch 10/50
275/275 [==============================] - 7s 25ms/step - loss: 2.6036e-04 -
val_loss: 5.2858e-04
Epoch 11/50
275/275 [==============================] - 7s 25ms/step - loss: 2.1735e-04 -
val_loss: 4.5902e-04
Epoch 12/50
275/275 [==============================] - 8s 29ms/step - loss: 2.1232e-04 -
val_loss: 5.6423e-04
Epoch 13/50
275/275 [==============================] - 8s 30ms/step - loss: 2.5864e-04 -
val_loss: 4.3635e-04
Epoch 14/50
275/275 [==============================] - 9s 31ms/step - loss: 1.7878e-04 -
val_loss: 0.0016
Epoch 15/50
275/275 [==============================] - 8s 28ms/step - loss: 2.6403e-04 -
val_loss: 5.1601e-04
Epoch 16/50
275/275 [==============================] - 8s 28ms/step - loss: 1.6488e-04 -
val_loss: 7.5435e-04
Epoch 17/50
275/275 [==============================] - 9s 32ms/step - loss: 1.7594e-04 -
val_loss: 4.3220e-04
Epoch 18/50
275/275 [==============================] - 8s 28ms/step - loss: 1.7323e-04 -
val_loss: 8.2670e-04
Epoch 19/50
275/275 [==============================] - 9s 33ms/step - loss: 1.7044e-04 -
val_loss: 7.6830e-04
INFO:tensorflow:Assets written to:
/mnt/c/Users/Anton/Documents/Models/TEL2-B.ST/assets
Epoch 1/50
275/275 [==============================] - 8s 30ms/step - loss: 0.0063 -
val_loss: 0.0029
Epoch 2/50
275/275 [==============================] - 7s 26ms/step - loss: 0.0014 -
val_loss: 0.0021
Epoch 3/50
275/275 [==============================] - 7s 25ms/step - loss: 0.0011 -
val_loss: 0.0017
```

```
Epoch 4/50
275/275 [==============================] - 7s 25ms/step - loss: 8.9597e-04 -
val_loss: 0.0016
Epoch 5/50
275/275 [==============================] - 8s 29ms/step - loss: 9.1881e-04 -
val_loss: 0.0021
Epoch 6/50
275/275 [==============================] - 8s 30ms/step - loss: 8.0175e-04 -
val_loss: 0.0013
Epoch 7/50
275/275 [==============================] - 8s 28ms/step - loss: 5.9578e-04 -
val_loss: 0.0011
Epoch 8/50
275/275 [==============================] - 8s 27ms/step - loss: 6.2568e-04 -
val_loss: 9.5404e-04
Epoch 9/50
275/275 [==============================] - 8s 27ms/step - loss: 5.8060e-04 -
val_loss: 8.9010e-04
Epoch 10/50
275/275 [==============================] - 7s 26ms/step - loss: 5.1552e-04 -
val_loss: 8.0673e-04
Epoch 11/50
275/275 [==============================] - 7s 26ms/step - loss: 5.9435e-04 -
val_loss: 9.4311e-04
Epoch 12/50
275/275 [==============================] - 7s 26ms/step - loss: 4.8592e-04 -
val_loss: 8.0744e-04
Epoch 13/50
275/275 [==============================] - 8s 28ms/step - loss: 3.9887e-04 -
val_loss: 6.7193e-04
Epoch 14/50
275/275 [==============================] - 8s 27ms/step - loss: 4.3029e-04 -
val_loss: 6.6171e-04
Epoch 15/50
275/275 [==============================] - 8s 29ms/step - loss: 3.9004e-04 -
val_loss: 6.6326e-04
Epoch 16/50
275/275 [==============================] - 10s 35ms/step - loss: 4.4947e-04 -
val_loss: 7.3528e-04
Epoch 17/50
275/275 [==============================] - 8s 30ms/step - loss: 4.3551e-04 -
val_loss: 6.1346e-04
Epoch 18/50
275/275 [==============================] - 8s 29ms/step - loss: 3.9355e-04 -
val_loss: 6.0655e-04
INFO:tensorflow:Assets written to:
/mnt/c/Users/Anton/Documents/Models/HM-B.ST/assets
```

3 Calculating returns and covariances

[7]:
```
# creates a function that loops over the stocks in the portfolio and reads in␣
↪the stocks actual / predicted data.
tickers = ['SKA-B.ST', 'EKTA-B.ST', 'GETI-B.ST', 'ERIC-B.ST', 'TEL2-B.ST',␣
↪'HM-B.ST']
def concat_df(type_ = ''):
    df = pd.DataFrame(columns = tickers)
    for stock in tickers:
        df[stock] = pd.read_csv(f'{save_path}/{stock}/{stock}-{type_}.
↪csv')[stock]
    return df
```

[8]:
```
actual = concat_df(type_ = 'actual') # saves the portfolio stocks actual stock␣
↪price in a dataframe
unknown_price = actual.iloc[-1] # allocates the real price that is unknown to␣
↪the model into a variable. comparable price
actual = actual[:-1] # all the real stock prices to day t
actual_last = actual.iloc[-1] # getting the time t stock price.
```

[9]:
```
actual_ret = (unknown_price - actual_last) / actual_last
```

[10]:
```
predicted = concat_df(type_='pred') # saves the potfolio stocks predicted stock␣
↪price in a dataframe
pred_price = predicted.iloc[-1] # the t+1 stock price
```

[11]:
```
# the returns should probably be calculated as log_return. it however␣
↪complicates the caluculation of the return for t+1 thus normal return has␣
↪been calculated.
# just showing below how the log returns and covariance matrix with log can be␣
↪calulated.

#log_actual = np.log(actual / actual.shift(1))
#log_actual_price = log_actual.iloc[-2]
#test_cov = log_actual.cov() # testing cov of actual
#log_pred = np.log(predicted / predicted.shift(1))
#log_pred_price = log_pred.iloc[-1]
```

3.0.1 Calulating the return for t –> t+1 with the predicted price at t+1

$E(R_A)_{t+1} = \frac{PP^A_{t+1} - P^A_t}{P^A_t}$

$E(R_A)_{t+1}$ = unweighted expected return of a single asset at time t+1

PP^A_{t+1} = predicted for a specific asset price at t+1

P^A_t = real price of asset at time t

[12]:
```
# calculating the returns of all the stocks in our portfolio. this is done with␣
↪the do

def calc_return():
    returns = []

    for stock in tickers:
        returns.append((pred_price[stock] - actual_last[stock]) /␣
↪actual_last[stock])

    df = pd.Series(returns, index = tickers, name = 'Predicted returns')
    return df
```

[13]:
```
pred_ret = calc_return()
```

[14]:
```
# the means/expected returns for t+1
pred_ret
```

[14]:
```
SKA-B.ST     0.003109
EKTA-B.ST    0.016434
GETI-B.ST   -0.005667
ERIC-B.ST   -0.000773
TEL2-B.ST    0.017092
HM-B.ST      0.006105
Name: Predicted returns, dtype: float64
```

3.0.2 Calculating covariance matrix of portfolio

$cov(X,Y) = \sum_{i=0}^{N} = \frac{(R_{X_i} - \bar{R}_X) - (R_{Y_i} - \bar{R}_Y)}{N-1}$

[15]:
```
# defining a function that returns the covariance matrix of all the assets.

def covariance_matrix(bias = False):
    matrix = np.zeros([len(tickers), len(actual) - 1])
    i = 0
    for stocks in tickers: # iterating over both the stocks and the created␣
↪matrix.
        numerator = actual[stocks].pct_change().dropna() - pred_ret[stocks] #␣
↪calculating the numerator of the function. subtracting the predicted mean/
↪return.
        matrix[i] = numerator # adding the numerator to the previos created␣
↪matrix.
        i += 1
    daily_cov = np.cov(matrix, bias = bias) # using np.cov to compute the␣
↪coveriance matrix of the whole portfolio. bias is used for N - 1.
    print('Daily covariance for the portfolio stocks are: ')
    print('')
    print(pd.DataFrame(daily_cov, columns=[tickers], index=[tickers]))
    return daily_cov
```

[16]:
```
daily_cov = covariance_matrix()
```

```
Daily covariance for the portfolio stocks are:

           SKA-B.ST EKTA-B.ST GETI-B.ST ERIC-B.ST TEL2-B.ST   HM-B.ST
SKA-B.ST   0.000451  0.000248  0.000096  0.000221  0.000157  0.000355
EKTA-B.ST  0.000248  0.000879  0.000184  0.000216  0.000113  0.000349
GETI-B.ST  0.000096  0.000184  0.000626  0.000200  0.000142  0.000026
ERIC-B.ST  0.000221  0.000216  0.000200  0.000591  0.000185  0.000252
TEL2-B.ST  0.000157  0.000113  0.000142  0.000185  0.000316  0.000165
HM-B.ST    0.000355  0.000349  0.000026  0.000252  0.000165  0.000823
```

4 Optimizing the portfolio

We create a class that will be used for optimizing the portfolio. The class has methods that can be used to optimize the portfolio for maximum sharpe ratio and minimum variance. There are also methods for benchmarking the predicted returns from the LSTM model.

These are 1overN which is a simple weight allocation of 1 / number of assets, giving a equal distribution of weights between the assets.

The other one is a random optimizer which only takes random weights from a normal distribution which has to add up to one and allocates these weights to the different assets in the portfolio.

There is also a method for plotting the efficient frontier of the different allocations. This is based on Monte-Carlo simulation. It's only used for visualization.

Unfortunately, we didn't manage to compute a minimum variance portfolio. No matter what we did, the weights were always the same as the initial guess. It might have something to do with that the portfolio volatility and returns is not scaled to 256 days(1 year). This makes the covariances extremely small and there is probably a very small difference in volatility between different portfolios.

[360]:
```
class portfolio_optimizer(object):
    def __init__(self, mean, daily_cov, log_ret = False):
        self.cons = ({'type': 'eq', 'fun': lambda x:  np.sum(x) - 1}) # setting␣
 ↪up the constraint for the opmitization. total sum of weights can only be 1
        self.bnds = tuple((0, 1) for x in range(len(tickers)))
        self.initialGuess = np.ones(len(tickers))*(1./len(tickers)) # intital␣
 ↪guess of weights for the optimizer to start from
        self.mean = mean # the calculated means/returns from the assets between␣
 ↪t --> t+1
        self.daily_cov = daily_cov # the calculated covariance from the␣
 ↪portfolio
```

```
    # method that returns the portfolio returns, volatility and sharpe-ratio␣
↪given inputted weights.
    def portfolio_stats(self, weights):
        weights = np.array(weights)
        pret = np.sum(self.mean * weights) # portfolio returns
        pvol = np.sqrt(np.dot(weights.T, np.dot(self.daily_cov, weights))) #␣
↪portfolio volatility
        return np.array([pret, pvol, pret / pvol])

    # function that returns weights for a minimized negative sharpe ratio
    def min_func_sharpe(self, weights):
        return -self.portfolio_stats(weights)[2]

    # function that returns weights for a minimized variance
    def min_func_variance(self, weights):
        return self.portfolio_stats(weights)[1] ** 2

    # function that return weights for portfolios with least standard devation
    def min_func_port(self, weights):
        return self.portfolio_stats(weights)[1]

    # method that minimizes the negative sharpe and variance.
    def optimizer(self, no_info = False):
        max_sharpe = sco.minimize(self.min_func_sharpe, self.initialGuess,␣
↪method='SLSQP',
                            bounds=self.bnds, constraints=self.cons)
        min_variance = sco.minimize(self.min_func_variance, self.initialGuess,␣
↪method='SLSQP',
                            bounds=self.bnds, constraints=self.cons)

        if no_info == False:
            print(f"Max Sharpe {max_sharpe['message']}")
            print('')
            print('The return, volatility and Sharpe-Ratio are:', self.
↪portfolio_stats(max_sharpe['x']).round(3))
            print('')
            print(pd.DataFrame(max_sharpe['x'].round(3), index=[tickers],␣
↪columns = ['Maximum Sharpe Weights']))
            print('')
            print(f"Min Variance {min_variance['message']}")
            print('')
            print('The return, volatility and Sharpe-Ratio are:', self.
↪portfolio_stats(min_variance['x']).round(3))
            print('')
            print(pd.DataFrame(min_variance['x'].round(3), index=[tickers],␣
↪columns = ['Minimum Variance Weights']))
```

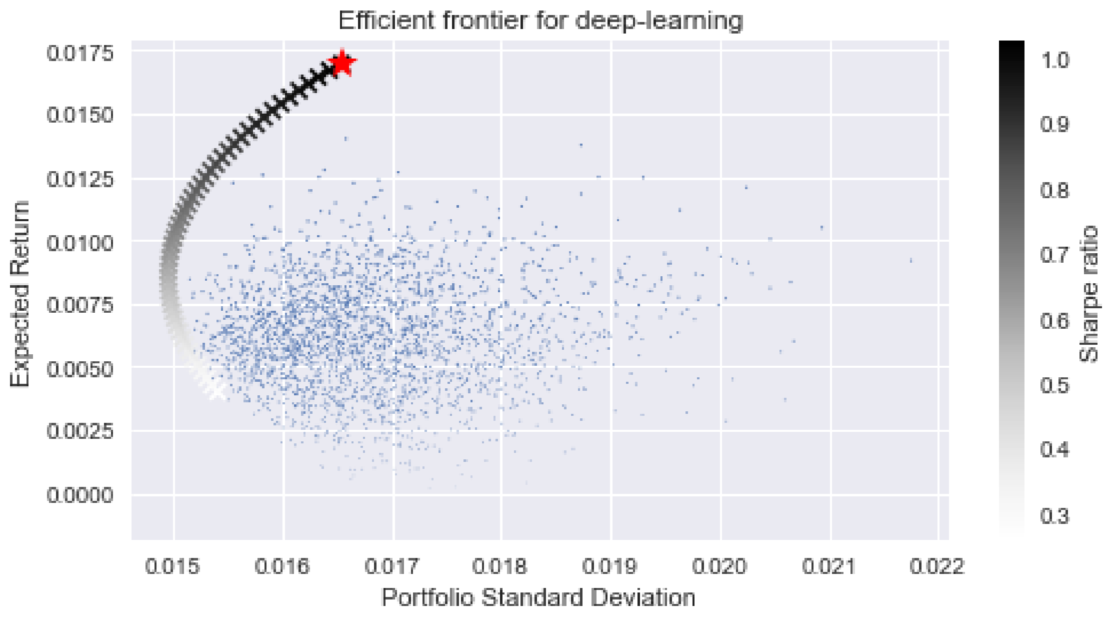
```
        else:
            pass
        return self.portfolio_stats(max_sharpe['x'].round(3))

    def one_over_n(self, no_info = False):
        """
        Simple N over 1 allocation to benchmark the deep-learning performance␣
↪with

        Returns: return, volatility and sharpe ratio
        """
        weights = np.ones(len(tickers)) * (1./len(tickers))
        if no_info == False:
            print('The return, volatility and Sharpe-Ratio are:', self.
↪portfolio_stats(weights.round(3)))
            print('')
        else:
            pass
        return self.portfolio_stats(weights)

    def random_optimizer(self, no_info = False):
        """
        Creates a random weight allocation of the portfolio without any␣
↪knowledge about the past, present
        or future. Returns the return, volatity and sharpe ratio.
        """
        np.random.seed(200)
        weights = np.random.random(len(tickers))
        weights /= np.sum(weights)
        if no_info == False:
            print('The return, volatility and Sharpe-Ratio are:', self.
↪portfolio_stats(weights.round(3)))
            print('')
        else:
            pass
        return self.portfolio_stats(weights)


    def graph_efficient_frontier(self):
        """
        Monte Carlo Simulation in order to produce a efficient frontier. The␣
↪same structure as above
        methods.

        """
        max_sharpe = sco.minimize(self.min_func_sharpe, self.initialGuess,␣
↪method='SLSQP',
                        bounds=self.bnds, constraints=self.cons)
        max_ret = self.portfolio_stats(max_sharpe['x'])[0]
        trets = np.linspace(0.004, max_ret, 50)
        tvols = []
        for tret in trets:
            cons = ({'type': 'eq', 'fun': lambda x:  self.portfolio_stats(x)[0]␣
↪- tret},
            {'type': 'eq', 'fun': lambda x:  np.sum(x) - 1})
            res = sco.minimize(self.min_func_port, self.initialGuess,␣
↪method='SLSQP',
                        bounds=self.bnds, constraints=cons)

            tvols.append(res['fun']) #the value of the objective, i.e. standard␣
↪deviation of portfolio returns
        tvols = np.array(tvols)

        # prepare lists for portfolio returns and volatilities
        prets = []
        pvols = []
        pSharpe =[]

        # randomly generate 2500 portfolios
        for p in range (2500):
            weights = np.random.random(len(tickers))
            weights /= np.sum(weights)

            # portfolio return
            ret = np.sum(self.mean * weights)
            #portfolio volatility
            vol = np.sqrt(np.dot(weights,
                         np.dot(self.daily_cov, weights.T)))

            prets.append(ret)
            pvols.append(vol)
            #portfolio Sharpe ratio
            sharpe = ret/vol
            pSharpe.append(sharpe)

        plt.figure(figsize=(8, 4))
        plt.scatter(pvols, prets, pSharpe, marker='o')
            # random portfolio composition
        plt.scatter(tvols, trets, c = trets/tvols, marker='x')
            # efficient frontier
        plt.plot(self.portfolio_stats(max_sharpe['x'])[1], self.
↪portfolio_stats(max_sharpe['x'])[0],'r*', markersize=15.0)
```

```
        # portfolio with highest Sharpe ratio
    #plt.plot(portfolio_stats(optv['x'])[1], portfolio_stats(optv['x'])[0],
    #          'y*', markersize=15.0)
        # minimum variance portfolio
    #plt.grid(True)
    plt.xlabel('Portfolio Standard Deviation')
    plt.ylabel('Expected Return')
    plt.colorbar(label='Sharpe ratio')
    plt.title('Efficient frontier for deep-learning')
```

[361]:
```
pred_portfolio = portfolio_optimizer(pred_ret, daily_cov)
```

5 Results

[362]:
```
pred_portfolio.graph_efficient_frontier()
```



[363]:
```
deep_learning = pred_portfolio.optimizer()
```

```
Max Sharpe Optimization terminated successfully

The return, volatility and Sharpe-Ratio are: [0.017 0.017 1.025]

           Maximum Sharpe Weights
SKA-B.ST                    0.000
EKTA-B.ST                   0.199
GETI-B.ST                   0.000
```

```
ERIC-B.ST                   0.000
TEL2-B.ST                   0.801
HM-B.ST                     0.000

Min Variance Optimization terminated successfully

The return, volatility and Sharpe-Ratio are: [0.006 0.016 0.372]

           Minimum Variance Weights
SKA-B.ST                      0.167
EKTA-B.ST                     0.167
GETI-B.ST                     0.167
ERIC-B.ST                     0.167
TEL2-B.ST                     0.167
HM-B.ST                       0.167
```

5.0.1 Graphs of the performance of the LSTM prediction when applied with portfolio optimization

[364]:
```
deeplearning = pred_portfolio.optimizer(no_info=True)
```

[365]:
```
one_over_n = pred_portfolio.one_over_n(no_info=False)
```

```
The return, volatility and Sharpe-Ratio are: [0.00606205 0.01628103 0.37233783]
```

[366]:
```
random_weights = pred_portfolio.random_optimizer(no_info=False)
```

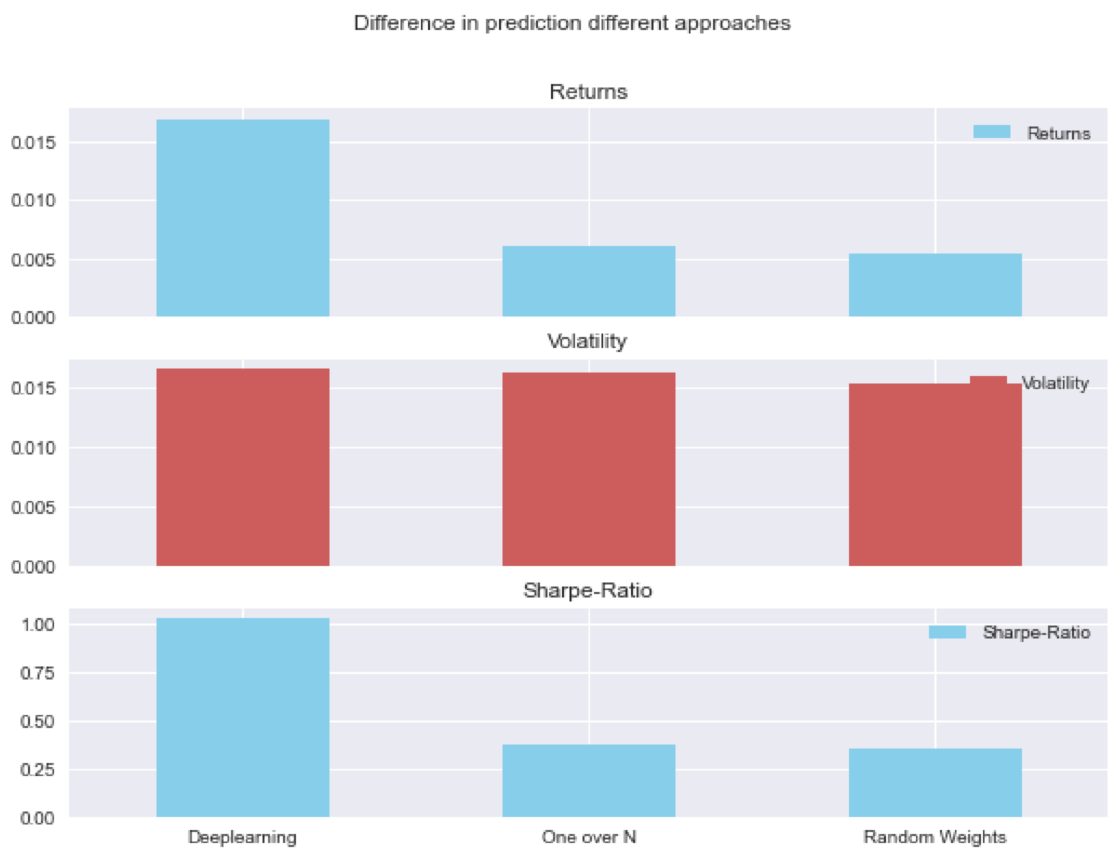
```
The return, volatility and Sharpe-Ratio are: [0.00540843 0.01537108 0.35185777]
```

[367]:
```
df1 = pd.DataFrame([deeplearning, one_over_n, random_weights],
                   columns = ['Returns', 'Volatility', 'Sharpe-Ratio'],
                   index = ['Deeplearning', 'One over N', 'Random Weights'])
```

[368]:
```
df1
```

[368]:
```
                 Returns  Volatility  Sharpe-Ratio
Deeplearning    0.016961    0.016542      1.025279
One over N      0.006050    0.016249      0.372338
Random Weights  0.005414    0.015355      0.352599
```

[369]:
```
ax = df1.plot.bar(color=["SkyBlue","IndianRed"], rot=0,
                 title="Difference in prediction different approaches",␣
 ↪subplots = True, figsize = (10, 7))
plt.show()
```

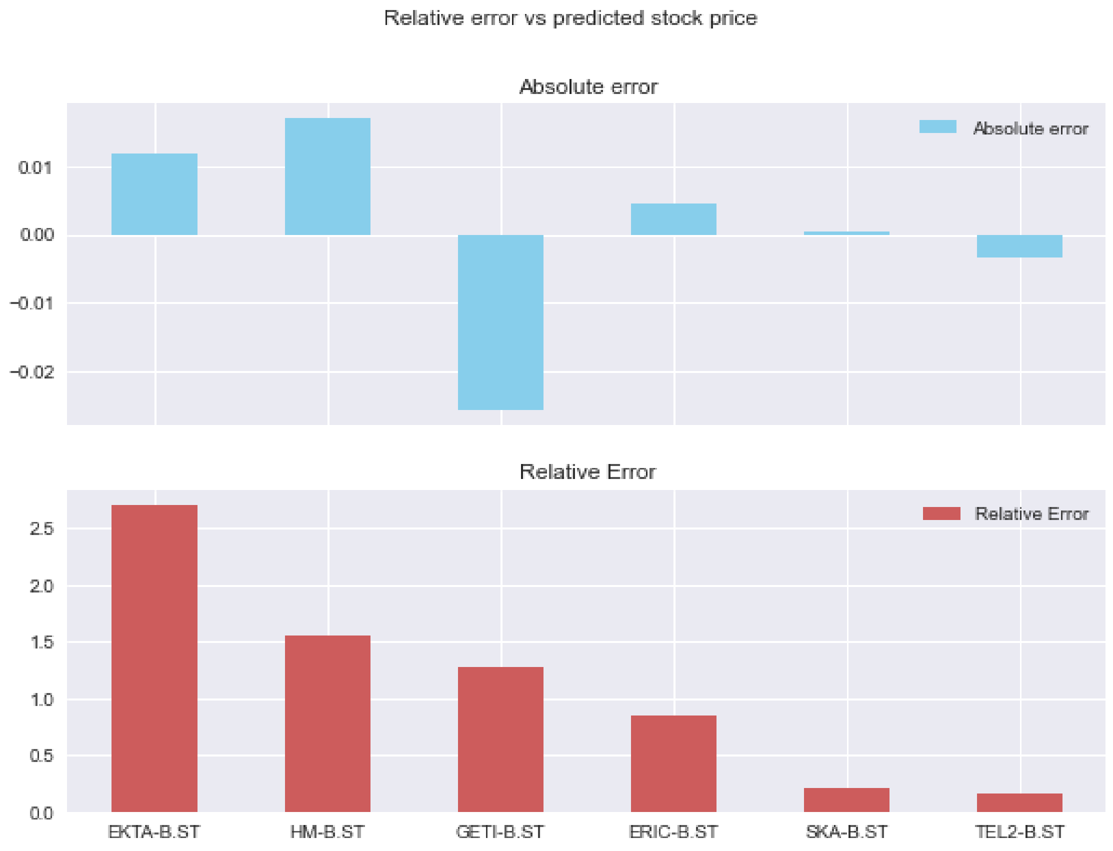


[370]:
```
# Error in price prediction

abs_error = pred_ret - actual_ret
print("Absolute error: ")
print(abs_error)
print("")
rel_error = abs(abs_error / actual_ret)
print("Relative error: ")
print(rel_error)
```

```
Absolute error:
SKA-B.ST     0.000530
EKTA-B.ST    0.011994
GETI-B.ST   -0.025894
ERIC-B.ST    0.004508
TEL2-B.ST   -0.003427
HM-B.ST      0.017062
dtype: float64

Relative error:
SKA-B.ST     0.205601
```

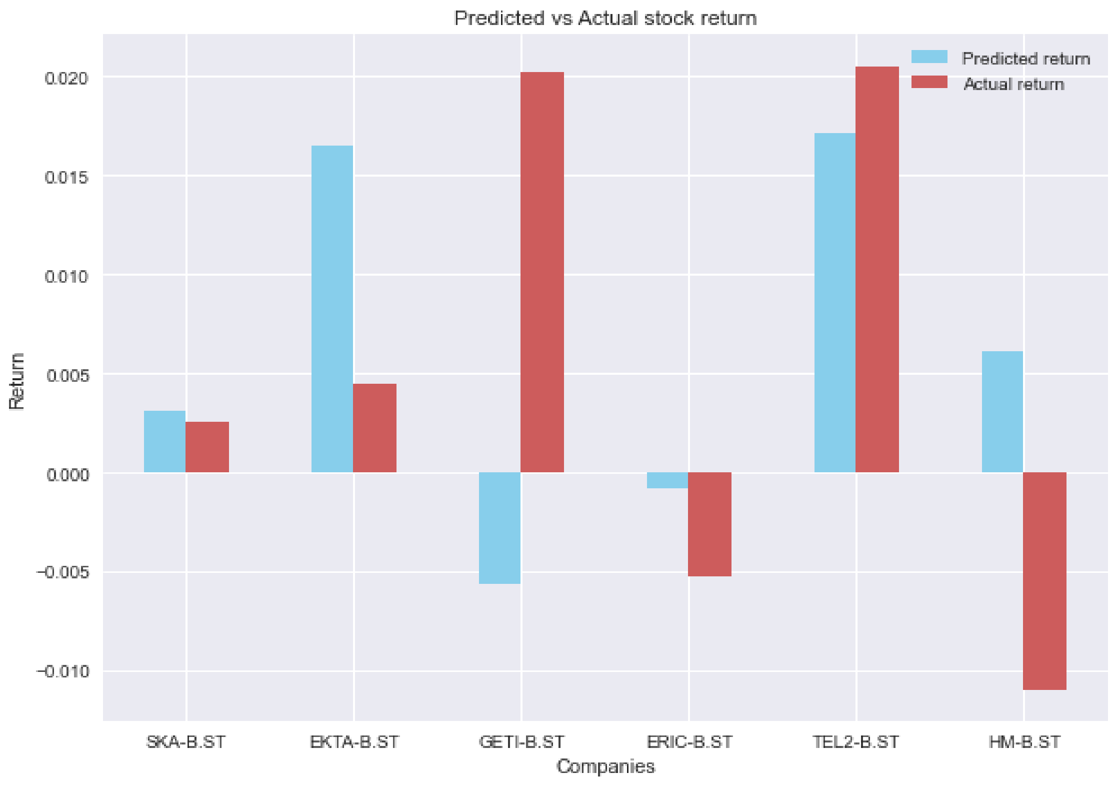
```
EKTA-B.ST    2.701826
GETI-B.ST    1.280165
ERIC-B.ST    0.853649
TEL2-B.ST    0.167029
HM-B.ST      1.557211
dtype: float64
```

[371]:
```
df = pd.DataFrame({"Absolute error":abs_error, "Relative Error":rel_error})
sorted_df = df.sort_values(by = ['Relative Error'], ascending=False)
ax = sorted_df.plot.bar(color=["SkyBlue","IndianRed", "Blueviolet",␣
 ↪"Darkcyan"], rot=0, title="Relative error vs predicted stock price",␣
 ↪subplots = True, figsize = (10, 7))
plt.show()
```



[372]:
```
df = pd.DataFrame({"Predicted return":pred_ret,"Actual return":actual_ret})
sorted_df = df.sort_values(by = ['Predicted return'], ascending=False)
ax = df.plot.bar(color=["SkyBlue","IndianRed"], rot=0, title="Predicted vs␣
 ↪Actual stock return", figsize = (10, 7))
ax.set_xlabel("Companies")
ax.set_ylabel("Return")
```

```
plt.show()
```



5.0.2 Optimizing with the real stock return to compare the predicted performance with the real

[373]:
```
actual_ret = (unknown_price - actual_last) / actual_last
```

[374]:
```
actual_portfolio = portfolio_optimizer(actual_ret, daily_cov)
```

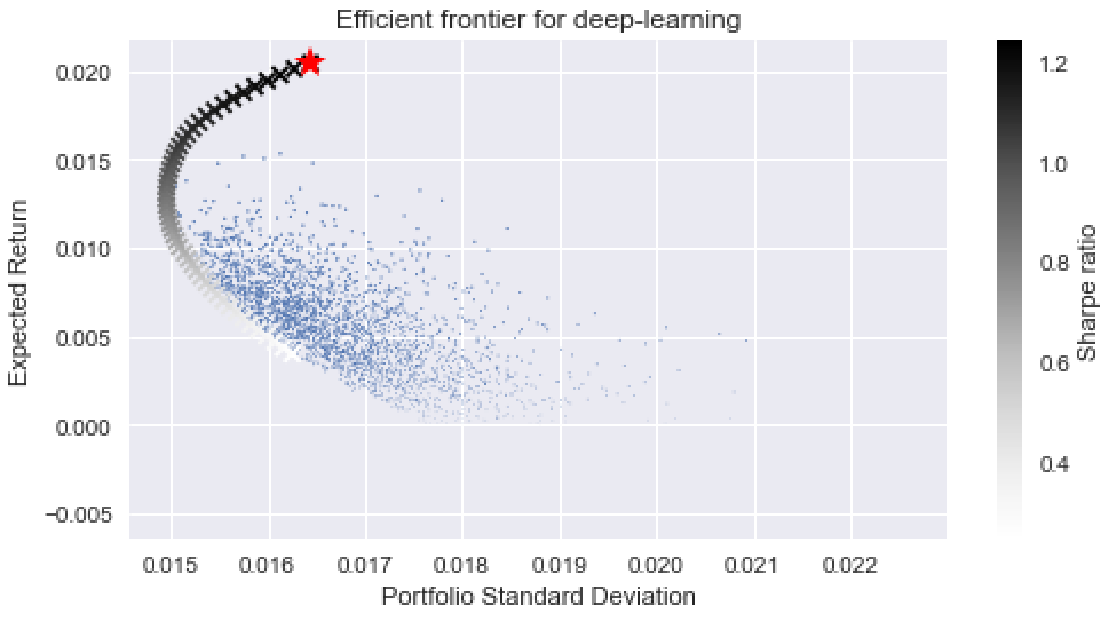
[375]:
```
no_pred = actual_portfolio.optimizer()
```

```
Max Sharpe Optimization terminated successfully

The return, volatility and Sharpe-Ratio are: [0.02  0.016 1.244]

           Maximum Sharpe Weights
SKA-B.ST                    0.000
EKTA-B.ST                   0.000
GETI-B.ST                   0.259
ERIC-B.ST                   0.000
TEL2-B.ST                   0.741
HM-B.ST                     0.000
```

```
Min Variance Optimization terminated successfully

The return, volatility and Sharpe-Ratio are: [0.005 0.016 0.323]

           Minimum Variance Weights
SKA-B.ST                      0.167
EKTA-B.ST                     0.167
GETI-B.ST                     0.167
ERIC-B.ST                     0.167
TEL2-B.ST                     0.167
HM-B.ST                       0.167
```

[376]:
```
actual_portfolio.graph_efficient_frontier()
```

```
/Users/antonerlandsson/.local/lib/python3.8/site-
packages/matplotlib/collections.py:922: RuntimeWarning: invalid value
encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```



[377]:
```
no_pred = actual_portfolio.optimizer(no_info=True)
```

[378]:
```
one_over_n_nopred = actual_portfolio.one_over_n(no_info=False)
```

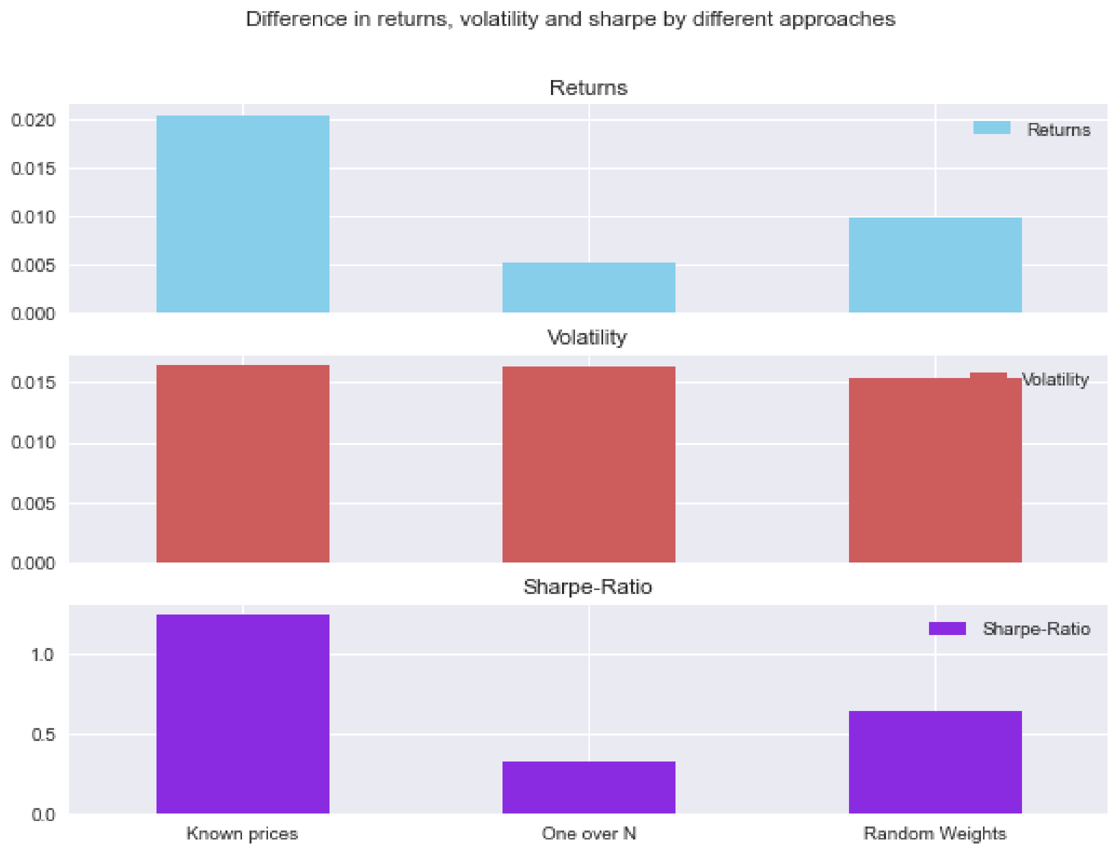
```
The return, volatility and Sharpe-Ratio are: [0.00526486 0.01628103 0.32337356]
```

[379]:
```
random_weights_nopred = actual_portfolio.random_optimizer(no_info=False)
```

```
The return, volatility and Sharpe-Ratio are: [0.00974538 0.01537108 0.63400753]
```

[380]:
```
df2 = pd.DataFrame([no_pred, one_over_n_nopred, random_weights_nopred],
                  columns = ['Returns', 'Volatility', 'Sharpe-Ratio'],
                  index = ['Known prices', 'One over N', 'Random Weights'])
```

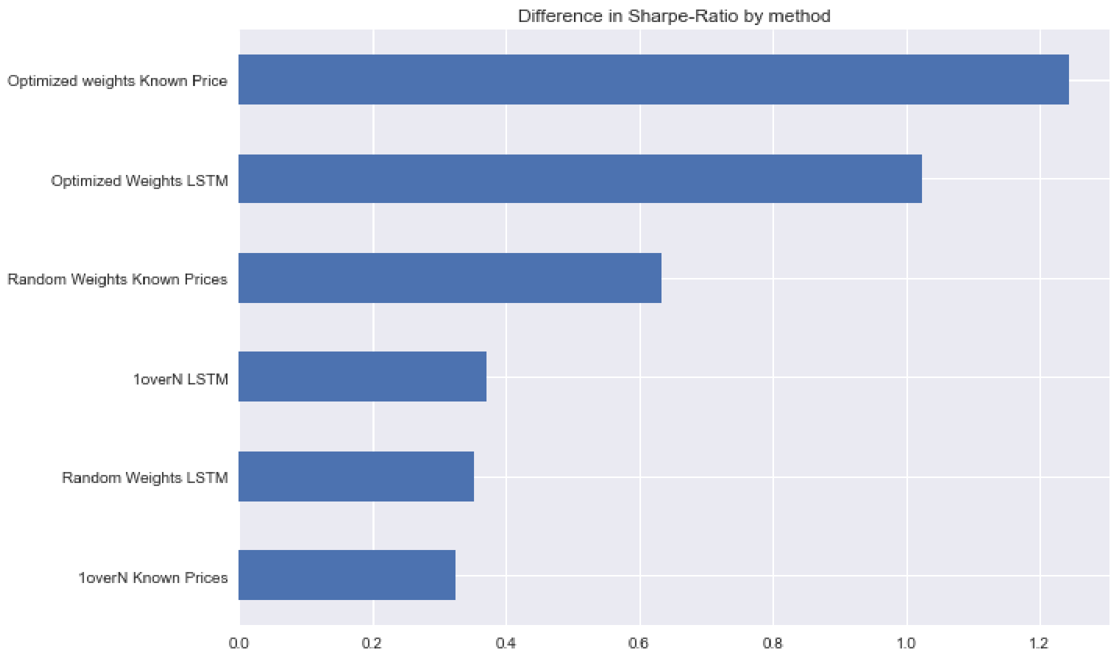
[381]:
```
ax = df2.plot.bar(color=["SkyBlue","IndianRed", "Blueviolet"], rot=0,
                 title="Difference in returns, volatility and sharpe by␣
 ↪different approaches", subplots = True, figsize = (10, 7))

plt.show()
```



5.0.3 Showing the difference between “approaches”

```
[382]: df3 = pd.DataFrame([no_pred, one_over_n_nopred, random_weights_nopred,␣
        ↪deep_learning, one_over_n, random_weights],
                          columns = ['Returns', 'Volatility', 'Sharpe-Ratio'],
                          index = ['Optimized weights Known Price', '1overN Known␣
        ↪Prices', 'Random Weights Known Prices', 'Optimized Weights LSTM', '1overN␣
        ↪LSTM', 'Random Weights LSTM'])
```

```
[383]: test = df3.sort_values(by = ['Sharpe-Ratio'], axis = 0, ascending = True)

       ax = test['Sharpe-Ratio'].plot.barh(rot=0,
                         title="Difference in Sharpe-Ratio by method", figsize = (10,␣
        ↪7))
```


Difference in Sharpe-Ratio by method
Optimized weights Known Price
Optimized Weights LSTM
Random Weights Known Prices
1overN LSTM
Random Weights LSTM
1overN Known Prices