University of Sussex

Portfolio Management and Comparison By Usage of Importance Distributions Dissertation for Monte Carlo Simulations

Uğur Eren Çanakçı*

School of Mathematical and Physical Sciences, University of Sussex
Mathematics with Mathematics BSc
Date: 17.05.2024

Abstract

This paper primarily explores the concept of importance sampling within the context of financial simulations. Importance sampling is a statistical technique used to enhance the efficiency and accuracy of simulations, particularly when dealing with rare events or distributions that are difficult to sample directly. By focusing on the theoretical underpinnings and practical applications of importance sampling, this study aims to demonstrate its utility and effectiveness in financial modeling and risk assessment. Importance distributions are formed by fitting mathematical models on stock values, simulating future values for the same stocks and finally converting these results into probability distributions to be used while sampling stock produces. Results prove the elligibility of model-fitting in financial simulations and projections to gain substantial profits on quick trade schemes or to strengthen the portfolios one holds in their hand.

Introduction

Importance sampling is a technique for sampling from a state space. It works by subjectively modifying the default probability density/mass function, either because of certain beliefs and intuitions, or because of statistical inference on existing records, if available. This dissertation will give examples for this technique over building a financial portfolio, made out of some financial assets, namely long/short positions. On the side of statistical inference, models will be fitted over existing data with the assumption that no "disrupting" events will occur in the time horizon. Models applied in this paper are autoregression on an asset's returns over a day and log-normal distribution model on an asset's ratio between two consecutive day's values. After the models are fitted, future ratios and returns are simulated to be used while creating various importance distributions.

For time series analysis, I took help from the book "Tsay, R. S. (2010). Analysis of financial time series (3rd ed.). Wiley.", which belongs in the reading list for the module "Financial Investment and Corporate Risk Analysis". I learned about the log-normal distribution while studying the Black-Scholes Equation in the "Mathematical Models in Finance and Industry" module. All the simulations and statistical analysis is done on R, which I started learning while enrolled in the "Linear Statistical Models" module in the Autumn Term, 2023-2024, and got better in it by attending laboratory sessions for the modules "Monte Carlo Simulations" and "Statistical Inference" in the Spring Term, 2023-2024.

*Corresponding author: uc38@sussex.ac.uk

Objectives

The main objectives of this paper are 1) developing a framework for practical implementation, 2) validation of mathematical models in finance through real life information, 3) improving risk assessment via mathematical means.

1. Accurate risk assessment is crucial in portfolio management. Importance Sampling allows for a more precise estimation of tail events, which are rare but have significant impacts on portfolio performance.
2. Beyond theoretical exploration, this project aims to create a practical framework for implementing Importance Sampling in real-world financial simulations. This involves developing algorithms and software tools that can be readily used by financial analysts and portfolio managers.
3. To demonstrate the efficacy of Importance Sampling, the project will include several case studies. These case studies will compare simple sampling methods with Importance Sampling in the financial medium, highlighting improvements in accuracy of projection of market values of trades and efficiency in portfolio management.

Theory

Prior And Posterior Distributions

In Bayesian statistics, a statistician treats a sample as if that sample is distributed in a very specific way, whose parameters are assumed to be random but represent the sample. Because of this, the statistician posesses the power of deciding how the parameters of the sample distribution is itself distributed.

The probability distribution chosen initially for a parameter is called the prior distribution(prior for short) for that parameter. For a parameter space Θ and a parameter $\theta \in \Theta$, a prior distribution $\pi(\theta)$ is defined with the prerequisite

$$\int_{\Theta} \pi(\theta) d\theta \in (0, \infty), \tag{1}$$

so that a proper prior π^* can be defined:

$$\pi^*(\theta) := \frac{\pi(\theta)}{\int_{\Theta} \pi(\theta) d\theta} \Rightarrow \int_{\Theta} \pi^*(\theta) d\theta = 1 \tag{2}$$

Priors for a random parameter θ are called objective only if existing data is taken into account. Otherwise, there is extra information or belief that's taken into account, resulting in a subjective distribution for θ.

Even though statistical analysis and Monte Carlo Simulations seem to work on different parts of similar questions, the mathematical background for both of them is built on Probability Theory.

Importance Sampling

Importance sampling is essentially built on the same mindset as setting a prior distribution for a parameter.

On a sampling simulation that creates values in a sample space S with a given distribution function $\pi : S \to [0, \infty)$, one can make the simulation act accordingly towards what is needed by building an importance distribution $\alpha : S \to [0, \infty)$, which modifies the original distribution π in the way one wants. The importance distribution has to satisfy the properties

$$\int_S \alpha(s)\pi(s) ds = 1; \forall s \in S, \alpha(s) > 0. \tag{3}$$

because, essentially a new prior distribution $\alpha\pi$ will be set on the random variable whose pdf or pmf is π.

There are two different types of importance distributions explained in the paper, which will be related to the type of statistics and models applied to real data. Both types manipulate the process financial positions are sampled by. In short, a uniform distribution will be assumed on the selected stocks and positions, and then the importance, or likelihood of being sampled, of the elements will be changed by what the models on data tell us.

Auto-Regressive Model

(Summarized from 1*)

A finite sequence $\{r(t) : t = 0, 1, 2, ..., T\}, T \in \mathbb{N}$ of real numbers is called "time series" when the index implies a total order between the elements related to time. One useful type of model that is built on time series is "auto-regressive model" where each element $r(t), t = 0, 1, ..., T, T \in \mathbb{N}$ is a linear combination made out of an average μ, previous entries in the sequence and a noise/error coefficient for each term:

$$\forall t \in \{1, 2, ..., T\}, r(t) = \mu + \sum_{i=1}^{t-1} \alpha_i^{(t)} r(t-i) + \varepsilon_t \tag{4}$$

The superscript on the scalars $\alpha_i^{(t)}$ and the subscript on the average μ_t show us which specific return rate the model is built on.

Assume we are trying to make an autoregressive representation for r_t for some fixed $t \in \{1, 2, ..., T\}$. If one decides to represent each term by using only k previous entries with $k < t$, then $\forall i \in \{1, 2, ..., t-k-1\}, \alpha_i^{(t)} =: 0$ and hence the elements are represented as

$$r(t) = \mu + \sum_{i=t-k}^{t-1} \alpha_i^{(t)} r(i) + \varepsilon_t = \mu + \alpha_{t-k}^{(t)} r(t-k) + \alpha_{t-k+1}^{(t)} r(t-k+1) + ... + \alpha_{t-1}^{(t)} r(t-1) + \varepsilon_t \tag{5}$$

For example, if we are representing the sixth element in the series, then

$$r(6) = \mu_6 + \alpha_1^{(6)} r(5) + \alpha_2^{(6)} r(4) + \alpha_3^{(6)} r(3) + \alpha_4^{(6)} r(2) + \alpha_1^{(5)} r(1) + \varepsilon_6 \tag{6}$$

In our studies, the data are inspected with various lengths of periods, i.e. we can choose to represent the return rates with using as many previous entries as we want since we are doing simulations for future values. One caution we have to have is that external resources might have specific structure to the models we have to follow; for example, in R, the coefficients α are set with regards to the distance of the values to the mean:

$$r_i - \mu = \alpha_1 (x_{i-1} - \mu) + ... + \alpha_j (x_{i-j} - \mu) + \varepsilon_i \tag{7}$$

Analysis

Note Throughout the analysis process, the significance level is set to be 0.99 and stocks are represented as:

- Apple: S_1
- Amazon: S_2
- Alphabet: S_3
- Meta: S_4
- Microsoft: S_5
- Nvidia: S_6

To begin the analysis, I first gathered the necessary data and applied the models explained in the theoretical framework. The data serves as the foundation of this study, so it was crucial to select stocks from companies that I have a particular interest in. For this reason, I chose a collection of stock prices over a 1-year period from prominent technology companies: Microsoft, Apple, Nvidia, Amazon, Alphabet, and Meta.

The data was sourced from the historical data tab on "finance.yahoo.com" on the 11th of May, 2024, before 5 pm. This ensured that the most recent and relevant information was included in the analysis. The focus is specifically on the "Adjusted Close" prices listed in the datasets, as these prices account for any corporate actions such as dividends, stock splits, or new stock issuance, providing a more accurate reflection of the stocks' true value over time.

By using these adjusted close prices, the analysis aims to mitigate the effects of such actions and offer a clearer picture of the underlying stock performance. This selection and preparation of data are critical steps in ensuring the robustness and reliability of the subsequent modeling and simulations. The following sections will detail the application of various statistical models, including autoregressive models and the use of importance sampling techniques, to analyze and interpret the financial performance and risk associated with these selected stocks.

Log-normal Ratio Model

The first model I assume for the ratios of stock prices is Log-normal distribution on the ratio of stock prices with one day distance. Below is the explanation:

For an arbitrary work-day $t \in \{1, 2, ..., 251\}$ in a year, and stock values $S(t) \in (0, \infty)$. I assume that

$$log(\frac{S(t+1)}{S(t)}) \approx \mathcal{N}(\mu, \sigma^2), \mu \in (-\infty, \infty), \sigma^2 \in (0, \infty) \tag{8}$$

where μ represents the "daily drift-rate" and σ represents the volatility of the stock. The number of days in the model is set to 251 because the data is taken with that constraint. There are 252 entries in each column of the data set, and we have to look at the ratios of these values. Hence, 251 ratios are calculated and used for fitting a model. R environment allows us to find the best estimates for the parameters of any distribution one chooses to fit over a data. The package "fitdistrplus" comes with a function "fitdist" that takes in data and distribution as chosen and presents all the parameter estimations in a matrix. For the 6 stocks, below is given a list of parameter estimations and estimate errors based on the data from finance.yahoo.com:

1. : Apple ratios $log(\frac{S_1(t+1)}{S_1(t)})$ are represented by

$$log(\frac{S_1(t+1)}{S_1(t)}) \approx \mathcal{N}(0.002185961, 0.008425355^2), \tag{9}$$

with standard errors on the parameters

$$\epsilon_\mu = 0.001538252, \epsilon_\sigma = 0.001019533. \tag{10}$$

2. : Amazon ratios $log(\frac{S_2(t+1)}{S_2(t)})$ are represented by

$$log(\frac{S_2(t+1)}{S_2(t)}) \approx \mathcal{N}(0.004222581, 0.018383984^2), \tag{11}$$

with standard errors on the parameters

$$\epsilon_\mu = 0.003356441, \epsilon_\sigma = 0.002341838. \tag{12}$$

3. : Alphabet ratios $log(\frac{S_3(t+1)}{S_3(t)})$ are represented by

$$log(\frac{S_3(t+1)}{S_3(t)}) \approx \mathcal{N}(0.0006186864, 0.0156883009^2), \tag{13}$$

with standard errors on the parameters

$$\epsilon_\mu = 0.002864279, \epsilon_\sigma = 0.001988443. \tag{14}$$

4. : Meta ratios $log(\frac{S_4(t+1)}{S_4(t)})$ are represented by

$$log(\frac{S_4(t+1)}{S_4(t)}) \approx \mathcal{N}(0.005545627, 0.015361453^2), \tag{15}$$

with standard errors on the parameters

$$\epsilon_\mu = 0.002804605, \epsilon_\sigma = 0.001945467. \tag{16}$$

5. : Microsoft ratios $log(\frac{S_5(t+1)}{S_5(t)})$ are represented by

$$log(\frac{S_5(t+1)}{S_5(t)}) \approx \mathcal{N}(0.002003263, 0.015114211^2), \tag{17}$$

with standard errors on the parameters

$$\epsilon_\mu = 0.002759465, \epsilon_\sigma = 0.001912937. \tag{18}$$

```
> print("apple lognormal ratio estimates: ");lognorm_model_apple = fitdist(apple_adj_ratio[1:30], distr = "norm"); print(lognorm
_model_apple)
[1] "apple lognormal ratio estimates: "
Fitting of the distribution ' norm ' by maximum likelihood
Parameters:
       estimate  Std. Error
mean 0.002185961 0.001538252
sd   0.008425355 0.001019533
> print("amazon lognormal ratio estimates: ");lognorm_model_amazon = fitdist(amazon_adj_ratio[1:30], distr = "norm"); print(logn
orm_model_amazon)
[1] "amazon lognormal ratio estimates: "
Fitting of the distribution ' norm ' by maximum likelihood
Parameters:
       estimate  Std. Error
mean 0.004222581 0.003356441
sd   0.018383984 0.002341838
> print("alphabet lognormal ratio estimates:");lognorm_model_alphabet = fitdist(alphabet_adj_ratio[1:30], distr = "norm"); print
(lognorm_model_alphabet)
[1] "alphabet lognormal ratio estimates:"
Fitting of the distribution ' norm ' by maximum likelihood
Parameters:
        estimate  Std. Error
mean 0.0006186864 0.002864279
sd   0.0156883009 0.001988443
> print("meta lognormal ratio estimates:");lognorm_model_meta = fitdist(meta_adj_ratio[1:30], distr = "norm"); print(lognorm_mod
el_meta)
[1] "meta lognormal ratio estimates:"
Fitting of the distribution ' norm ' by maximum likelihood
Parameters:
       estimate  Std. Error
mean 0.005545627 0.002804605
sd   0.015361453 0.001945467
> print("microsoft lognormal ratio estimates:");lognorm_model_microsoft = fitdist(microsoft_adj_ratio[1:30], distr = "norm"); pr
int(lognorm_model_microsoft)
[1] "microsoft lognormal ratio estimates:"
Fitting of the distribution ' norm ' by maximum likelihood
Parameters:
       estimate  Std. Error
mean 0.002003263 0.002759465
sd   0.015114211 0.001912937
> print("nvidia lognormal ratio estimates:");lognorm_model_nvidia = fitdist(nvidia_adj_ratio[1:30], distr = "norm"); print(logno
rm_model_nvidia)
[1] "nvidia lognormal ratio estimates:"
Fitting of the distribution ' norm ' by maximum likelihood
Parameters:
       estimate  Std. Error
mean 0.01173407 0.008447867
sd   0.04627087 0.005960993
```

Figure 1: *Returns of the "fitdist" function applied to ratios of values*

6. : Nvidia ratios $log(\frac{S_6(t+1)}{S_6(t)})$ are represented by

$$log(\frac{S_6(t+1)}{S_6(t)}) \approx \mathcal{N}(0.01173407, 0.04627087^2), \tag{19}$$

with standard errors on the parameters

$$\epsilon_\mu = 0.008447867, \epsilon_\sigma = 0.005960993. \tag{20}$$

The model summary results from R is given in figure 1.

Autoregressive Model

Similar to "fitdistrplus", the "stats" package in R comes with a function "ar" with various subtypes, which do estimations on the given data and find the best fitting autoregressive model. On default, the criterion that is used for choosing the best fitting model in these functions is the Akaike Information Criterion. If one wants to model the data over a specific length, for example, 5 days, instead of applying the Akaike Criterion to find the best fitting model, one has to call the function ar with the aic argument = FALSE and the order.max argument = 5.

Since the model is built onto the return rates, it is crucial that the analysis is done on the right data. For example, if I want to obtain a model by using the adjusted close prices of the first 30 days, the obtain will be built by using 29 return rates.

Model summaries for each stock's return rate is given in figure 2.

Value Projection

Projection via Log-normal Ratio Estimation

I used the classic rnorm function to obtain simulated log-ratios by putting in the estimated means and standard deviations listed in the analysis part as the arguments.

```
> print("apple autoregression results:");apple_ar_model
[1] "apple autoregression results:"

Call:
ar.ols(x = apple_adj_return[1:ar_days])

Coefficients:
      1       2       3       4       5       6       7       8       9      10      11      12      13      14
-2.4129 -2.3234 -1.4865 -1.1196 -1.1917 -1.4432 -1.2858 -1.0980 -0.7732  0.1147  0.5886 -0.2496 -0.6635 -0.3786

Intercept: 0.01436 (1.406e-15)

Order selected 14  sigma^2 estimated as  1.962e-31
> print("amazon autoregression results:");amazon_ar_model
[1] "amazon autoregression results:"

Call:
ar.ols(x = amazon_adj_return[1:ar_days])

Coefficients:
      1       2       3       4       5       6       7       8       9      10      11      12      13      14
 0.6939  9.4508  1.0800 -1.9795 -2.9725 -4.0451 -8.7040 -5.1412 -3.4024 -6.8343 -4.0615 -9.5078 -6.0484 -8.7859

Intercept: 0.1243 (4.248e-15)

Order selected 14  sigma^2 estimated as  2.747e-30
> print("alphabet autoregression results:");alphabet_ar_model
[1] "alphabet autoregression results:"

Call:
ar.ols(x = alphabet_adj_return[1:ar_days])

Coefficients:
      1       2       3       4       5       6       7       8       9      10      11      12      13      14
-0.7157 -2.0218 -0.9648 -1.7247 -1.6746 -1.0221 -0.7191 -0.9614  0.4953 -0.0735  0.9329  1.0173  0.2103  1.2036

Intercept: -0.02727 (4.233e-17)

Order selected 14  sigma^2 estimated as  2.666e-33
> print("meta autoregression results:");meta_ar_model
[1] "meta autoregression results:"

Call:
ar.ols(x = meta_adj_return[1:ar_days])

Coefficients:
      1       2       3       4       5       6       7       8       9      10      11      12      13      14
-1.0621  0.8999  1.2207  0.0704 -0.9521 -0.8409 -0.6036 -0.4711 -0.1169  0.7076  0.2472 -1.8125 -2.0872 -1.1270

Intercept: 0.01132 (2.74e-17)

Order selected 14  sigma^2 estimated as  1.836e-33
> print("microsoft autoregression results:");microsoft_ar_model
[1] "microsoft autoregression results:"

Call:
ar.ols(x = microsoft_adj_return[1:ar_days])

Coefficients:
      1       2       3       4       5       6       7       8       9      10      11      12      13      14
-0.3022 -1.0232 -0.8030 -0.2564  0.0848 -0.6126 -0.2090 -0.8519 -0.1743  0.0661  0.3464  1.2955  0.7384  1.0194

Intercept: -0.01329 (9.858e-18)

Order selected 14  sigma^2 estimated as  5.066e-34
> print("nvidia autoregression results:");nvidia_ar_model
[1] "nvidia autoregression results:"

Call:
ar.ols(x = nvidia_adj_return[1:ar_days])

Coefficients:
      1       2       3       4       5       6       7       8       9      10      11      12      13      14
-1.8839 -0.4405 -0.2769  0.5202  0.0399 -0.2983 -0.1016 -0.4224 -0.1833  0.2917  0.0723  0.3857  0.4690  0.1725

Intercept: -0.02906 (3.676e-16)

Order selected 14  sigma^2 estimated as  1.174e-31
```

Figure 2: *Returns of the Auto-Regressive model fitting on returns of stocks*

```
> lognorm_importance_long_matrix
          [,1]       [,2]      [,3]       [,4]        [,5]
[1,] 0.0000000 0.06202740 0.1527139 0.06654542 0.002960665
[2,] 0.0000000 0.03940404 0.0000000 0.34957190 0.000000000
[3,] 0.0000000 0.00000000 0.0000000 0.19639499 0.000000000
[4,] 0.3736304 0.18226294 0.0000000 0.00000000 0.080772038
[5,] 0.0000000 0.12232641 0.1584419 0.00000000 0.189415916
[6,] 0.6263696 0.59397920 0.6888443 0.38748768 0.726851381
> lognorm_importance_short_matrix
           [,1] [,2]       [,3]      [,4]      [,5]
[1,] 0.04763625    0 0.00000000 0.0000000 0.0000000
[2,] 0.36736684    0 0.72700921 0.0000000 0.7592965
[3,] 0.20501811    1 0.26111341 0.0000000 0.2407035
[4,] 0.00000000    0 0.01187739 0.1918421 0.0000000
[5,] 0.37997880    0 0.00000000 0.8081579 0.0000000
[6,] 0.00000000    0 0.00000000 0.0000000 0.0000000
```

Figure 3: *Importance distributions based on log-normal estimations*

My inner statistician wanted to use the average of the returns for each company. Yet, when I looked at the results, I realized: the log-normal parameters were estimated to be very small; since the companies I have chosen are not "volatile", the average of the means were too close to each other to create a useful importance distribution. Instead, I decided to use a smaller number of simulations and use these projections seperately for creating different importance distributions and hence different portfolios.

The values from the Analysis section will be used directly in sampling functions. The rnorm function embedded in R suits perfectly to our needs. I simply put in the mean and standard deviation estimates as the arguments and sampled one element with rnorm. Then collected all the outcomes in one matrix called "lognorm_estimates" and seperated it into its positive and negative values with two new matrices called "lognorm_importance_long_matrix" and "lognorm_importance_short_matrix" for future use.

The "long" and "short" importance distributions can be seen in figure 3.

Projection via Autoregressive Returns

I did the analysis and model-fitting on the first 30 days of data. Then I stored the values I obtained from the model-fitting in a new 6-by-30 matrix, each row representing the companies and each column representing the new estimations. The new estimations are done by assuming that the simulated values for each new day are real values. Then over the new values, a new model fitting is done. There is a bit of fussy work here, because after creating the simulated returns matrix, I foresaw that I might need to combine estimated values with the existing data. Below is the explanation for the data management in doing repeated model fitting (explained over apple):

1. The data was taken from the working directory and was stored in the vector "apple_data". The return rates $r_1(1), r_1(2), ..., r_1(30)$ are stored in the vector "apple_adj_return".
2. The projections for return rates are stored in the matrix "ar_estimated_ratios" (projections for apple are in the first row) and are created by the process in the next item.
3. The estimation for day thirty-one, $\hat{r_1}(31)$ is stored in the first column of "ar_estimated_ratios" and is put in "by hand". Other columns are filled in with a "for loop" of desired length n which is >1 without loss of generality.
4. at each iteration $i = 1, 2, ..., n$, a new autoregression model is fitted over the first 30 days and the first $i-1$ entries of the first row of "ar_estimated_ratios". Then a mean μ and some coefficients $\alpha_1^{(30+i)}(1), \alpha_1^{(30+i)}(2), ..., \alpha_1^{(30+i)}(k)$ are obtained from the data frame returned by the autoregression process.
5. If k(= length of the coefficient vector of the model, stored in "apple_ar_length") $= 0$, then the next return rate for apple is estimated to be μ which is the estimated x.intercept in .
6. If $k < i - 1$, then the next return rate for apple is estimated strictly from the projections

$$r_1(30+i) = \mu_1^{(k)} + \alpha_1^{(30+i)}(1)\hat{r}_1(i-1) + \alpha_1^{(30+i)}(2)\hat{r}_1(i-2) + ... + \alpha_1^{(30+i)}(k)\hat{r}_1(i-k+1) \tag{21}$$

7. If $k > i - 1$, then the next return rate for apple is estimated to be

$$r_1(30+i) = \mu_1^{(k)} + \alpha_1^{(30+i)}(1)\hat{r}_1(i-1) + \alpha_1^{(30+i)}(2)\hat{r}_1(i-2) + ... + \alpha_1^{(30+i)}(i-1)\hat{r}_1(1) \tag{22}$$

```
             [,1]         [,2]         [,3]         [,4]         [,5]          [,6]          [,7]          [,8]
[1,]  0.095535505 -0.1828941367  0.2154088397 -0.502049353  2.151614e+00 -1.338165e+01 -3.942766e+02 -1.796846e+05
[2,]  0.005563015 -0.0008033848 -0.0007630929 -0.049928681 -5.206973e-19 -4.049862e-19 -2.537957e-19 -1.933947e-19
[3,] -0.144611287  0.0121206339  0.0448158101 -0.282736017  2.764540e-01  3.502515e-01 -9.821381e-01  1.202282e+00
[4,]  0.031502354 -0.0496721803  0.0088057021 -0.090848760 -6.846327e-02 -9.183743e-02 -1.058476e-02 -2.060143e-02
[5,] -0.047901292 -0.0782514826 -0.1898288105 -0.254598212 -3.433761e-01 -4.663134e-01 -6.383286e-01 -8.816122e-01
[6,]  0.002310066 -0.0255624923  0.0432765332 -0.001292071 -1.194261e-02 -2.257651e-02 -1.487745e-02 -1.437202e-02
```

Figure 4: *Autoregression Model Estimates for days from 31 to 38*

$$+\alpha_1^{(30+i)}(i)r_1(30) + \alpha_1^{(30+i)}(i+1)r_1(29) + ... + \alpha_1^{(30+i)}(k-1)r_1(30+(i-1)-(k-2)) \tag{23}$$

$$+\alpha_1^{(30+i)}(k))r_1(30+(i-1)-(k-1)) \tag{24}$$

Hence, after creating this matrix we can get each column and treat it as an importance distribution while sampling portfolios.

Since the estimations above are done on top of estimations, it is wise to keep the time horizon small; the effect of the real data diminishes as number of iterations increase, so we are more prone to having errors.

Remark If I am trying to estimate a day $30+n$ value for an apple stock with this model, then I have to multiply day 30 value $S_1(30)$ with

$$\Pi_{i=1}^{n} exp(\hat{r_1}(i)) = exp(\sum_{i=1}^{n} \hat{r_1}(i)). \tag{25}$$

Portfolio Creation

When I'm creating a portfolio, my purpose is to maximize the returns I obtain by picking the right assets. For this purpose, I will be comparing the results I get with the results calculated over a portfolio created by simple sampling. For simplicity, each portfolio will have 100 assets, partitioned between long and short positions.

To create a portfolio with simple sampling, I simply sample one element from the vector $(1,2,3,4,5,6)$ with replacement and over uniform probability for 100 times in total. I could also sample a hundred numbers at once, but then an extra step of counting how many of each asset have appeared in the vector returned by the sample function I just ran would be needed.

While doing the comparisons between the "simply-sampled" portfolios with the portfolios sampled by other means, I compare the average gains. If I can't get any positive returns for some reasons, I can still compare the outcomes of different portfolios to choose the best method, which let us to the least loss in the actions.

Simple Sampling

I created a vector of length 12, first 6 representing long positions on each company, and last 6 representing shorts. Then I sampled one natural number between 1 and 12 a hundred times. To use the "sample" function, one has to input the sample space into the argument "x" and number of samples into the "size" argument. If an element should be drawn more than once, then the "replacement" argument must be $TRUE$. The probability argument in the same function attains $p = \frac{1}{n}$ where $n = length(x)$, x is the previously mentioned sample space. Figure 4 shows some random portfolios created by this process:

If needed, the matrix containing all the simple-sample portfolios can be called by the command "random_portfolios" after the R script is run.

Log-Normal Importance

I had already found various examples of estimations via Log-Normal modelling and stored them in the vectors "lognorm_importance_long_matrix" and "lognorm_importance_short_matrix", whose rows contain positive (non-inclusive)or negative estimates for one company, ordered as in the start of the Analysis section. Now, to create proper vectors to be used as probability distributions while sampling portfolios, all I have to do is divide each column to its sum of values.

After this, similar to the simple-sampling case, I used the "sample" function with the importance distributions placed above. In the script, I sampled five different portfolios to later use.

The portfolios created via these means can be seen in figure 6.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
1	8	6	13	10	3	8	8	7	5	7	12	13
2	8	10	6	10	5	8	14	9	10	6	7	7
3	15	6	13	13	9	2	4	6	10	8	6	8
4	7	12	9	10	6	5	12	7	10	12	3	7
5	6	6	9	10	8	9	12	10	5	5	7	13
6	4	9	9	7	9	6	9	6	13	9	9	10
7	8	7	5	11	10	5	10	8	6	10	13	7
8	15	9	7	6	8	9	5	8	9	6	9	9
9	9	6	13	7	7	9	2	13	6	10	10	8
10	13	8	5	6	6	9	8	9	8	13	7	8

Figure 5: *Some portfolios sampled with the default probability distribution in the "sample" function*

```
> lognorm_portfolios_day31 #necessary
      [,1] [,2] [,3] [,4] [,5]
 [1,]   50    4    6    1    9
 [2,]    0   15    3    0    0
 [3,]    0    0    0    4    0
 [4,]    0    8   41    0    0
 [5,]    0    2    0    0    0
 [6,]    0   21    0   45   41
 [7,]    0    0    0    0    0
 [8,]   19    0    0   11   20
 [9,]   11   50   40    0    7
[10,]   17    0    0    7   21
[11,]    3    0    0   32    2
[12,]    0    0   10    0    0
```

Figure 6: *Portfolios sampled by distributions in Figure 3*

Autoregressive Importance

Let's continue with an example built over the last equation mentioned in the Projection via Autoregressive Returns Estimation subsection. Say we are creating an importance distribution on stocks with a 3-day projection. Then the importance value $I_j : j = 1, 2, ..., 12$ for each long and short stock (defined similarly to the simple sampling case) will be obtained by the process explained below:

1. Obtain the cumulative sum of the first 3 columns of the autoregression estimates in vector "ar_estimated_returns". These are the values used to obtain the day 33 simulations for company stock values
2. Seperate positive and negative cumulative sums; the positive values will be used to for long positions and negative values will be used for short positions.
3. Add 1 to each value to obtain the ratios between day 33 simulated values and day 30 data
4. Divide the vectors containing the positive and negative values seperately to the sum of their elements so that this final product is usable as a pmf while sampling portfolios.

The importance vectors I obtained can be seen below:

```
> ar_importance_long; ar_importance_short
[1] 0.0000000 0.2826518 0.0000000 0.0000000 0.0000000 0.7173482
[1] 0.61113095 0.00000000 0.03235071 0.08948074 0.26703760 0.00000000
```

The portfolios I obtained are in figure 7.

Comparison of Portfolio Profits and Losses

A hundred portfolios with "simple" sampling of assets gave us an average of $333,1$ pounds of loss on the long positions and $333,6$ pounds of profit on the short positions. Below is the summary of the outcomes.

Five portfolios with "log-normal" sampling of assets gave us an average of $470,38$ pounds of loss on the long positions and $275,9$ pounds of profit on the short positions. Below is the summary of the outcomes:

Portfolios with "autoregressive" sampling of long positions gave us an average of $98,81$ pounds of loss on the long positions and 367 pounds of profit on the short positions. Below is the summary of the outcomes:

	V1	V2	V3	V4	V5
1	2	4	2	1	1
2	48	46	48	49	49
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	35	32	37	33	21
10	1	1	0	0	0
11	4	2	6	6	6
12	10	15	7	11	23

Figure 7: *Portfolios obtained by autoregressive importance distributions*

```
> print("Summary of the profits:");summary(random_long_profits[which(random_long_profits>0)])
[1] "Summary of the profits:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.

> #Summary of the losses on the long positions
> print("Summary of the losses:");summary(random_long_profits[which(random_long_profits<0)])
[1] "Summary of the losses:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -523.2  -356.4  -326.6  -328.3  -291.2  -229.7
> #Summary of the profits on the short positions
> print("Summary of the profits:");summary(random_short_profits[which(random_short_profits>0)])
[1] "Summary of the profits:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  207.7   282.9   340.7   328.9   368.6   433.5
> #Summary of the losses on the short positions
> print("Summary of the losses:");summary(random_short_profits[which(random_short_profits<0)])
[1] "Summary of the losses:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

Figure 8: *Statistics of the returns from simply-sampled portfolios*

Note that there are empty lines the summaries. That is because the returns are obtained from filtered vectors; when there is no positive/negative value in a vector, i. e. no profits/losses were made from attaining that type of positions, then there will be no statistics that come out of that vector.

Conclusion

Long Positions

Among the methods evaluated, the autoregressive model demonstrated the most favorable outcomes. Specifically, the autoregressive model minimized the average loss on long positions, achieving a reduction to less than one-fourth of the average loss observed with the log-normal weighting. Furthermore, it reduced the loss to less than one-third of the loss from simulations that did not employ importance sampling. This significant disparity shows the efficiency of the autoregressive model in managing long positions, marking it as the winner in this context, by comparisons. The ability of the autoregressive model to localize the analysis likely contributes to its performance.

Short Positions

Short positions exhibited a similar pattern, with the autoregressive model emerging as the most effective, again. Notably, the margin of superiority over the other methods was narrower compared to that seen in long positions. Despite this smaller margin, the autoregressive model still managed to outperform the simple-sample portfolios.

Conversely, the log-normal model produced the least favorable results for short positions. This underperformance may stem from the time constraints on the data, such as capturing the effects of an entire trading day in each data point, and the limited number of data points used to build the model. The reliance of the Black-Scholes equation on the log-normal ratio model, and its success in many financial contexts, suggests that the model's poor performance here could be attributed to the specific characteristics of the dataset and the inherent assumptions on the model. The Black-Scholes equation, widely recognized for its effectiveness in pricing options, fundamentally assumes a log-normal distribution of asset ratios. However, real-world data

```
> #Summary of the profits from long positions
> print("Summary of the profits:");summary(lognorm_long_profits[which(lognorm_long_profits>0)])
[1] "Summary of the profits:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.

> #Summary of the losses from long positions
> print("Summary of the losses:");summary(lognorm_long_profits[which(lognorm_long_profits<0)])
[1] "Summary of the losses:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-726.62 -659.05 -461.46 -470.38 -434.63  -70.13
>
> #Summary of the profits from short positions
> print("Summary of the profits:");summary(lognorm_short_profits[which(lognorm_short_profits>0)])
[1] "Summary of the profits:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  196.5   274.6   295.5   275.9   298.0   314.9
> #Summary of the losses from short positions
> print("Summary of the losses:");summary(lognorm_short_profits[which(lognorm_short_profits<0)])
[1] "Summary of the losses:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

Figure 9: *Statistics of the returns from log-norm-sampled portfolios*

```
> #Summary of the profits from long positions
> print("Summary of the profits:");summary(ar_long_profits[which(ar_long_profits>0)])
[1] "Summary of the profits:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.

> #Summary of the losses from long positions
> print("Summary of the losses:");summary(ar_long_profits[which(ar_long_profits<0)])
[1] "Summary of the losses:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -99.40  -99.40  -98.81  -98.81  -98.81  -97.61
>
> #Summary of the profits from short positions
> print("Summary of the profits:");summary(ar_short_profits[which(ar_short_profits>0)])
[1] "Summary of the profits:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  294.1   331.0   341.4   367.0   385.3   483.5
> #Summary of the losses from short positions
> print("Summary of the losses:");summary(ar_short_profits[which(ar_short_profits<0)])
[1] "Summary of the losses:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

Figure 10: *Statistics of the returns from auto-regressive-sampled portfolios*

always come with the error, which we assume to behave around zero without much accuracy, i.e. the variance of the error is unknown or supposed in general.

Importance of Deep Statistical Analysis

The examples provided offer valuable insights into the application of simulations in financial scenarios, yet they also highlight the necessity for thorough statistical analysis. A deeper investigation into both the data and the simulation outputs is strongly recommended to refine these models further and to validate their robustness. Such analysis should involve examining the sensitivity of the models to different market conditions, and the potential impact of outliers or structural breaks in the data.

Additionally, exploring alternative models and methods could provide further enhancements. For instance, incorporating more sophisticated techniques such as machine learning algorithms or hybrid models that combine multiple statistical approaches might yield better predictive accuracy and risk management capabilities. It is also essential to consider the practical implications of these findings for portfolio management, including how these models can be integrated into trading strategies and risk assessment frameworks.

Final Notes

The R script that contains all the steps of each calculation, model fitting, simulations, etc, is given in the appendix. For future use, the script should be converted into a more "functional" progression, meaning that all the actions should be remade by defining functions and types of arguments should be generalized for different statistics.

1. The data analysis process can be significantly improved by implementing a comprehensive function capable of handling multiple file formats such as CSV or XLSX. This function should be designed to parse

all relevant information from a given list of files, ensuring that no potentially useful data is overlooked. The function would iterate through each file, extract the necessary data, and compile everything into a list of data frames. Additionally, this function should generate plots and summaries for each data frame, providing a visual and statistical overview of the data. This approach would streamline the initial data processing phase, making it more efficient and comprehensive, and set a solid foundation for subsequent analyses.

2. Model fitting can be united under a single function as well, simply a for function is enough in this case; extra modifications to the script itself for readable code is optional. These modifications would aid in maintaining and understanding the script, especially for collaborative projects or future revisions.
3. The current analysis of the simulated data is limited, focusing solely on basic statistics. To gain deeper insights, the analysis should be expanded to include comprehensive plotting and advanced statistical metrics. Generating plots, such as density and box plots, would provide a visual representation of the simulation results, highlighting the distribution and variability of the simulated outcomes.
4. The number of simulations currently chosen is arbitrary and lacks a systematic approach. To improve this aspect, simulations should be conducted with varying parameters, such as different sample sizes, time periods, and model assumptions. Comparative analyses of these varied simulations are crucial, as they can reveal how different conditions and parameters impact the results. Comparing the outcomes of portfolios simulated under different market conditions could provide valuable insights into their robustness and reliability.
5. Some inconsistencies are present in between subsections; writing could take an overhaul in general for a denser explanation. This revision would focus on creating a cohesive narrative. Additionally, a dedicated section explaining the script in detail would be beneficial. This section could include a step-by-step walkthrough of the code, explaining the purpose and function of each part. Such documentation would enhance the script's usability, making it easier for others to understand and replicate the analysis.

Appendix

References

1. Tsay, R. S. (2010). Analysis of financial time series (3rd ed.). Wiley.

R Simulation Code

```
set.seed(26)


#### Data Analysis and Model Fitting ####
library("fitdistrplus")

#Data from WD
#Stock data can be obtained from sources such as finance.yahoo.com. Make sure
    the names of the
#files fit the code below.
apple_data = read.csv("AAPL.csv")
amazon_data = read.csv("AMZN.csv")
alphabet_data = read.csv("GOOG.csv")
meta_data = read.csv("META.csv")
microsoft_data = read.csv("MSFT.csv")
nvidia_data = read.csv("NVDA.csv")

#Adjusted Close prices
apple_adj = apple_data$Adj.Close
apple_adj_return = (apple_adj[2:length(apple_adj)] - apple_adj[1:(length(apple_
    adj)-1)])/ apple_adj[1:(length(apple_adj)-1)]
apple_adj_ratio = log(apple_adj[2:length(apple_adj)]/ apple_adj[1:(length(apple_
    adj)-1)])

amazon_adj = amazon_data$Adj.Close
amazon_adj_return = (amazon_adj[2:length(apple_adj)] - amazon_adj[1:(length(
    apple_adj)-1)])/ amazon_adj[1:(length(apple_adj)-1)]
amazon_adj_ratio = log(amazon_adj[2:length(apple_adj)]/ amazon_adj[1:(length(
    apple_adj)-1)])

alphabet_adj = alphabet_data$Adj.Close
alphabet_adj_return = (alphabet_adj[2:length(apple_adj)] - alphabet_adj[1:(
    length(apple_adj)-1)])/ alphabet_adj[1:(length(apple_adj)-1)]
alphabet_adj_ratio = log(alphabet_adj[2:length(apple_adj)]/ alphabet_adj[1:(
    length(apple_adj)-1)])

meta_adj = meta_data$Adj.Close
meta_adj_return = (meta_adj[2:length(apple_adj)] - meta_adj[1:(length(apple_adj)
    -1)])/ meta_adj[1:(length(apple_adj)-1)]
meta_adj_ratio = log(meta_adj[2:length(apple_adj)]/ meta_adj[1:(length(apple_adj
    )-1)])

microsoft_adj = microsoft_data$Adj.Close
microsoft_adj_return = (microsoft_adj[2:length(apple_adj)] - microsoft_adj[1:(
    length(apple_adj)-1)])/ microsoft_adj[1:(length(apple_adj)-1)]
microsoft_adj_ratio = log(microsoft_adj[2:length(apple_adj)]/ microsoft_adj[1:(
    length(apple_adj)-1)])
```

```
nvidia_adj = nvidia_data$Adj.Close
nvidia_adj_return = (nvidia_adj[2:length(apple_adj)] - nvidia_adj[1:(length(
    apple_adj)-1)])/ nvidia_adj[1:(length(apple_adj)-1)]
nvidia_adj_ratio = log(nvidia_adj[2:length(apple_adj)]/ nvidia_adj[1:(length(
    apple_adj)-1)])

#### Simple Sampled Portfolios ####
#### Simple Sampled Portfolios - Portfolio Creation ####
buy_assets_random = function(number=100){

  assets_random = rep(0, 12)

  for(i in 1:number){
    a = sample(1:12, size=1)
    assets_random[a] = assets_random[a] + 1
  }
  return(assets_random)
}

random_portfolios = matrix(rep(0, 12*100), byrow=TRUE, nrow=100)

for(i in 1:100) {
  a = buy_assets_random()
  random_portfolios[i,] = a
}; print(random_portfolios)

#### Simple Sampled Portfolios - Results ####

#random_portfolios is necessary, rows are portfolios and columns are number of
    long and short positions for
#each company, columns 1:6 are long positions, 7:12 are short positions
random_portfolios
#Day n Results
n=31
#From each portfolio, I get a payoff value on the estimated day and an
    investment amount on day 30

#Long positions
random_portfolio_long_payoffs = rep(0, 100)
random_portfolio_long_investments = rep(0,100)

#Payoff on the estimated day n
for (i in 1:length(random_portfolios[,1])){
  u = 0
  u = u + random_portfolios[i,1]*apple_adj[n]
  u = u + random_portfolios[i,2]*amazon_adj[n]
  u = u + random_portfolios[i,3]*alphabet_adj[n]
  u = u + random_portfolios[i,4]*meta_adj[n]
  u = u + random_portfolios[i,5]*microsoft_adj[n]
  u = u + random_portfolios[i,6]*nvidia_adj[n]
  random_portfolio_long_payoffs[i] = u
}

#Investment on day 30
```

```
for (i in 1:length(random_portfolios[,1])){
  u = 0
  u = u + random_portfolios[i,1]*apple_adj[30]
  u = u + random_portfolios[i,2]*amazon_adj[30]
  u = u + random_portfolios[i,3]*alphabet_adj[30]
  u = u + random_portfolios[i,4]*meta_adj[30]
  u = u + random_portfolios[i,5]*microsoft_adj[30]
  u = u + random_portfolios[i,6]*nvidia_adj[30]
  random_portfolio_long_investments[i] = u
}

random_long_profits = random_portfolio_long_payoffs - random_portfolio_long_
    investments
print("Simple-Sample Long Position Analysis\n")
cat("positive payoffs from the long positions on day 31:",random_long_profits[
    which(random_long_profits>0)])
cat("negative payoffs from the long positions on day 31:",random_long_profits[
    which(random_long_profits<0)])

cat("smallest negative payoff from simple-sample longs:", random_long_profits[
    which.max(random_long_profits)],
     "on index: ",which.max(random_long_profits))
cat("long portfolio with the smallest loss:",random_portfolios[which.max(random_
    long_profits), 1:6])


#Short portfolio profits
random_portfolio_short_payoffs = rep(0, 100)
random_portfolio_short_investments = rep(0,100)


#Payoff on the estimated day n
for (i in 1:length(random_portfolios[,1])){
  u = 0
  u = u + random_portfolios[i,7]*apple_adj[n]
  u = u + random_portfolios[i,8]*amazon_adj[n]
  u = u + random_portfolios[i,9]*alphabet_adj[n]
  u = u + random_portfolios[i,10]*meta_adj[n]
  u = u + random_portfolios[i,11]*microsoft_adj[n]
  u = u + random_portfolios[i,12]*nvidia_adj[n]
  random_portfolio_short_payoffs[i] = u
}

#Investment on day 30
for (i in 1:length(random_portfolios[,1])){
  u = 0
  u = u + random_portfolios[i,7]*apple_adj[30]
  u = u + random_portfolios[i,8]*amazon_adj[30]
  u = u + random_portfolios[i,9]*alphabet_adj[30]
  u = u + random_portfolios[i,10]*meta_adj[30]
  u = u + random_portfolios[i,11]*microsoft_adj[30]
  u = u + random_portfolios[i,12]*nvidia_adj[30]
  random_portfolio_short_investments[i] = u
}
```

```
#Important caution: since short stocks are borrowed and sold, we gain money on
    day 30
#and give a stock back at the estimation day. Hence, the profit is calculated by

random_short_profits = random_portfolio_short_investments - random_portfolio_
    short_payoffs
print("Simple-Sample Short Position Analysis\n")
cat("positive payoffs from the short positions on day 31:",random_short_profits[
    which(random_short_profits>0)])
cat("negative payoffs from the short positions on day 31:",random_short_profits[
    which(random_short_profits<0)])

cat("smallest negative payoff:", random_short_profits[which.max(random_short_
    profits)],
     "on index: ",which.max(random_long_profits))
cat("portfolio with the smallest loss:",random_portfolios[which.max(random_short
    _profits), 1:6])


#Summary of the profits on the long positions
print("Summary of the profits:");summary(random_long_profits[which(random_long_
    profits>0)])
#Summary of the losses on the long positions
print("Summary of the losses:");summary(random_long_profits[which(random_long_
    profits<0)])
#Summary of the profits on the short positions
print("Summary of the profits:");summary(random_short_profits[which(random_short
    _profits>0)])
#Summary of the losses on the short positions
print("Summary of the losses:");summary(random_short_profits[which(random_short_
    profits<0)])

mean(random_long_profits)
mean(random_short_profits)


#### Log-normal Ratio Model ####
install.packages("fitdistrplus")
library(fitdistrplus)

#### Log-normal Ratio Model - Model Fitting ####

print("apple lognormal ratio estimates: ");lognorm_model_apple = fitdist(apple_
    adj_ratio[1:30], distr = "norm")
print(lognorm_model_apple)
print("amazon lognormal ratio estimates: ");lognorm_model_amazon = fitdist(
    amazon_adj_ratio[1:30], distr = "norm")
print(lognorm_model_amazon)
print("alphabet lognormal ratio estimates:");lognorm_model_alphabet = fitdist(
    alphabet_adj_ratio[1:30], distr = "norm")
print(lognorm_model_alphabet)
print("meta lognormal ratio estimates:");lognorm_model_meta = fitdist(meta_adj_
    ratio[1:30], distr = "norm")
print(lognorm_model_meta)
print("microsoft lognormal ratio estimates:");lognorm_model_microsoft = fitdist(
```

```
    microsoft_adj_ratio[1:30], distr = "norm")
print(lognorm_model_microsoft)
print("nvidia lognormal ratio estimates:");lognorm_model_nvidia = fitdist(nvidia
    _adj_ratio[1:30], distr = "norm")
print(lognorm_model_nvidia)

#### Log-normal Ratio Model - Day 31 Log-Normal Estimations ####


lognorm_estimate_apple = rnorm(n=1, mean=lognorm_model_apple$estimate[1], sd=
    lognorm_model_apple$estimate[2])
lognorm_estimate_amazon = rnorm(n=1, mean=lognorm_model_amazon$estimate[1], sd=
    lognorm_model_amazon$estimate[2])
lognorm_estimate_alphabet = rnorm(n=1, mean=lognorm_model_alphabet$estimate[1],
    sd=lognorm_model_alphabet$estimate[2])
lognorm_estimate_meta = rnorm(n=1, mean=lognorm_model_meta$estimate[1], sd=
    lognorm_model_meta$estimate[2])
lognorm_estimate_microsoft = rnorm(n=1, mean=lognorm_model_microsoft$estimate
    [1], sd=lognorm_model_microsoft$estimate[2])
lognorm_estimate_nvidia = rnorm(n=1, mean=lognorm_model_nvidia$estimate[1], sd=
    lognorm_model_nvidia$estimate[2])

cat("apple lognorm estimated mean:", lognorm_model_apple$estimate[1])
cat("amazon lognorm estimated mean:", lognorm_model_amazon$estimate[1])
cat("alphabet lognorm estimated mean:", lognorm_model_alphabet$estimate[1])
cat("meta lognorm estimated mean:", lognorm_model_meta$estimate[1])
cat("microsoft lognorm estimated mean:", lognorm_model_microsoft$estimate[1])
cat("nvidia lognorm estimated mean:", lognorm_model_nvidia$estimate[1])

###Single estimations
lognorm_apple_value_31 = apple_data$Adj.Close[30]*exp(lognorm_estimate_apple)
cat("day 31 lognorm estimate for apple:",lognorm_apple_value_31,"\n")
cat("the actual day 31 value of apple:", apple_data$Adj.Close[31],"\n")
cat("and the day 30 value of apple:", apple_data$Adj.Close[30],"\n")

lognorm_amazon_value_31 = amazon_data$Adj.Close[30]*exp(lognorm_estimate_amazon)
cat("day 31 lognorm estimate for amazon:",lognorm_amazon_value_31,"\n")
cat("the actual day 31 value of amazon:", amazon_data$Adj.Close[31],"\n")
cat("and the day 30 value of amazon:", amazon_data$Adj.Close[30],"\n")

lognorm_alphabet_value_31 = alphabet_data$Adj.Close[30]*exp(lognorm_estimate_
    alphabet)
cat("day 31 lognorm estimate for alphabet:",lognorm_alphabet_value_31,"\n")
cat("the actual day 31 value of alphabet:", alphabet_data$Adj.Close[31],"\n")
cat("and the day 30 value of alphabet:", alphabet_data$Adj.Close[30],"\n")

lognorm_meta_value_31 = meta_data$Adj.Close[30]*exp(lognorm_estimate_meta)
cat("day 31 lognorm estimate for meta:",lognorm_meta_value_31,"\n")
cat("the actual day 31 value of meta:", meta_data$Adj.Close[31],"\n")
cat("and the day 30 value of meta:", meta_data$Adj.Close[30],"\n")

lognorm_nvidia_value_31 = nvidia_data$Adj.Close[30]*exp(lognorm_estimate_nvidia)
cat("day 31 lognorm estimate for nvidia:",lognorm_nvidia_value_31,"\n")
cat("and the day 30 value of nvidia:", nvidia_data$Adj.Close[30],"\n")
```

```
###Multiple estimations
lognorm_estimate_amount = 5
lognorm_estimates = matrix(rep(0, 6*lognorm_estimate_amount), byrow = TRUE, nrow
    = 6)
A = lognorm_estimates
for (i in 1:lognorm_estimate_amount){
  A[1,i] =  rnorm(n=1, mean=lognorm_model_apple$estimate[1], sd=lognorm_model_
      apple$estimate[2])
}

for (i in 1:lognorm_estimate_amount){
  A[2,i] =  rnorm(n=1, mean=lognorm_model_amazon$estimate[1], sd=lognorm_model_
      amazon$estimate[2])
}

for (i in 1:lognorm_estimate_amount){
  A[3,i] =  rnorm(n=1, mean=lognorm_model_alphabet$estimate[1], sd=lognorm_model
      _alphabet$estimate[2])
}

for (i in 1:lognorm_estimate_amount){
  A[4,i] =  rnorm(n=1, mean=lognorm_model_meta$estimate[1], sd=lognorm_model_
      meta$estimate[2])
}

for (i in 1:lognorm_estimate_amount){
  A[5,i] =  rnorm(n=1, mean=lognorm_model_microsoft$estimate[1], sd=lognorm_
      model_microsoft$estimate[2])
}

for (i in 1:lognorm_estimate_amount){
  A[6,i] =  rnorm(n=1, mean=lognorm_model_nvidia$estimate[1], sd=lognorm_model_
      nvidia$estimate[2])
}
lognorm_estimates = A;

cat("Log-Normal Estimations for day 31:\n",lognorm_estimates)

#### Log-normal Ratio Model - Log-Normal Importance ####


#In the matrices below, rows are companies and columns are simulations
lognorm_importance_long_matrix = matrix(rep(0,6*lognorm_estimate_amount), byrow=
    TRUE, nrow=6)
lognorm_importance_short_matrix = matrix(rep(0,6*lognorm_estimate_amount), byrow
    =TRUE, nrow=6)

#lognorm_estimates is necessary for the rest to work.

for(i in 1:lognorm_estimate_amount) {
  for(j in 1:6){
    if (lognorm_estimates[j,i] > 0){
      lognorm_importance_long_matrix[j,i] = lognorm_estimates[j,i]
    } else {
      lognorm_importance_short_matrix[j,i] = lognorm_estimates[j,i]
```

```
    }
  }
}

for(i in 1:lognorm_estimate_amount){
  lognorm_importance_long_matrix[,i] = (lognorm_importance_long_matrix[,i]
                                        /sum(lognorm_importance_long_matrix[,i])
                                          )
  lognorm_importance_short_matrix[,i] = (lognorm_importance_short_matrix[,i]
                                         /sum(lognorm_importance_short_matrix[,i
                                           ]))
}
cat("Log-Normal Importance distributions for Long Positions:\n",lognorm_
    importance_long_matrix)
cat("Log-Normal Importance distributions for Short Positions:\n",lognorm_
    importance_short_matrix)


#### Log-normal Ratio Model - Log-Normal Sampling ####


#lognorm_importance_long_matrix and lognorm_importance_long_matrix are necessary
#in these vectors, rows should be companies and columns should be simulations
#Form 50 long, 50 short positions with each different log-norm importance
    distributions

#Log-Normal Portfolios for day 31
lognorm_portfolios_day31 = matrix(rep(0, 12*lognorm_estimate_amount), byrow=TRUE
    , nrow=12)
for (i in 1:lognorm_estimate_amount){
  for(j in 1:50){
    asset_long = sample(1:6, size=1, prob = lognorm_importance_long_matrix[,i])
    lognorm_portfolios_day31[asset_long,i] = lognorm_portfolios_day31[asset_long
        ,i] + 1
  }
  for(j in 1:50){
    asset_short = sample(7:12, size=1, prob = lognorm_importance_short_matrix[,i
        ])
    lognorm_portfolios_day31[asset_short,i] = lognorm_portfolios_day31[asset_
        short,i] + 1
  }
}
print(lognorm_portfolios_day31)

#### Log-normal Ratio Model - Profits and Losses ####

#Log-Normal Payoffs on day 31
lognorm_portfolios_day31 #necessary

#Long profits/losses
lognorm_long_payoffs = rep(0, length(lognorm_portfolios_day31[1,]))
lognorm_long_investment = rep(0, length(lognorm_portfolios_day31[1,]))

for(i in 1:length(lognorm_long_investment)){
  u = 0
```

```
  u = u + lognorm_portfolios_day31[1,i]*apple_adj[31]
  u = u + lognorm_portfolios_day31[2,i]*amazon_adj[31]
  u = u + lognorm_portfolios_day31[3,i]*alphabet_adj[31]
  u = u + lognorm_portfolios_day31[4,i]*meta_adj[31]
  u = u + lognorm_portfolios_day31[5,i]*microsoft_adj[31]
  u = u + lognorm_portfolios_day31[6,i]*nvidia_adj[31]
  lognorm_long_payoffs[i] = u
}
for(i in 1:length(lognorm_long_investment)){
  u = 0
  u = u + lognorm_portfolios_day31[1,i]*apple_adj[30]
  u = u + lognorm_portfolios_day31[2,i]*amazon_adj[30]
  u = u + lognorm_portfolios_day31[3,i]*alphabet_adj[30]
  u = u + lognorm_portfolios_day31[4,i]*meta_adj[30]
  u = u + lognorm_portfolios_day31[5,i]*microsoft_adj[30]
  u = u + lognorm_portfolios_day31[6,i]*nvidia_adj[30]
  lognorm_long_investment[i] = u
}
lognorm_long_profits = lognorm_long_payoffs - lognorm_long_investment
cat("positive payoffs from the lognorm long positions on day 31:",lognorm_long_
    profits[which(lognorm_long_profits>0)])
cat("negative payoffs from the lognorm long positions on day 31:",lognorm_long_
    profits[which(lognorm_long_profits<0)])

lognorm_short_payoffs = rep(0, length(lognorm_portfolios_day31[1,]))
lognorm_short_investment = rep(0, length(lognorm_portfolios_day31[1,]))

for(i in 1:length(lognorm_short_investment)){
  u = 0
  u = u + lognorm_portfolios_day31[7,i]*apple_adj[31]
  u = u + lognorm_portfolios_day31[8,i]*amazon_adj[31]
  u = u + lognorm_portfolios_day31[9,i]*alphabet_adj[31]
  u = u + lognorm_portfolios_day31[10,i]*meta_adj[31]
  u = u + lognorm_portfolios_day31[11,i]*microsoft_adj[31]
  u = u + lognorm_portfolios_day31[12,i]*nvidia_adj[31]
  lognorm_short_payoffs[i] = u
}
for(i in 1:length(lognorm_short_investment)){
  u = 0
  u = u + lognorm_portfolios_day31[7,i]*apple_adj[30]
  u = u + lognorm_portfolios_day31[8,i]*amazon_adj[30]
  u = u + lognorm_portfolios_day31[9,i]*alphabet_adj[30]
  u = u + lognorm_portfolios_day31[10,i]*meta_adj[30]
  u = u + lognorm_portfolios_day31[11,i]*microsoft_adj[30]
  u = u + lognorm_portfolios_day31[12,i]*nvidia_adj[30]
  lognorm_short_investment[i] = u
}
lognorm_short_profits = lognorm_short_investment - lognorm_short_payoffs
cat("profits from the lognorm short positions on day 31:",lognorm_short_profits[
    which(lognorm_short_profits>0)])
cat("losses  from the lognorm short positions on day 31:",lognorm_short_profits[
    which(lognorm_short_profits<0)])

#Summary of the profits from long positions
print("Summary of the profits:");summary(lognorm_long_profits[which(lognorm_long
```

```
    _profits >0)])
#Summary of the losses from long positions
print("Summary of the losses:");summary(lognorm_long_profits[which(lognorm_long_
    profits <0)])

#Summary of the profits from short positions
print("Summary of the profits:");summary(lognorm_short_profits[which(lognorm_
    short_profits >0)])
#Summary of the losses from short positions
print("Summary of the losses:");summary(lognorm_short_profits[which(lognorm_
    short_profits <0)])

mean(lognorm_long_profits)
mean(lognorm_short_profits)

#### Autoregressive Model ####
library(stats)
install.packages("stats")
ar_days = 29 #necessary, decides which days' data will be taken into the model
help(ar)

#### Autoregressive Model - Model Fitting ####

#First 29 returns are taken into the model => First 30 days are used in the
    calculations

apple_ar_model=ar.ols(apple_adj_return[1:ar_days])
cat("apple autoregression results:");apple_ar_model

amazon_ar_model=ar.ols(amazon_adj_return[1:ar_days])
cat("amazon autoregression results:");amazon_ar_model

alphabet_ar_model=ar.ols(alphabet_adj_return[1:ar_days])
print("alphabet autoregression results:");alphabet_ar_model

meta_ar_model=ar.ols(meta_adj_return[1:ar_days])
cat("meta autoregression results:");meta_ar_model

microsoft_ar_model=ar.ols(microsoft_adj_return[1:ar_days])
cat("microsoft autoregression results:");microsoft_ar_model


nvidia_ar_model=ar.ols(nvidia_adj_return[1:ar_days])
print("nvidia autoregression results:");nvidia_ar_model


#### Autoregressive Model - Estimations on Day 31 ####

apple_ar_length = length(apple_ar_model$ar)
amazon_ar_length = length(amazon_ar_model$ar)
alphabet_ar_length = length(alphabet_ar_model$ar)
meta_ar_length = length(meta_ar_model$ar)
microsoft_ar_length = length(microsoft_ar_model$ar)
nvidia_ar_length = length(nvidia_ar_model$ar)
```

```
apple_ar_estimate = (apple_ar_model$x.intercept
                     + (apple_adj_return[(ar_days - apple_ar_length + 1):ar_days
                        ]
                        -apple_ar_model$x.intercept)
                     %*%apple_ar_model$ar)
amazon_ar_estimate = (amazon_ar_model$x.intercept
                      + (amazon_adj_return[(ar_days - amazon_ar_length + 1):ar_
                         days]
                         -amazon_ar_model$x.intercept )
                      %*%amazon_ar_model$ar)
alphabet_ar_estimate = (alphabet_ar_model$x.intercept
                        + (alphabet_adj_return[(ar_days - alphabet_ar_length +
                           1):ar_days]
                           -alphabet_ar_model$x.intercept)
                        %*%alphabet_ar_model$ar)
meta_ar_estimate = (meta_ar_model$x.intercept
                    + (meta_adj_return[(ar_days - meta_ar_length + 1):ar_days]
                       -meta_ar_model$x.intercept)
                    %*%meta_ar_model$ar)
microsoft_ar_estimate = (microsoft_ar_model$x.intercept
                         + (microsoft_adj_return[(ar_days - microsoft_ar_length
                            + 1):ar_days]
                            -microsoft_ar_model$x.intercept )
                         %*%microsoft_ar_model$ar)
nvidia_ar_estimate = (nvidia_ar_model$x.intercept
                      + (nvidia_adj_return[(ar_days - nvidia_ar_length + 1):ar_
                         days]
                         -nvidia_ar_model$x.intercept )
                      %*%nvidia_ar_model$ar)
apple_ar_estimate

#### Autoregressive Model - Further Estimations  ####

#ar_days necessary to signify on how many days the model was built on.
ar_days
ar_estimated_returns = matrix(rep(x=0, times=6*(60-ar_days-1)), byrow=TRUE, nrow
    =6)
#The initial vector has to be set for the script to work.
ar_estimated_returns[, 1] = c(apple_ar_estimate, amazon_ar_estimate, alphabet_ar
    _estimate,
                              meta_ar_estimate, microsoft_ar_estimate, nvidia_ar
                                  _estimate)
ar_estimated_returns
for (i in 2:(ar_days+1)) {

  cat("Iteration Number:", i)
  apple_ar_model=ar.ols(c(apple_adj_return[1:ar_days],
                          ar_estimated_returns[1,1:(i-1)])); apple_ar_model

  amazon_ar_model=ar.ols(c(amazon_adj_return[1:ar_days],
                           ar_estimated_returns[2,1:(i-1)]))

  alphabet_ar_model=ar.ols(c(alphabet_adj_return[1:ar_days],
                             ar_estimated_returns[3,1:(i-1)]))
```

```
meta_ar_model=ar.ols(c(meta_adj_return[1:ar_days],
                       ar_estimated_returns[4,1:(i-1)]))

microsoft_ar_model=ar.ols(c(microsoft_adj_return[1:ar_days],
                            ar_estimated_returns[5,1:(i-1)]))

nvidia_ar_model=ar.ols(c(nvidia_adj_return[1:ar_days],
                         ar_estimated_returns[6,1:(i-1)]))


apple_ar_length = length(apple_ar_model$ar)

amazon_ar_length = length(amazon_ar_model$ar)

alphabet_ar_length = length(alphabet_ar_model$ar)

meta_ar_length = length(meta_ar_model$ar)

microsoft_ar_length = length(microsoft_ar_model$ar)

nvidia_ar_length = length(nvidia_ar_model$ar)

if (apple_ar_length == 0) {apple_ar_estimate = apple_ar_model$x.intercept}
else if (i > apple_ar_length) {
  new_returns_vector_apple = ar_estimated_returns[1,((i - apple_ar_length):(i
      -1))]
  apple_ar_estimate = (apple_ar_model$x.intercept + new_returns_vector_apple
                       %*%apple_ar_model$ar)

}
else {
  new_returns_vector_apple = c(apple_adj_return[(ar_days-(apple_ar_length-(i
      -1))+1):ar_days],
                               ar_estimated_returns[1,1:(i-1)])
  apple_ar_estimate = (apple_ar_model$x.intercept + new_returns_vector_apple
                       %*%apple_ar_model$ar)

}

if (amazon_ar_length == 0) {amazon_ar_estimate = amazon_ar_model$x.intercept}
else if (i > amazon_ar_length) {
  new_returns_vector_amazon = ar_estimated_returns[2,((i - amazon_ar_length):(
      i-1))]
  amazon_ar_estimate = (amazon_ar_model$x.intercept + new_returns_vector_
      amazon
                        %*%amazon_ar_model$ar)

}
else {
  new_returns_vector_amazon = c(amazon_adj_return[(ar_days-(amazon_ar_length-(
      i-1))+1):ar_days],
                                ar_estimated_returns[2,1:(i-1)])
  amazon_ar_estimate = (amazon_ar_model$x.intercept + new_returns_vector_
      amazon
                        %*%amazon_ar_model$ar)
```

```
}

if (alphabet_ar_length == 0) {alphabet_ar_estimate = alphabet_ar_model$x.
    intercept}
else if (i > alphabet_ar_length) {
  new_returns_vector_alphabet = ar_estimated_returns[3,((i - alphabet_ar_
      length):(i-1))]
  alphabet_ar_estimate = (alphabet_ar_model$x.intercept + new_returns_vector_
      alphabet
                          %*%alphabet_ar_model$ar)

}
else {
  new_returns_vector_alphabet = c(alphabet_adj_return[(ar_days-(alphabet_ar_
      length-(i-1))+1):ar_days],
                                  ar_estimated_returns[3,1:(i-1)])
  alphabet_ar_estimate = (alphabet_ar_model$x.intercept + new_returns_vector_
      alphabet
                          %*%alphabet_ar_model$ar)

}
if (meta_ar_length == 0) {meta_ar_estimate = meta_ar_model$x.intercept}
else if (i > meta_ar_length) {
  new_returns_vector_meta = ar_estimated_returns[4,((i - meta_ar_length):(i-1)
      )]
  meta_ar_estimate = (meta_ar_model$x.intercept + new_returns_vector_meta
                      %*%meta_ar_model$ar)

}
else {
  new_returns_vector_meta = c(meta_adj_return[(ar_days-(meta_ar_length-(i-1))
      +1):ar_days],
                              ar_estimated_returns[4,1:(i-1)])
  meta_ar_estimate = (meta_ar_model$x.intercept + new_returns_vector_meta
                      %*%meta_ar_model$ar)

}

if (microsoft_ar_length == 0) {microsoft_ar_estimate = microsoft_ar_model$x.
    intercept}
else if (i > microsoft_ar_length) {
  new_returns_vector_microsoft = ar_estimated_returns[5,((i - microsoft_ar_
      length):(i-1))]
  microsoft_ar_estimate = (microsoft_ar_model$x.intercept + new_returns_vector
      _microsoft
                           %*%microsoft_ar_model$ar)

}
else {
  new_returns_vector_microsoft = c(microsoft_adj_return[(ar_days-(microsoft_ar
      _length-(i-1))+1):ar_days],
                                   ar_estimated_returns[5,1:(i-1)])
  microsoft_ar_estimate = (microsoft_ar_model$x.intercept + new_returns_vector
      _microsoft
```

```
                                %*%microsoft_ar_model$ar)
  }

  if (nvidia_ar_length == 0) {nvidia_ar_estimate = nvidia_ar_model$x.intercept}
  else if (i > nvidia_ar_length) {
    new_returns_vector_nvidia = ar_estimated_returns[6,((i - nvidia_ar_length):(
        i-1))]
    nvidia_ar_estimate = (nvidia_ar_model$x.intercept + new_returns_vector_
        nvidia
                          %*%nvidia_ar_model$ar)
  }
  else {
    new_returns_vector_nvidia = c(nvidia_adj_return[(ar_days-(nvidia_ar_length-(
        i-1))+1):ar_days],
                                  ar_estimated_returns[6,1:(i-1)])
    nvidia_ar_estimate = (nvidia_ar_model$x.intercept + new_returns_vector_
        nvidia
                          %*%nvidia_ar_model$ar)
  }

  ar_estimated_returns[, i] = c(apple_ar_estimate, amazon_ar_estimate, alphabet_
      ar_estimate,
                                meta_ar_estimate, microsoft_ar_estimate, nvidia_
                                    ar_estimate)
};print(ar_estimated_returns) #column length is 6 => rows represent companies,
#columns represent estimations
print(ar_estimated_returns)

#### Autoregressive Model - Importance Distributions####
ar_estimated_day = 31


ar_importance_long = rep(0,6)
ar_importance_short = rep(0,6)

for (i in 1:6){
  z = cumsum(ar_estimated_returns[i,1:(ar_estimated_day-30)])
  if (z[(ar_estimated_day-30)] > 0){
    ar_importance_long[i] = z[(ar_estimated_day-30)]
  } else {
    ar_importance_short[i] = z[(ar_estimated_day-30)]
  }
}

ar_importance_long = ar_importance_long/sum(ar_importance_long);cat(ar_
    importance_long)
ar_importance_short = ar_importance_short/sum(ar_importance_short);cat(ar_
    importance_short)

ar_importance_long; ar_importance_short

#### Autoregressive Model - Portfolio Sampling  ####

#ar_importance_long,ar_importance_short necessary
#ar_estimated_day necessary since ar_importance_long and ar_importance_short
```

```
#was created using this value. For now, importance vectors are created with
ar_estimated_day
ar_estimate_amount = 5
#Portfolio sampling on day "ar_estimated_day"
ar_portfolio_estimated_day = matrix(rep(0, 12*ar_estimate_amount), byrow=TRUE,
    nrow=12)
ar_importance_long;ar_importance_short

for(j in 1:ar_estimate_amount)  {
  for(i in 1:50){
    asset_long = sample(1:6, size=1, prob=ar_importance_long)
    ar_portfolio_estimated_day[asset_long, j] = ar_portfolio_estimated_day[asset
        _long, j] + 1
    asset_short = sample(7:12, size=1, prob=ar_importance_short)
    ar_portfolio_estimated_day[asset_short, j] = ar_portfolio_estimated_day[
        asset_short, j] + 1
  }
}; ar_portfolio_estimated_day
cat("Portfolios sampled via AR Model:", ar_portfolio_estimated_day)

#### Autoregressive Model - Profits and Losses ####

#ar_portfolio_estimated_day necessary, holds the positions for each type of
    asset
ar_portfolio_estimated_day

#Long positions
ar_long_payoffs = rep(0, ar_estimate_amount)
ar_long_investment = rep(0, ar_estimate_amount)

#Total value of the long positions on the estimated day
for(i in 1:ar_estimate_amount) {
  u = 0
  u = u + ar_portfolio_estimated_day[1,i]*apple_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[2,i]*amazon_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[3,i]*alphabet_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[4,i]*meta_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[5,i]*microsoft_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[6,i]*nvidia_adj[ar_estimated_day]
  ar_long_payoffs[i] = u
}
#Investment on day 30
for(i in 1:ar_estimate_amount) {
  u = 0
  u = u + ar_portfolio_estimated_day[1,i]*apple_adj[30]
  u = u + ar_portfolio_estimated_day[2,i]*amazon_adj[30]
  u = u + ar_portfolio_estimated_day[3,i]*alphabet_adj[30]
  u = u + ar_portfolio_estimated_day[4,i]*meta_adj[30]
  u = u + ar_portfolio_estimated_day[5,i]*microsoft_adj[30]
  u = u + ar_portfolio_estimated_day[6,i]*nvidia_adj[30]
  ar_long_investment[i] = u
}
ar_long_profits = ar_long_payoffs - ar_long_investment
cat("profits from the ar long position on the estimated day:",ar_long_profits[
    which(ar_long_profits>0)])
```

```
cat("losss from the ar long position on the estimated day:",ar_long_profits[
    which(ar_long_profits<0)])

#Short positions
ar_short_payoffs = rep(0, ar_estimate_amount)
ar_short_investment = rep(0, ar_estimate_amount)

#Total value of the short positions on the estimated day
for(i in 1:ar_estimate_amount) {
  u = 0
  u = u + ar_portfolio_estimated_day[7,i]*apple_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[8,i]*amazon_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[9,i]*alphabet_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[10,i]*meta_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[11,i]*microsoft_adj[ar_estimated_day]
  u = u + ar_portfolio_estimated_day[12,i]*nvidia_adj[ar_estimated_day]
  ar_short_payoffs[i] = u
}
#Investment on day 30
for(i in 1:ar_estimate_amount) {
  u = 0
  u = u + ar_portfolio_estimated_day[7,i]*apple_adj[30]
  u = u + ar_portfolio_estimated_day[8,i]*amazon_adj[30]
  u = u + ar_portfolio_estimated_day[9,i]*alphabet_adj[30]
  u = u + ar_portfolio_estimated_day[10,i]*meta_adj[30]
  u = u + ar_portfolio_estimated_day[11,i]*microsoft_adj[30]
  u = u + ar_portfolio_estimated_day[12,i]*nvidia_adj[30]
  ar_short_investment[i] = u
}
ar_short_profits = ar_short_investment - ar_short_payoffs
cat("profits from the ar short positions on estimated day:",ar_short_profits[
    which(ar_short_profits>0)])
cat("losses from the ar short positions on estimated day:",ar_short_profits[
    which(ar_short_profits<0)])

#Summary of the profits from long positions
print("Summary of the profits:");summary(ar_long_profits[which(ar_long_profits
    >0)])
#Summary of the losses from long positions
print("Summary of the losses:");summary(ar_long_profits[which(ar_long_profits<0)
    ])

#Summary of the profits from short positions
print("Summary of the profits:");summary(ar_short_profits[which(ar_short_profits
    >0)])
#Summary of the losses from short positions
print("Summary of the losses:");summary(ar_short_profits[which(ar_short_profits
    <0)])

#### Linear Regression Model Fitting #### (not used in the paper)

#First 30 days
#mlr_model_apple_first30 = lm(apple_adj[1:30] ~ amazon_adj[1:30]
#                             + alphabet_adj[1:30]
#                             + meta_adj[1:30]
```

```
#                             + microsoft_adj[1:30]
#                             + nvidia_adj[1:30])
#summary(mlr_model_apple_first30)
##Refined model for apple
#mlr_model_apple_first30_new = lm(apple_adj[1:30] ~ amazon_adj[1:30]
#                                 + alphabet_adj[1:30]
#                                 + microsoft_adj[1:30])
#summary(mlr_model_apple_first30_new)
#mlr_model_apple_first30_new$coefficients

#Comparison of two models
#anova(mlr_model_apple_first30,mlr_model_apple_first30_new)
#F-value too big => No significant effect on the results by dropping Meta and
    Nvidia variables,
#yet it makes the simulation process easier, so the new model is useful.

#Initial Linear Regression
#mlr_model_amazon = lm(amazon_adj ~ apple_adj + alphabet_adj + meta_adj +
    microsoft_adj + nvidia_adj)
#summary(mlr_model_amazon)

#mlr_model_alphabet = lm(alphabet_adj ~ apple_adj + amazon_adj + meta_adj +
    microsoft_adj + nvidia_adj)
#summary(mlr_model_alphabet)

#mlr_model_meta = lm(meta_adj ~ apple_adj + amazon_adj + alphabet_adj +
    microsoft_adj + nvidia_adj)
#summary(mlr_model_meta)

#mlr_model_microsoft = lm(microsoft_adj ~ apple_adj + amazon_adj + alphabet_adj
    + meta_adj + nvidia_adj)
#summary(mlr_model_microsoft)

#mlr_model_nvidia = lm(nvidia_adj ~ apple_adj + amazon_adj + alphabet_adj + meta
    _adj + microsoft_adj)
#summary(mlr_model_nvidia)
```