UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Tõnn Talpsepp

Data Analytics for Estimating the Disposition Effect

Master's Thesis (30 ECTS)

Supervisor: Rajesh Sharma, Ph.D.

Tartu 2021

Data Analytics for Estimating the Disposition Effect

Abstract:

The aim of the thesis is to design a data warehouse and develop data wrangling and feature engineering procedures that enable to store and transform stock market transaction data to conduct estimations of the disposition. The disposition effect is a probabilistic measure describing the behaviour of investors and is estimated using transaction level stock market data set. I develop the requirements for feature engineering, create a data model based on the star schema, provide detailed structure of the physical data model, and implement it using a relational database. I develop data transformation procedures, which include data generation procedures and financial calculation algorithms written in Java. The data transformation procedures enable to generate data for the data warehouse fact table and make various calculations for dimension tables. I run simulations to validate the suitability of the developed database structure and data transformation procedures. The simulations generate over three hundred million cases of stock market investment transactions in the fact table with hypothetical stop loss orders. The analysis of the simulation results show that the proposed data models and their implementation is appropriate for the task; and the developed data generation and calculation algorithms work as expected and enable to gain important new information about the disposition effect.

Keywords:

Data analytics, data wrangling, simulations, disposition effect, stock market, stop-loss orders

CERCS: P170 Computer science, numerical analysis, systems, control; S181 Financial science

Andmeanalüütika dispositsiooniefekti hindamiseks

Lühikokkuvõte:

Käesoleva magistritöö eesmärgiks on disainida andmeait ning luua andmete muundamise ning tunnusehõive protseduurid mis aitaks salvestada ja transformeerida aktsiaturu tehinguandmed kujule hindamaks dispositsiooniefekti. Dispositsiooniefekt on tõenäosuslik mõõdik, mis kirjeldab investorite käitumist ning mida hinnatakse kasutades aktsiaturu tehinguandmestikke. Käesolevas luuakse nõuded tunnushõive teostamiseks, vajalikud andmemudelid, antakse ülevaade detailsest füüsilisest andmestruktuurist, mis on rakendatud kasutades relatsioonilist andmebaasimootorit. Töös arendatakse andmete transformeerimise protseduurid, mis hõlmavad andmete genereerimist ning finantsarvutuste algoritme. Vastavad protseduurid on kirjutatud Java programmeerimiskeeles. Andmete transformeerimise protseduurid võimaldavad genereerida andmeid loodud andmeaida faktitabeli täitmise jaoks ning teha vajalikke arvutusi dimensioonitabelite andmete leidmiseks. Töös viiakse läbi simulatsioonid, mille käigus genereeritakse üle 300 miljoni andmerea aktsiatehingute kohta faktitabelisse, kasutades hüpoteetilisi *stop-loss* tehingukorraldusi. Simulatsioonitulemuste analüüs näitab, et töös väljapakutud andmemudelid ja nende realisatsioon on seatud ülesande jaoks sobiv; väljatöötatud andmete genereerimise protseduurid ning arvutused töötavad vastavalt ootustele ning võimaldavad saada uut väärtuslikku informatsiooni dispositsiooniefekti kohta.

Võtmesõnad:

Andmeanalüütika, simulatsioonid, dispositsiooniefekt, aktsiaturg, stop-loss tehingukorraldused

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria); S181 Rahandus

Table of Contents

1 Introduction

The volume of financial data is growing year by year and it is becoming more readily available for all type of investors. However, not all investors are able to make rational investment decisions but are affected by behavioural biases and tend to make systematic mistakes when investing. The disposition effect is one of most widely known behavioural mistakes that stock market investors tend to make. The disposition effect is the tendency of investors to sell their winning positions too early and hold on to their losing positions too long. The effect was first documented by [1].

Improving the estimation of the disposition effect can be considered the business problem that the current thesis tries to solve. Various data modelling, automated data transformation, financial calculation, statistical analysis, and overall data analytics tasks need to be carried out to gain new insights how to solve this business problem and analyse the effects of the business problem that has not been possible so far.

Identifying the disposition effect from stock market transaction data requires large volumes of transactions. It is possible to calculate the disposition effect at individual level [2] but generally such calculations do not yield accurate results because one generally needs to calculate the probabilities of selling positions dependent on whether the position is in a loss or in a gain. Individual investors make rarely that many stock market transactions (leaving aside professional day-traders) that such calculations would give meaningful results. Thus, in most cases estimating the disposition effect requires analysing large volumes of transactions and making conclusions about certain groups of investors i.e. pooling transaction data dependent on investor group and gaining insights about those investor groups on aggregate.

The disposition effect literature (see e.g. [2–8]) identifies many different types of investor groups which may or may not be overlapping. The literature also identifies many socioeconomic factors, educational factors, trading style and investment portfolio level factors etc. that can all affect the level of the disposition effect. Thus, it becomes especially important to prepare the large volume of stock market transaction data in a way that it can be easily analysed by statistical methods. As the result of feature engineering, various metrics could be calculated from the data set that can be, in turn, used in the subsequent analysis. Data preparation and calculation procedures and algorithms should be also prepared to make analysing the transaction data set feasible because data analysis needs can change (i.e. differently defined investor groups can become of interest) and new data is added when new transactions become available.

The goal of the thesis is to design and develop database and calculation procedures and algorithms that are necessary for transforming the data into the format and storage that can be at a later stage easily used for making estimations of the disposition effect. The developed procedures include data transformations at the database level as well as complex calculation algorithms necessary for calculating different metrics (e.g., position and portfolio level investment returns etc.) about each distinct investor in the data set.

The developed procedures will be validated by running a large number of simulations using a real stock market data set. This enables to estimate the different disposition effect under various scenarios. The simulations will focus on simulating hypothetical stop-loss orders (i.e., "what if… then…" type of scenarios) to see whether using stop loss orders would increase of decrease the disposition effect and how such orders would affect the investment returns of investors.

The contribution of the thesis includes feature extraction for the disposition effect estimations; developing and demonstrating detailed procedures necessary for the disposition effect; developing the necessary calculation algorithms; and obtaining new insights about the disposition effect by running various "what if… then…" type of simulations. A unique perspective of the current work is that the use of similar stock market transaction data sets is limited to only a small number of countries for which similar data is available. The insights obtained from the simulations provide novel information about the disposition effect and how the stock market performance is linked to the disposition effect.

The thesis is structured as the following. Chapter 2 gives and overview of the literature and describes various methods how the disposition effect can be calculated. It lists various metrics based on literature, which should be calculated to investigate the possible causes of the disposition effect. Chapter 3 focuses on the methodology of applying various methods identified in Chapter 2. Chapter 4 proposes the design of the database and necessary procedures that need to be developed to achieve the aim of the thesis. The necessary procedures are developed and presented in Chapter 5. The procedures are validated by running a large number of trading simulations in Chapter 6 by using the real transaction data and placing hypothetical stop-loss orders.

2 Literature review

Current chapter gives an overview of the literature regarding the disposition effect and lists various factors that can affect the disposition effect. I will use the information to generate a number of variables from the transaction data to be able to conduct further analysis of the disposition effect. The chapter also explains various statistical methodology that is used to get estimates about the disposition effect. The methodology dictates the form to which the data must be converted in order to estimate the magnitude of the disposition effect.

2.1 Overview of the disposition effect

The disposition effect is a bias that investors tend to liquidate winning positions too early and hold on to losing positions for too long. It was first documented by [1] and is widely explained by the prospect theory of [9] which implies that investors make their decisions based on the reference price (usually their purchase price) rather than various economic reasons.

Suppose an investor buys a stock with a price of 10 euros. If the investor is acting economically in a rational way, the investor should sell the stock based on economic or financial events or outcomes. For example, sell the stock when the investor thinks that the economic or financial outlook worsens and thus the price starts to decline; or for example, when the investor has earned enough investment return on that investment. However, when the investor sells the stock based on whether the stock position is in a loss or in a gain relative to the purchase price, such action may not be rational. It is not possible to control for all external reasons or motivations for selling a stock, but given enough observations, it is possible to estimate the probability of selling the stock based on a large number of observable events or news. If it turns out that after controlling for various other factors, the probability of selling the stock in a loss (e.g. with a price of 9 euros) is lower than selling the stock in a gain (e.g. with a price of 11 euros), the investor is regarded to exhibit the disposition effect.

Thus, the disposition effect can be described as being a probabilistic measure which compares the probability of selling the losing stock position to the probability of selling a winning position (given other conditions being the same). The condition when the probability of selling a losing position is lower, is called the disposition effect. If the probability of selling a winning position is lower, it is called the reverse of the disposition effect. The assumed situation by the economic and finance theory is when the purchase price of the stock does not play a role and when taking into account the measurement and estimation

error, the probability of selling a losing position is roughly equal to the probability of selling a winning position.

Given the probabilistic nature of estimating the disposition effect, a large number of observation is needed to be able to statistically estimate the probabilities of selling a losing or a winning positions because many other (financial) factors can affect the probability of selling a stock position as well. There are hundreds or even thousands of factors that can play a role in affecting the selling position. Obtaining an accurate estimation of the disposition effect requires to control for as many other factors as well which introduces a large number of possible features in the data that must be extracted before estimating the disposition effect.

In addition to the prospect theory [9] based explanation, alternative explanations of the disposition effect are provided by for example by [3,10,11]. The explanations mentioned here all require that the reference price must be calculated to be able to estimate the magnitude of the disposition effect. However, it is debated what should be the reference price. Generally, the average purchase price of each individual stock position is used as the reference.

Other possible explanations of the disposition effect include the hypothesis of contrarian strategy [12]. This includes the belief that stocks revert to the mean and thus previous returns of the position can become factors of the disposition effect. It is not generally stated in the literature, which periods of previous price returns should be considered but for example [5,13], use a number of various time intervals that can be considered.

The disposition effect has been explained by rebalancing needs [14]. However, that explanation has not been supported by the more recent literature. [11] propose an explanation which includes mental accounting combined with backward looking optimization. None of those works provide additional variables that can be derived from the data set.

One of the most important works in the area is [7] which was one of the first empirical research that used a large real world transaction data set. [7] finds that investors demonstrate a strong preference for realizing winners rather than losers but does not find evidence to support the hypothesis of the desire to rebalance portfolios, or transaction costs [7]. He also finds that by realizing winners rather than losers is not justified by the subsequent portfolio performance and is disadvantageous to investors. This implies that calculating various portfolio performance metrics becomes necessary as well.

When identifying which metrics and factors should be derived from the data, I consider a large number of previous empirical work in the area. Many of the authors use similar variables, thus I provide an overview of the most relevant and earliest works that have been using such variables. Most of the recent work relies on what has been proposed in the previous literature, but none seems to be covering a comprehensive set of factors or variables. Having an overview of various features, enables me to develop necessary procedures to extract relevant information from the raw data. It also provides information which metrics could be calculated about investor portfolios that would be relevant for subsequent analysis of the data.

Number of previous studies about the disposition effect compare the disposition effect among genders (e.g. [4,15] among the earliest and most well-known authors). Large number of studies (e.g. [8,13,16,17] [18,19]) study investor trading behaviour in various settings. [4,20–25] study the disposition effect in relation to the sophistication level of investors or the amount of wealth the investor holds.

From the perspective of data extraction requirements, this would yield to using any variables that relate to differentiating between investor groups or types of investors. In addition, portfolio size would be an indicator distinguishing either individual investors from professional investors; or more sophisticated investors from less sophisticated investors.

[6] concentrate on the stock market performance issues related to the disposition effect. Some research (e.g. [26,27]) relate disposition effect to the general market conditions (including economic cycle or uncertainty) or market returns. [28] study how various trading strategies relate to the disposition effect.

From the perspective of data extraction requirements, that would mean trying to measure and capture various metrics related to the market cycle and measuring market returns in addition to portfolio returns. It would be more difficult to extract information about trading strategies, as this would be more likely a task for trading pattern recognition and trying to define various strategies. However, it would still be possible to record various trading related metrics such as trading frequency, trading size and similar.

2.2 Methodology for estimating the disposition effect

There are various methods that can be used for estimating the disposition effect. The earlier studies (e.g. [7]) use ratio analysis by calculating the proportion of investment gains realized and the proportion of losses realized and compared the proportions. Subsequent studies (e.g.

[5]) utilized logit methodology and then the focus switched to survival analysis [4]. Each of the method requires slightly different data setup but logit and survival analysis require quite similar data preparation.

Much of the disposition effect literature has switched to using survival analysis as the main methods (see for example [2,4]). In most cases the Cox proportional hazard model [29] is used to estimate the magnitude of the disposition effect. Survival analysis enables to estimate how long a stock is typically held in a portfolio and estimate the effect on various factors on the probability of selling the position. The advantages of survival analysis is that it uses all the data available for each trading data – not only the selling decision - and thus takes into account also the time dimension of each selling decision.

Cox proportional hazard model is semiparametric which means that the baseline probability of selling is not directly estimated, and the model does not impose any structure on the baseline hazard. Parameters associated with various factors affecting the probability of selling a stock position are estimated using the Cox's partial likelihood approach [29]. [30] provide details about estimating the proportional hazard model.

Cox proportional hazard model allows for fixed and time-varying covariates. This means that when creating the data set for analysis, variables values should be calculated for all periods when changes in variable values occur. The specification of a typically used Cox proportional model is the following [31]:

$$h(t, X, Z_t) = h_0(\mathrm{t}) \exp\left(X\beta + Z_t\gamma + \varepsilon_t\right) \quad (1)$$

where h is the hazard rate, X and Z_t are vectors of fixed and time-varying covariates and h_0 is the baseline hazard. The Cox proportional hazard model allows censored observations so that data can be analysed before all investors have sold the stock position of interest.

Some literature such as [5] use logistic regression to estimate the disposition effect. Survival analysis and logistic regressions use similar data set up where the outcome variable is binary indicator of whether a sale of the position occurred. Input variables can be binary, categorical or continuous. Logistic regression does not take into account the time dimension of the data which is a required attribute in the proportional hazard model.

[7,26] use *PGR-PLR* ratio analysis which can be found in many subsequent studies as well. Compared to survival analysis and logistic regressions, *PGR-PLR* ratio analysis requires

different data set up, uses less data and can be considered more simplistic in nature. One of the reasons why numerous studies still use the methodology come from the relatively simple nature of using the methodology and in many cases not as detailed data set is available as can be used for the current study.

[7] constructs a portfolio of securities for which the purchase date and price are known for each date a sale in the portfolio occurs [7]. The methodology counts realized gains, realized losses, paper gains, and paper losses for each day a sale of a position occurs in the portfolio. After that the proportion of gains minus the proportion of losses realized is calculated and a higher proportion of realized losses compared to realized gains indicates the disposition effect. The methodology can be applied both at aggregate as well as individual level, although the result is more meaningful at the aggregate or group level.

The methodology requires calculating the following metrics at the group or individual level and for a period of interest (which can span from covering the whole sample period or be limited to a shorter period):

- *RG - Number of realized gains*
- *PG - Number of paper gains*
- *RL - Number of realized losses*
- *PL - Number of paper losses*

After calculating the above metrics, the proportion of realized losses (PLR) and the proportion of realized gains (PGR) can be calculated as [7]:

$$PLR = \frac{RL}{RL + PL} \tag{2}$$

$$PGR = \frac{RG}{RG + PG} \tag{3}$$

[7] finds that the reference point from which gains and losses are determined can include the average purchase price, the highest purchase price, the first purchase price, or the most recent purchase price [7]. The measure of the disposition effect comes from subtracting the proportion of realized gains from the proportion of realized losses. Whether the result is positive ($PGR - PLR > 0$), then it is assumed that the disposition effect can be observed for the group or individual. [7]

2.3 Feature extraction based on the literature

The previously reviewed literature includes various approaches how to estimate the disposition effect as well as features in the data that can be used for analysing the disposition effect. Based on the literature, I develop the following requirements and features that will be used as an input for constructing the TO-BE data model and to the estimation of the disposition effect at the later stages of the work:

- A stock position is defined starting when the first know purchase in the data takes place in a particular stock
    - Any possible purchases before the starting point in data are discarded
- Each different (company's) stock is defined as a different stock position
- A stock position is defined ending when the position goes to zero
    - It is possible that an investor has purchased a part of stock position before the start date of the used transaction data and thus is able to seemingly sell more shares than has been recorded in the stock position. In such cases such selling transactions are discarded.
- During the duration of the stock position the following metrics are recorded:
    - The average purchase price (volume weighted);
    - The highest purchase price;
    - The lowest purchase price;
    - The first purchase price;
    - The last purchase price;
    - First date the stock was purchased;
    - Each buy and sale transaction date;
    - The number of shares in the position;
    - Whether the position is in profit or loss or breakeven;
        - Calculating profit or loss of a position must take into account various different possibilities to calculate the purchase price and the comparison price (i.e. current market price);
    - The percentage return on the position;
    - The above metrics are recorded for each trading day and each stock position;
- The stock position can be built up with multiple purchases and liquidated with multiple sales until the balance goes to zero

- At the time when a sale occurs in the portfolio of an investor, the following metrics are calculated:
    - The number of realized gains;
    - The number of realized losses;
    - The number of unrealized gains;
    - The number of unrealized losses;
- For each day an investor holds at least one stock position the following data is recorded:
    - Closing price of each stock;
    - Highest price of each stock;
    - Lowest price of each stock;
    - Opening price of each stock;
    - Return of the stock for previous time periods (using various time intervals);
- For each day an investor holds at least one stock position the following metrics are calculated:
    - The number of positions in a portfolio
    - The size of the portfolio in monetary terms
    - The return of the portfolio on the given day
    - The return on the portfolio since inception
    - Whether the portfolio is in profit or loss or breakeven
    - Whether the portfolio is experiencing a large gain or loss
    - Whether the portfolio has experienced a large gain or loss previously
- For each investor the following descriptive characteristics are calculated:
    - The average number of stocks in the portfolio
    - The average size of the portfolio
    - Experience measured in time (days or years) since entering the market
    - Trading frequency and size indicators such as:
        - The average number of trades made in a year
        - The average monetary size of a trade
        - The average holding period in days
        - The average size of a buy or sale transaction separately
        - The average monetary size of a buy or sale transaction separately

The requirements will be used in the next chapters when developing data transformation procedures to bring the raw data into the necessary format required for the subsequent analysis using the methodology described in the current chapter.

3 Methodology and data

The current chapter briefly describes the methodology and process used to achieve the goals of the thesis. More details about the methodology of estimating the disposition effect is provided in the previous chapter. Relevant literature, principles and methods of data modelling; and development of data transformation procedures is provided in the following dedicated chapters. Current chapter mainly serves the purpose of summarizing the overall approach and process.

3.1 Methodology

The thesis can be viewed as a data science project developed iteratively by the author. The business problem for the thesis is improving the estimating of the disposition effect and gaining more insights how the disposition effect and investment returns are affected when using stop-loss orders.

To gain more insights about the business problem (the disposition effect), a number of steps need to be taken. The overall process can be classified into four main categories of activities:

1. Activities related to requirements development
2. Data modelling
3. Development of data transformation and calculation algorithm procedures
4. Running simulations and estimating the disposition effect

The activity diagram is provided in Figure 1.

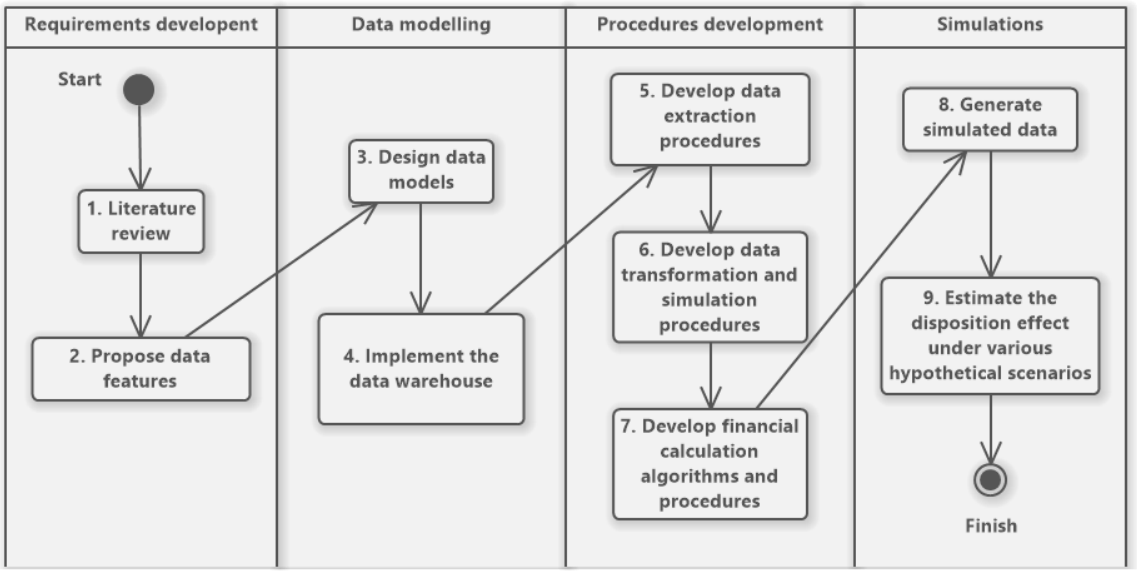


Figure 1. Activity diagram of steps and activities conducted in the thesis

Requirements' development

The activities related to requirements development (activities No 1-2 in Figure 1) relate to working with the literature to propose new features that can be extracted from the raw data. Overview of the literature and the requirements were provided in the previous chapter. In some cases feature engineering involves reducing the number of features or combining various features to make the number of data attributes more manageable for further analysis. In the current business context, the number of data attributes will be increased to be able to take into account as many economic and financial factors as possible. which may affect the disposition effect and that will be used in the final fourth step (running simulations and estimating the disposition effect).

Data modelling

Data modelling (activities No 3-4 in Figure 1) includes designing the data models based on the suggestions of the data warehouse literature and taking into account the requirements of the business problem. The proposed data warehouse data model is implemented using a relational database MariaDB with further details provided in the following chapter.

Programming tasks

Development of data transformation and calculation algorithm procedures (activities No 5-7 in Figure 1) involve a separate development project written in Java programming language. This step can be further broken down into activities of which data extraction (activity No 5) includes loading the original raw data which is a relatively simple process. It also includes extracting additional data from the stock exchange webpage which also does not include technical complexities but requires some coding.

Development of data transformation processes (activity No 6) includes also developing data simulation algorithms. The raw data needs to be transformed and expanded to be able to estimate the disposition effect in a later stage. The complexities involved in this activity include writing algorithms which deal with exceptional cases when the raw data is not consistent (i.e., includes trades which cannot occur based on the data, e.g. selling more stocks than are in the portfolio), calculating various statistics: over all cases, over certain investor group cases, or over certain investor related cases. As data transformation procedures generate the data which is stored in the data warehouse, the procedures also include code which is used to simulate hypothetical "what if… then…" type of scenarios that are called when running simulations in the last step.

Developing financial calculation algorithms and procedures (activity No 7) involves development of various financial algorithms that use the methodology described in [32–34]. Some of the required calculations include algorithms with optimization algorithms (e.g., calculating the money-weighted return), or running linear regressions (e.g., calculating the risk adjusted performance metric or Jensen's alfa) among other things. Calculating investment returns introduces a large number of edge test cases and problematic cases when precise calculations do not always become possible (e.g., multiple local optimums) and the developed algorithms need to deal with such cases to be able to produce meaningful estimations that later serve as inputs (data features) for the disposition effect estimation. The developed procedures have to be universal enough that those could be run also after generating data by running simulations which is done in the last step.

Simulations

The final step of running simulations (activities No 8-9 in Figure 1) involves running the previously developed procedures to generate over three hundred million observations (cases). I will then use survival analysis (Cox proportional hazard model) to estimate the disposition effect under various scenarios using Stata software and the some of the previously developed financial calculation procedures to calculate the investment returns now based on the results of the simulations.

Project development methodology

The project was developed iteratively. Although the activity diagram shows sequential activities, the development of the activities was not always sequential. For example data modelling requirements changed during the development of the data transformation procedures. Financial calculation algorithms proposed new requirements for the data transformation procedures etc. New emerging edge cases for investment return calculations required addition of unit tests and sometimes changes to the transformation or calculation algorithms. The final step of running the simulations was necessary to verify that all of the developed models and processes work as necessary and to provide new findings published in the finance scientific literature.

3.2 Data

The thesis uses stock market transaction data from Nasdaq Tallinn from 2004 to 2012 that contains all stock market transactions made during that period. The number of transactions is approximately 1.3 million. The necessary data transformations increase the number of

observations used for the disposition effect calculations to approximately sixty million. Further, running simulations that are used to validate the procedures increase the volume of observations to over three hundred million. In total, the transactions cover the investment and trading activity of 33 843 distinct investors, including both institutional and individual investors. The data covers all investors that have participated in the Estonian stock market during the period.

4 Designing the data warehouse

Current chapter focuses on designing the necessary data warehouse that can be used for storing the final data in the format suitable for statistical analysis. The chapter specifies data models necessary for storing the raw data as well as the data models for the data warehouse. The data warehouse includes transformed data and results (i.e. calculated metrics) from various calculations.

4.1 Practical considerations for applying statistical methods

When applying survival analysis methods or logistical regression, the key variable in the specification is usually referred to as the trading loss indicator (TLI) or trading gain indicator (TGI) in the literature. There is a large combination of possible ways to calculate e.g. the TLI as the reference price can either be the average price of a position, the latest price of the position etc. Comparisons of the reference price can be made against each day's closing price or highest or lowest price. As the number of combinations is large, a practical approach would be to choose one or few main approaches and allow for possible modifications if that is deemed necessary in the future. Thus from the practical point of view, the following considerations have been taken into account based on [2,4].

I record losses only when the reference price is higher than the closing price of the day. If a sale has occurred, I use the transaction price instead of the closing price and compare that to the reference price. In cases when there were no transactions made with the stock during the day, I use the previous known closing price instead of the current closing price. The similar principle applies for recording gains. I record a gain only when the reference price is lower that the closing price or the transaction price. The variable TLI takes the value of 1 when the position is in loss and value 0 otherwise. The variable TGI takes the value of 1 when the position is in profit (gain) and value 0 otherwise. On special occasions, it is possible that both variables are zero when the reference price is equal to the closing or the transaction price.

The TGI or the TLI must be recorded on each day, for each stock position of each account. This increases the number of observations greatly compared to the number of observations in the raw transaction data set.

4.2 AS-IS data model

The data available for the thesis comes in the format suitable for recording stock market transactions. It is exported from the stock exchange system and comes in the denormalized

format. It means that each transaction includes information about the person making the transaction in addition to transaction related information (i.e. price, quantity, stock etc.). Information about the person have been added to the transaction data set during export procedure from the stock market database. Thus, information about the person is static in the transaction data set.

The subsequent analysis needs a data model that is suitable for analytical purposes and contains clearly more information. Statistical software which will be used for analysing the data and calculating the disposition effect metrics requires denormalized data, spanning over a large number of columns, each containing information about a particular stock market transaction; the person (or institution) making the transaction; or about the portfolio of the person. A suitable data model for such a task is star schema popularized by Kimball [35].

The initial data model is shown in Figure 2. I extract stock prices from Nasdaq Baltic webpage. The initial data model enables to associate transactions with the stock price information on the transaction date. Transaction info includes the actual transaction price but stock price info includes the opening, closing, high and low price of the same day for a stock. Table 1 and 2 provide detailed information about the initial data structure. After extraction, I store data in a MariaDB database. I follow the practice suggested by [36] and store the raw data with its initial structure. This requires saving the same information in a different structure for the data warehouse.

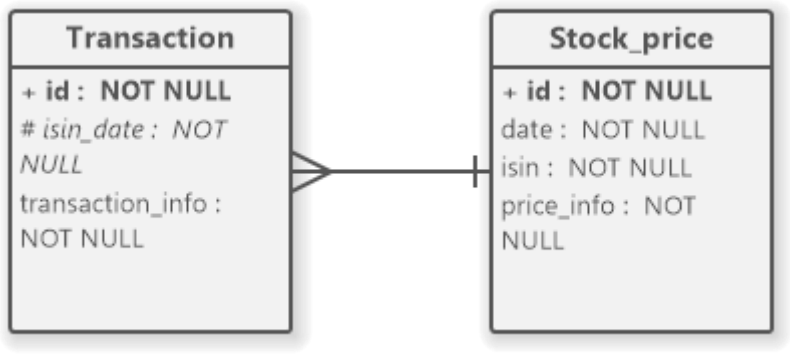


Figure 2. AS-IS data model

Table 1. Structure of raw transaction data provided by the stock exchange

Attribute	Type	Comment
id	varchar(50)	Identifier for the legal entity making the transaction
birty_year	varchar(10)	Birth year if available (for physical persons only)
gender	varchar(2)	Gender if available
OwnerType	varchar(2)	Owner type provided in the original data
residence	varchar(4)	Information about residence provided in the original data
s_isin	varchar(20)	Identifier for the stock
s_name	varchar(255)	Name of the stock
price	float	Transaction price
quantity	bigint(255)	Transaction quantity
rec_time	datetime	Transaction time
direction	tinyint(4)	Direction of the transaction (buy or sell)
date	date	Date of the transaction

The stock exchange provided only essential information collected about the transaction. However, it is necessary to calculate other metrics connected with the stock that was included in the transaction. This requires collecting information about the stock price, calculating stock returns and adjusting stock price for dividends. It is possible to collect data from Nasdaq Baltic web page but as it turns out, it does not include proper data adjustments. Thus, the data model needs to be adjusted as well by including additional attributes for properly adjusting the data.

Initial price data includes more information than later needed. Although it would be possible to record daily gains and losses compared to either high, low or close price; only one of those will be used at a time. Thus, preserving the original data after extraction is useful, but when developing further procedures only one comparison price needs to be recorded. Since the most common comparison price (i.e. the price used for determining whether the position is in a loss or gain) is the close price, I use a naming convention and call it the close price in the final data model. It would be possible to replace the closing price with another price (e.g. high or low) to change the transformation process in the staging area of the data warehouse that will be constructed next.

Table 2. Structure of the price data extracted from the stock exchange webpage

Attribute	Type	Comment
isin	varchar(20)	Identifier for the stock
date	date	Date
average_price	double	Average price
open_price	double	Open price
high_price	double	Highest price of the day
low_price	double	Lowest price of the day
close_price	double	Close price
last_price	double	Last price (equal to close price at the end of the day if close is not missing)
adj_factor	double	Adjustment factor provided by the stock exchange
adj_last	double	Adjusted last price provided by the stock exchange
change_prc	double	Percentage change in price
trades	int(255)	Number of trades
volume	double	Number of shares traded
turnover	double	Value of the trades conducted during the day
currency	varchar(20)	Currency

4.3 Data warehouse design principles

Designing the data warehouse can include conceptual and logical models and be approached by using various methods. For example [37] proovide an overview of different approaches and discuss various problems that can be encountered during the process. When considering data modelling in a narrower perspective, the requirement of further analysing the data using various statistical software mandates the use of a schema where queries are fairly simple and fast. Additionally, data transformations and aggregations are needed to extract additional information that would require changing the initial data model anyways. Thus, I follow the practices of multidimensional modelling and utilize the star schema based approach as described by [35].

The benefits of star schema is that is quite simple to create, can use the existing relational database and it is understandable both to the information technology specialist as well as the business side [38]. All the benefits are relevant in the context of the current task.

The star schema uses fact tables to store measurement events and additional information associated with the measurement even is stored in the dimension tables [35]. It is also possible to use fact tables that do not contain any facts but the main purpose is to join different

dimensions with a certain event during which no measurements were created [39]. Fact tables can have different level of aggregation. Some fact table can already include aggregated results at the same time when some fact tables can include event measurements without aggregation. Higher level of granularity can mean a large number of data rows, which slows down queries [39]. Although it might be necessary to take into account that information connected with events can change in time, which would introduce the necessity to use slowly changing dimensions. Currently it is not the case as the stock exchange only provided static information about the persons making the transactions. However, when recording price information that acts as a dimension to stock market transactions, the considerations of slowly changing dimensions must be taken into account as well.

4.4 TO-BE data model

The new data model needs to include all information possible about each transaction. As described in the previous section, using the normalized data model would not have any clear benefits in the current case. Thus, I use a star schema that includes fact tables and dimension tables. The use of star schema means that data is stored in the denormalized format.

The main events of interest are stock market transactions. To estimate the disposition effect, an event should be recorded on each day for each stock position of every person regardless of whether a transaction occurred on that day with that stock or not. Thus, the actual events are not transactions anymore but daily observations about each stock in the portfolio of every investor. I call it portfolio events or shortly portfolio facts. It is possible that two investors hold exactly the same stock in their portfolio with even exactly the same purchasing price. However, such observations should still be recorded separately because describing information about the investor or investor's portfolio (i.e. the dimensions related to the fact) is likely to be different. The data model with the star schema is shown in Figure 3.

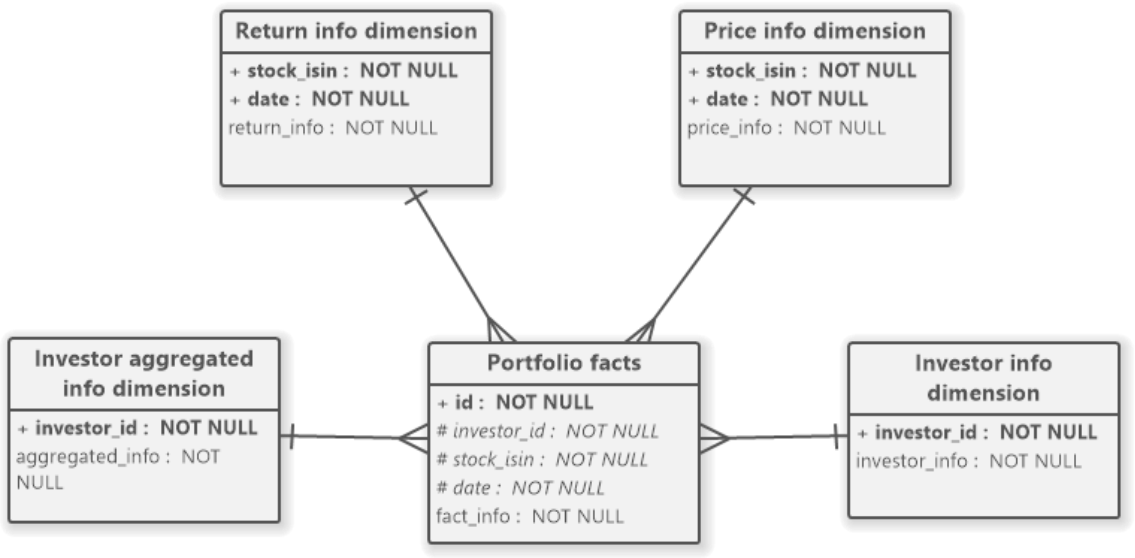


Figure 3. Star schema with portfolio facts

As data need to be prepared for analysis with different statistical methods, a single fact table is not able to store information in a suitable format for all analysis methods. Thus, I use another much simpler fact table for *PGR-PLR* ratio analysis requirements and call it Odean fact table based on the author who popularized the method [7]. Such an approach requires calculating only one record per each investor. Since the Odean fact table includes much less information than the portfolio fact table, it does not share all the dimensions but only the ones related to the investor. Thus, I will provide a separate star schema model (see Figure 4), which uses the Odean fact table for events (see Table 3 for structure of the table).

Figure 4. Star schema with Odean facts

The dimensions include information about the investor. For example, the investor can be a private person or an institution. And there are various types of institutions as well. A private person has a birth year and gender information recorded.

I use a separate dimension about the stock price. Although recorded stock prices on its own can be considered as facts, it is possible that a fact table in one star schema acts as a dimension in another star schema. This is the case here as well. When viewed separately, price data would be a fact table in a schema where information about each stock would be stored in dimension tables. As I do not need to store information about different stocks rather than prices and dividend and split adjustments (which will be done in the programming code), I will not include a separate star schema about stock prices in my data model and use stock price as a dimension for the daily portfolio events. I do show the stock information entity in my model for clarity and showing the relation between different entities. However, I do not add a separate table in the physical model because all of the tables will be using the same stock ISIN code as a key which will make joining those tables easier with SQL queries rather than introducing separate joins with a stock information table.

Stock price dimension includes only price information for a particular day. However, it is necessary to have information about previous stock returns as well. It would be too time consuming to calculate stock returns within statistical software of when making queries from the data warehouse. Thus, it is much more feasible to pre-calculate all of the necessary information and similarly to stock price data, include that in a separate dimension table. In special cases it would even be interesting to gain access to information what has happened to the stock price after an investor has sold the position. Thus, I will include both historical and forward-looking (i.e. returns calculated with future stock prices which were actually not known on the date of the record) stock return information in the stock return dimension.

Both stock prices and stock returns are slowly changing dimensions. The only feasible option in this case is to use multiple keys: stock ISIN and date. This enables to join multiple facts with a single stock price or stock return observation. As it turns out later when testing the physical database implementation, using multiple non-unique keys with tens of millions of rows does not produce fast enough queries, thus in a physical model a single unique key is generated and used. The unique key is a combination of stock ISIN and date.

Another dimension includes aggregated information about an investor. Such information is calculated based on the transactions of the investor. It could be considered a separate fact

table as well but in the context of the disposition effect calculations it will act as a dimension table. It is possible to filter out different types of investors based on the aggregated information such as average return, or frequency of trades etc. As it is calculated separately, it is not feasible to include such information in the investor info dimension, but store it also separately in investor aggregate dimension, which uses the same key (i.e. account number as an identificator) as the investor info dimension.

It would be possible to record some of the information about the portfolio in dimensions as well. However, since the generation and calculation of such information is part of the same process generating the portfolio fact table, I will not store such information is a dimension table but in the fact table. As [35] discuss, it is appropriate to include denormalized information in the fact table. Given that the subsequent statistical analysis requires fully denormalized data anyways and the physical data size is not as big to impose any storage restrictions, I follow an approach of having a lot of information in the main fact table. The structure of the main fact table is shown in Table 4.

Table 3. Structure of the Odean fact table.

Attribute	**Type**	**Comment**
id	bigint(255)	Primary key
investor_id	bigint(255)	Identifier for the investor making the transaction
realized_losses	int(255)	
realized_profits	int(255)	
unrealized_losses	int(255)	
unrealized_profits	int(255)	

Table 4. Structure of the portfolio fact table.

Attribute	Data type	Default	Comment
id	bigint(255)	None	Primary key
investor_id	bigint(255)	None	Identifier for the investor making the transaction
stock_isin	varchar(20)	None	Stock identifier
sold	tinyint(2)	None	Dummy (0 or 1) whether the position was sold
bought	tinyint(2)	None	Dummy (0 or 1) whether the position was bought
time0	int(255)	None	Time of the record
time1	int(255)	None	Time of the record +1 day
loss	tinyint(2)	None	Dummy (0 or 1) whether the position is in a loss
profit	tinyint(2)	None	Dummy (0 or 1) whether the position is in a profit
position_return	double	None	Position return
time_bought	int(255)	None	Time the position was opened
portfolio_size	bigint(255)	None	Portfolio size in EUR
porfolio_size_prev	bigint(255)	None	Portfolio size in EUR the previous day
t_date	date	None	Date of the record
number_of_trades	int(255)	None	Number of trades made so far by the person
days_return	double	None	Return in the current trading day
num_stocks_portfolio	int(255)	None	Number of different stocks in the portfolio
num_stocks_portfolio_prev	int(255)	None	Number of different stocks in the portfolio the day before
portfolio_loss	tinyint(2)	None	Dummy (0 or 1) whether the portfolio is in a loss
portfolio_profit	tinyint(2)	None	Dummy (0 or 1) whether the portfolio is in a profit
portfolio_smallgain	tinyint(4)	None	The following attribute indicate (a dummy of 0 or 1) whether the portfolio is currently or has ever encountered a (very) small/large gain or loss
portfolio_smallgain_ever	tinyint(4)	None	
portfolio_largegain	tinyint(4)	None	
portfolio_largegain_ever	tinyint(4)	None	
portfolio_verylargegain	tinyint(4)	None	
portfolio_verylargegain_ever	tinyint(4)	None	
portfolio_smallloss	tinyint(4)	None	
portfolio_smallloss_ever	tinyint(4)	None	
portfolio_largeloss	tinyint(4)	None	
portfolio_largeloss_ever	tinyint(4)	None	
portfolio_verylargeloss	tinyint(4)	None	
portfolio_verylargeloss_ever	tinyint(4)	None	
forceful_sale	tinyint(2)	None	Dummy (0 or 1) whether the sale occurred in mandatory takeover
count_of_active_halfyears	tinyint(2)	None	The number of 6-month periods the person has been active
active2004_2005	tinyint(2)	0	Dummy (0 or 1) whether the person was active during those particular years
active2006_2007	tinyint(2)	0	Dummy (0 or 1) whether the person was active during those particular years
active2008_2009	tinyint(2)	0	Dummy (0 or 1) whether the person was active during those particular years
active2010_2012	tinyint(2)	0	Dummy (0 or 1) whether the person was active during those particular years
subsample_id	varchar(100)	None	The identifier of the generated subsample, used in the case when multiple subsamples are generated based on different time periods and stored in the same table

Depending on how many various time periods are used for calculation stock returns, the stock return dimension can have very many rows. If necessary, it is possible to define additional time periods and thus even add even more columns. Compared to the structure of stock price dimension (see Table 5), the structure of the stock return dimension requires calculating and recording much more information (see Table 6).

Table 5. Structure of the stock price dimension table.

Attribute	**Type**	**Comment**
stock_isin	varchar(20)	Identifier for the stock
date	date	Date
orig_close	double	Close price in the original data
adj_close	double	Adjusted close provided by the stock exchange
adj_factor	double	Adjustment factor provided by the stock exchange
split	double	Stock split factor
close	double	Final adjusted close price used for calculations
adjustment	double	Final adjustment factor

Table 6. Structure of the stock return dimension table.

Attribute	**Type**	**Comment**
stock_isin	varchar(20)	Identifier for the stock
date	date	Date
return_day_1-m1	double	return of the stock t...t-1 days
return_day_m1-m2	double	return of the stock t-1…t-2 days
return_day_m2-m3	double	
return_day_m3-m5	double	
return_day_m5-m10	double	
return_day_m10-m20	double	
return_day_m20-m62	double	
return_day_1	double	return of the stock t…t+1 days
return_day_1-2	double	return of the stock t+1…t+2 days
return_day_2-3	double	
return_day_3-5	double	
return_day_5-10	double	
return_day_10-20	double	
return_day_20-62	double	

Dimensions regarding the information about an investor also contain a lot of information. The investor info dimension that contains data extracted from the data provided by the stock market is relatively straightforward (the structure is shown in Table 7). It records invest type and basic information.

Investor aggregate information, on the other hand, contains a lot of different aggregations and calculated metrics. I follow the feature engineering principles as described by [40] and try to come up with as many different metrics as possible. Some insights are provided by the literature described in Section 2.1, but the literature does not provide a comprehensive list. Thus, the data features shown in Table 8 and Appendix III can serve as a more comprehensive list of possible features that can serve as inputs to various data analysis methods (including statistical methods usually used for estimating the disposition effect as well as machine learning methods that can be utilized for other purposes). As the number of attributes is very large, I show only the categories in Table 8 and provide a comprehensive list in Appendix III.

Table 7. Structure of the investor info dimension table.

Attribute	**Type**	**Default**	**Comment**
id	int(255)	None	Identifier for the investor making the transaction
OwnerType	varchar(2)	None	Owner type provided in the original data
residence	varchar(4)	None	Information about residence provided in the original data
birth_year	int(255)	None	Birth year if available (for physical persons only)
male	tinyint(1)	0	The following are dummy indicators about type or characteristics of the person (legal entity) making the trades
female	tinyint(1)	0	
male_inc_foreign	tinyint(2)	0	
female_inc_foreign	tinyint(2)	0	
private_person	tinyint(1)	0	
corporation	tinyint(1)	0	
local_investor	tinyint(1)	0	
foreign_investor	tinyint(1)	0	
state_owned	tinyint(1)	0	
private_corp	tinyint(1)	0	
fund_account	tinyint(1)	0	
nominee_account	tinyint(1)	0	
client_account	tinyint(1)	0	

Table 8. Summary of the structure of the investor aggregate info dimension table.

Attribute	Type	Default	Comment
id	int	None	Identifier of the investor
ipo_participation	tinyint(1)	0	Participated in IPO (different IPOs separately or in any)
average_return	double	None	Time or money weighted return; or risk adjusted performance (RAP) - with and without transaction costs
portfolio_beta	double	None	Portfolio beta (calculated from different models)
std_portf_ret	double	None	Standard deviation of portfolio return (calculated from different models)
std_excess_ret	double	None	Standard deviation of excess return (calculated from different models)
std_market_ret	double	None	Standard deviation of market return
market_ret	double	None	Market return
holding_period	int	None	2004-2012 without transaction cost holding period days
number_of_sales_counted	int	None	Number of sales for which we know also buys (max 1 sale per stock per day)
number_of_buys_counted	int	None	number of buys counted (max 1 sale per stock per day)
portf_smallgain_ever	tinyint(1)	0	Has portfolio ever been in gain
portf_largegain_ever	tinyint(1)	0	Has portfolio ever been in more than 20% gain
portf_verylarge-gain_ever	tinyint(1)	0	Has portfolio ever been in more than 100% gain
portf_smallloss_ever	tinyint(1)	0	Has portfolio ever been in loss
portf_largeloss_ever	tinyint(1)	0	Has portfolio ever lost more than 20%
portf_verylarge-loss_ever	tinyint(1)	0	Has portfolio ever lost more than 50%
fakesale_ever	tinyint(1)	0	Has any stocks been ever forcefully bought out from portfolio
active2004_2005	tinyint(1)	0	Investor active in 2004-2005 (at least 3 counted trades)
active2006_2007	tinyint(1)	0	Investor active in 2006-2007 (at least 3 counted trades)
active2008_2009	tinyint(1)	0	Investor active in 2008-2009 (at least 3 counted trades)
active2010_2012	tinyint(1)	0	Investor active in 20010-2012 (at least 3 counted trades)
total_stock_days_portf	int	None	(number of days stocks in portfolio * number of stocks in portfolio for each specific day). It can overestimate the number if sales were conducted in multiple parts (different days).
avg_portf_size	double	None	Average portfolio size in EUR (average size before each sale transaction)
avg_stocks_portf	int	None	How many different stocks in portfolio on average (average number before each sale transaction)
avg_holding_per_d	int	None	How many days each position was held in portfolio (counted by every sale transaction)
portf_turnover_rate	double	None	totals_stock_days_portfolio/avg_holding_peridod_days. It should indicate how many times the stocks were changed during the period.
number_of_transactions	int	None	Each transaction is counted which can overestimate the number if one trade had multiple counterparties (it is calculated separately for sales and purchases and the sum of both)
turnover	double	None	the sum of all transactions in EUR (it is calculated separately for sales and purchases and the sum of both)
avg_sum_of_transaction	double	None	Average transaction size EUR (it is calculated separately for sales and purchases and the sum of both)
stock_ownership	tinyint(1)	0	has ever owned a stock (various stocks)
tr_num_period	int	None	number of trades in certain period (various periods)
tr_2004_Q1	int	None	number of trades in 2004_Q1
first_trade_date	date	None	date of the first trade made

To be able to calculate returns in different ways (as presented in Table 8 and Appendix III), it is necessary to record cash flows resulting from purchases and sales of stocks. Since returns can be calculated in various ways and using various algorithms, it turns out to be useful to save intermediate results for cash flows. Those become inputs to other algorithms that will be applied on the data. Thus, I introduce another table in the physical model (the structure is shown in Table 9) which was not included in the data warehouse data model shown previously. It will be used mainly for temporary storage and testing return calculations.

Table 9. Structure of the table storing cash flows from transactions.

Attribute	Type	Comment
investor_id	bigint(255)	Identifier for the investor making the transaction
date	date	
cash_flow	double	
year	int(255)	

Return calculations are one of the most challenging aspects of the project. It is not feasible to store return calculations straight into the data warehouse, but another intermediate table is needed. The same table can be used for storing simulation results as well. Structure of the table for storing return calculations as well as simulation results is shown in Table 10. I calculate both time-weighted and money-weighted returns. After that I use Sharpe's model [32], RAP model [33] and Jensen's model [34] to calculate risk adjusted returns. Since risk adjusted returns enable to better compare investors, it is important to take risk into account as well. I use standard deviation as a risk metric and compare portfolio returns to stock market index returns to calculate excess returns when that is required by the used models. As one of the used approaches also requires comparing portfolio and market risk, I also calculate stock index standard deviation.

Table 10. Structure of the table storing return calculation results and simulation results.

Attribute	**Type**	**Comment**
investor_id	bigint(255)	Identifier for the investor making the transaction
twr	double	Time-weighted return
return	double	Cumulative return
mwr	double	Money-weighted return
portfolio_beta	double	Portfolio average beta
beta_model_twr	double	Time-weighted alpha return from Jensen's model
beta_model_mwr	double	Money-weighted alpha return from Jensen's model
rap_twr	double	Time-weighted alpha return from RAP model
rap_mwr	double	Money-weighted alpha return from RAP model
sharpe_twr	double	Time-weighted alpha return from Sharpe's model
sharpe_mwr	double	Money-weighted alpha return from Sharpe's model
std_portfolio_return	double	Standard deviation of portfolio return
std_excess_return	double	Standard deviation of portfolio excess return
std_market_return	double	Standard deviation of market return
annualized_index	double	Annualized index return
index_return	double	Cumulative index return
holding_period_days	int(255)	Number of holding days
year	int(255)	Year
loss_stopped	int(255)	How many times is loss stopped
profit_stopped	int(255)	How many times profit stopped
cash_in	double	Cash flow in
cash_out	double	Cash flow out
loss_limit	double	Limit loss boundary
profit_limit	double	Limit profit boundary

5 Data transformation procedures

Data transformation procedures are written in Java, using Apache Maven project. Hibernate framework [41] is used to interact with the database. In case of more complex queries that cannot be that easily implemented with data persistence framework, pure SQL queries are used instead. I also use simple SQL queries when constructing queries on the fly dependent on input conditions; and for convenience in cases when simple updates are necessary to keep track of generation process.

5.1 Development process

I followed an iterative development process. The code was rewritten and refactored numerous times. The main problems arise with duplicate code as many of the portfolio fact calculations and portfolio return calculations are similar but not necessarily exactly the same. Thus, the first iterations introduced a lot of code duplication which was eliminated during later iterations.

The development process was further complicated by the fact that some assumingly simple calculations turned out to be clearly more challenging when using real data. For example, there were missing prices in the data which required decisions what should the methods return in such cases. A lot of work and effort went into developing the code related to the business logic. It required thorough considerations and consulting with the financial literature to develop the necessary code which would correspond to the business logic.

For example, an investor could be extremely profitable in one day by making very small transactions which result only a small profit in monetary terms (e.g. 10 euros) but in percentage terms it could be very high (500%). Even if the results were computationally absolutely correct, certain outlier detection checks had to be implemented as original raw data may have erroneous information as well.

Best efforts have been made to follow the principles of clean code and meaningful naming principles as suggested by [42,43] and constant and iterative refactoring as popularized by [44]. As the project has still not matured and is being iteratively improved and developed further, further refactoring will be done during the process.

5.2 Interacting with the database

I use MariaDB version 10 for storing the data. The use of that database is arbitrary and based on the convenience of having such a database server already set up and running. As the code uses JPA with Hibernate framework along with very simple pure SQL statements, it would be possible to migrate the project on any other common database.

However, data provided by the stock exchange come with strict confidentiality requirement and can only be used in certain secured server environment. Some of the data and structure of the data coming from the stock exchange is not shown in the current thesis because of confidentiality agreements of using the data.

The project includes a number of entity classes corresponding to the data model shown in the previous section. I approach the project from the perspective of database first and use the Hibernate framework with Intellij IDE to generate all the necessary POJO[1] classes corresponding to database tables. However, it becomes necessary to break some of the entities into smaller pieces than shown in the data warehouse model. Such a requirement comes from the limitation to calculate only proportion of aggregated information about an investor at a time. Inserting only the calculated data into a separate table is faster and less error prone.

The problem with database first approach is that the data warehouse structure has been changing throughout the iterations. This is also one of the reasons why pure SQL statements are being still used in places where Hibernate would help to simplify the code as well as speed up queries. This is also a reason why I did not generate POJO classes for fact tables as regenerating the classes after each data warehouse change interfered and slowed down the development process. Thus, I followed an approach that I used JPA with Hibernate in places where speed was an important factor. For example, those were cases when inserting cash flow information or return calculations. Since the number of insertions became quite large, simple SQL queries turned out to be around 10-1000 times slower than Hibernate and would mean hundreds of hours of running time instead of current couple of hours.

5.3 Structure of the project

The project is organized in packages based on the functionality or type of the classes. Entity classes belong to a separate package. Data transformation procedures must be started by classes called “generators”. Those are runnable Java classes which are organized in a separate package. There are two types of generators. Fact generators are related to generating

[1] Plain Old Java Object

information needed for the fact tables. Other generators can be broadly called as dimension generators as they are mostly related to calculating necessary metrics for the dimension tables of the used star schema.

Generator classes call "calculators" which are Java classes with primary functionality of calculating something. Most of the calculations are related to calculating portfolio metrics, returns or similar. I also use a separate third-party library JXIRR [45] for calculating money-weighted return. This is referred as to XIRR calculator. Some of the calculations also utilize Apache Commons Mathematics Library [46]. The actual code also uses some helper classes, such as for connecting to the database or as custom data structures, which are not separately shown on the package or class diagrams. The package diagram is presented in Figure 4.

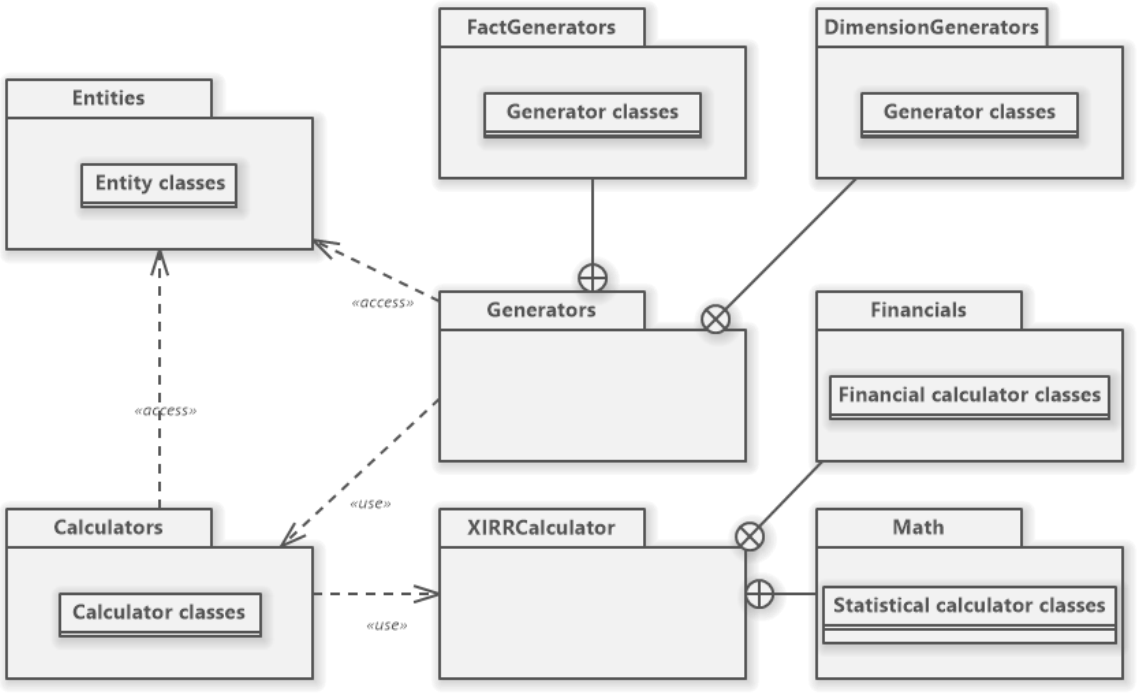


Figure 4. Package diagram of the project.

More detailed view of the project structure is shown on Figure 5 which lists all the Java classes in the main package. The figure only omits tests. I developed unit tests for return calculations. Reference to the project code is provided in Appendix IV.

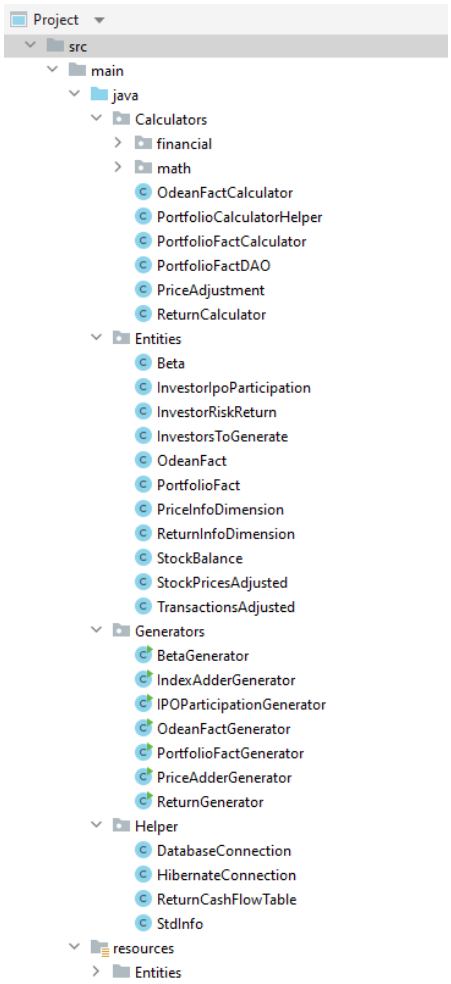


Figure 5. Project structure.

5.4 Procedures for data warehouse fact table creation

The data warehouse contains two fact tables. The first one is called PortfolioFacts and the second one OdeanFacts. I use separate generators and calculators for making the necessary calculations and inserting information to the fact table. Fact generators are basically just runnable classes which have configurable fields and pass on that information to calculator classes. At the current stage, the generators are not meant to be called from the command line or compiled into executable files because the information (table names, parameter values) is currently hard coded. If there were separate business side users of the project, it would be easy to extract current fields into arguments for the main() method. Currently there was no such requirement from the end user(s) of the program.

Both PortfolioFactCalculator and OdeanFactCalculator classes utilize heavily a common helper class PortfolioCalculatorHelper. The latter includes all common methods that both

portfolio calculator classes utilize. In reality, about 80% of the methods and code found in the PortfolioCalculatorHelper class is related to the required functionality of the PortfolioFactCalculator as the related fact table records hundreds of times more data rows and has significantly more data fields. Figure 6 presents the class diagram for fact generation process. The figure lists all methods of the classes but some of the attributes are omitted for keeping the picture more readable. A separate data structure that acts similarly to a data transfer object is used in those classes to avoid meaningless duplication of fields.

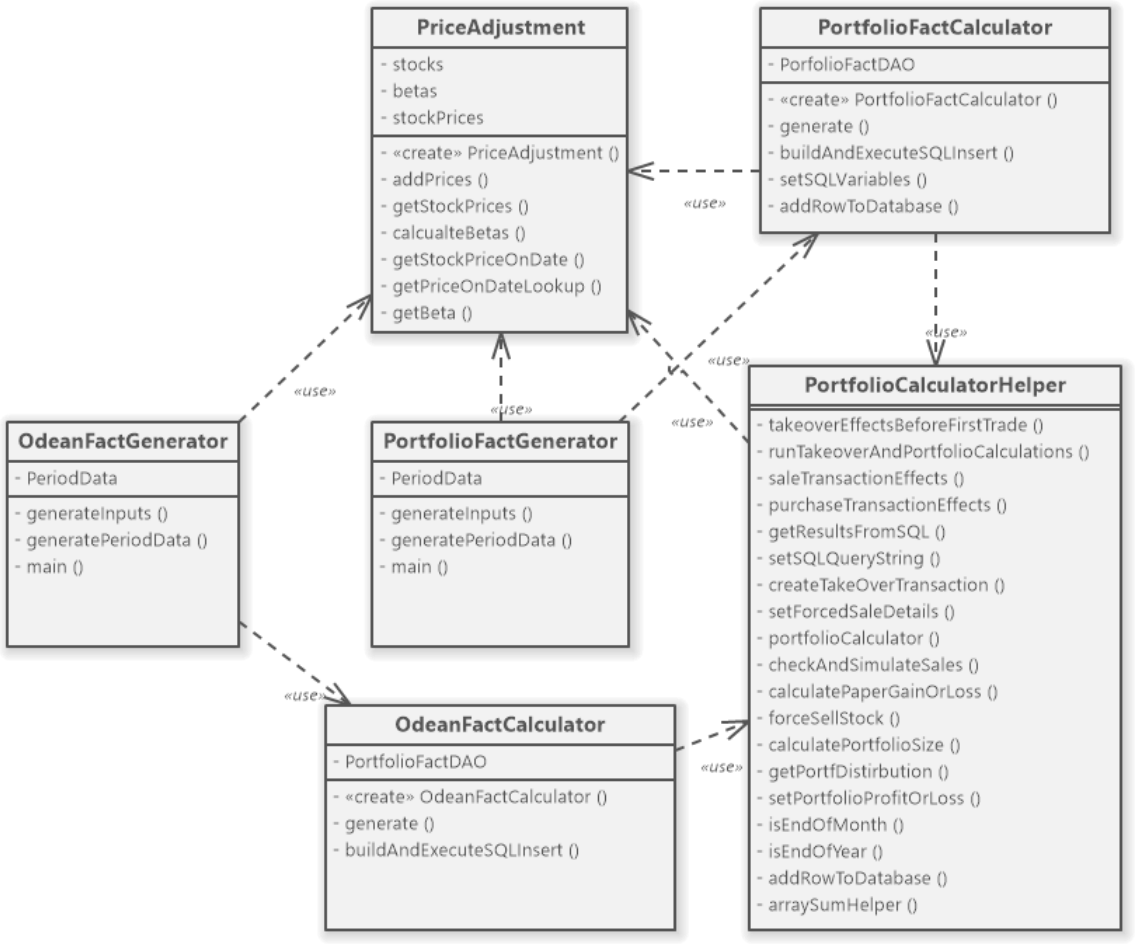


Figure 6. Class diagram of fact generation operations.

I provide an extract from the code to illustrate the fact generation process in Figure 7. This is the main methods used for generating portfolio facts. The code utilizes helper class but includes fact table specific methods on its own. This is one of the simplest methods but illustrates a large number of steps that have to be taken to generate one row of the fact table.

If possible, each of the steps is extracted into a separate method. This approach helps to reuse at least some of the methods but also introduces a risk of double responsibility. Two of the fact generators need slightly different functionality and the current architectural choice is that the shared functionality is included in the helper class.

For example, the method portfolioCalculator in the class PortfolioCalculatorHelper makes calculations for both PortfolioFactCalculator and OdeanFactCalculator. They shared part of the method is about 70%. OdeanFactCalculator adds about 5% of the code to the method and PortfolioFactCalculator additional 25% of the code which the other fact calculator does not use. Since there are no clear ways to make the method into smaller pieces which could be reused by both fact calculators by keeping the rest of the pieces separate, I have made a clear architectural decision to include both functionality in the same helper function. The negative consequences are mitigated because the main functionality of the method is to calculate various metrics. The caller calculator can just discard unnecessary metrics if such facts are not needed.

```
public void generate(DatabaseConnection connection, boolean saveFlag) {

    try {

        ResultSet resultsFromSQL = PortfolioCalculatorHelper.getResultsFromSQL(connection, dao);
        while (resultsFromSQL.next()) {
            PortfolioCalculatorHelper.takeoverEffectsBeforeFirstTrade(resultsFromSQL, dao);
            dao.tradeNumber += 1;

                if (resultsFromSQL.getInt( columnLabel: "direction") == 1) { // purchase transaction
                    PortfolioCalculatorHelper.purchaseTransactionEffects(resultsFromSQL,dao);
                    setSQLVariables();
                }
                else { // sale transaction
                    setSQLVariables();
                    PortfolioCalculatorHelper.saleTransactionEffects(resultsFromSQL,dao);
                }
        } // end of transactions
        resultsFromSQL.close();

        // run takeover and portfolio calculations for remaining distinct days to take takeover effects into account
        while (dao.counter < dao.dates.size()) {
            PortfolioCalculatorHelper.runTakeoverAndPortfolioCalculations(dao);
        }
        if (saveFlag) {
            buildAndExecuteSQLInsert(connection);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Figure 7. Java code example of generating fact table.

5.5 Procedures for return calculations

Filling the fact tables with accurate information is the highest priority of the development project. However, the most challenging part involves making various calculations related to stock portfolio return calculations. The main structure of the code is similar to generating fact table contents. For return calculations, the user can run the class ReturnGenerator with various parameters. The generator calls a calculator class ReturnCalculator which utilizes helper functions from static PortfolioCalculatorHelper class. Additionally it uses Apache Commons Mathematics Library and third-party open source XIRR package for various calculation tasks. An illustrative class diagram of the classes involved in return calculations is provided in Figure 8.

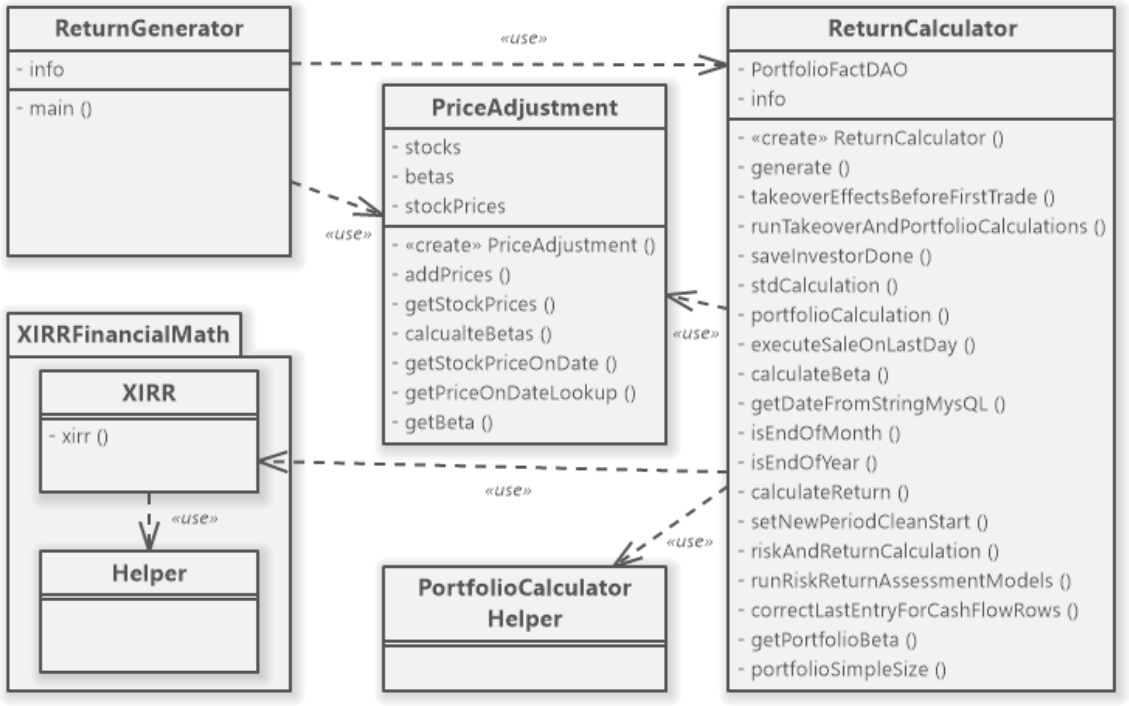


Figure 8. Class diagram of return calculation operations.

One of the most challenging aspects of calculating returns is the use of various different approaches. Possible different approaches and methods were mentioned at the end of the previous chapter. One of the first design choices was to separate all different methods into separate classes or methods. In reality, it turned out that various return calculation methods

use mostly overlapping data and share many commonalities. During an iterative process the code for calculations had to change many times.

There is a problem with a real-world uncleaned data. This meant that simple unit tests that were written in the first place did not capture nearly enough edge cases for calculations. It was not that easy to write tests as well because to know the correct answer to return calculation, it was necessary to calculate returns using various other tools (e.g. MS Excel). However, some of the return calculations use goal seek type of optimization procedures which may have multiple optimums. Since for example MS Excel and the utilized third-party library use slightly different algorithm for the same calculations, it was possible that the result from the program and calculation result by hand (or other software) were also slightly different. In the end, I just added a more test cases and allowed for larger error.

Additionally, I had to introduce various checks in the code to capture cases when an investor had some stocks in the portfolio but the data did not have record of the investor buying it. It was a known limitation of the data which could be quite easily handled when recording portfolio facts. What made it easier was that only a loss or profit had to be recorded. However, return calculations had to return precise results and any special cased had to be handled by the code before running the final return calculations. My approach was to get the results which made sense and were logical from the business perspective. Thus, to get return calculation results without big outliers. This meant that some outliers were discarded in the end.

Figure 9 illustrates the calculation of portfolio beta, which is a small part of return calculations. It shows an extract of code where I have to use two different methods to get the price of the stock if the first method does not give a desired result. The overall challenge here was that beta calculations do not always yield meaningful results for stocks that do not trade very often (as is the case for many stocks in the used data set). This was overcome by calculating betas separately and making the necessary corrections and saving the reviewed betas to the database. This was done in addition to running the necessary regressions for beta calculations in the code.

However, even that did not solve the whole problem. Some of the stocks did not have prices for all trading days. That problem had many possible solutions. One would be to adjust the stock database and write a procedure that replaces missing values with last known values. I did not choose that because it may introduce other inaccuracies as I would be overwriting

missing values. I chose an approach that will look up the last known price value from the database. Although it might be the most correct approach, it had a drawback of being relatively slow compared to taking a price of a predefined date. The slowness would probably not be noticeable for a small-scale operational system. But I was running simulations producing tens of millions of rows and each of which utilized the same function multiple times. In the end the feasible approach was to try to retrieve the price of a stock of the exact day and if this fails, call a slower lookup function as illustrated in Figure 9.

```
private Double getPortfolioBeta(Double portfolioSize) {
    Double betaTemp = 0.0;
    for (String stock : dao.stocks) {
        Double sPrice = null;
        try {
            sPrice = dao.PriceInfoDimension.getStockPriceOnDate(stock, dao.dates.get(dao.counter));
            if (sPrice == null) {
                sPrice = dao.PriceInfoDimension.getPriceOnDateLookup(stock, dao.dates.get(dao.counter));
            }
            if (portfolioSize != 0.0) {
                betaTemp += (( dao.portfolios.get(stock) * sPrice) / portfolioSize) * dao.PriceInfoDimension.getBeta(stock);
            }
        } catch (NullPointerException e) {
            e.printStackTrace();
        }
    }
    return betaTemp;
}
```

Figure 9. Dealing with special cases when prices may not be returned with a straightforward method and look-up function is needed.

Due to the design choice of not duplicating the code for various calculation methods, some of the methods in ReturnCalculator class are quite long which makes them hard to follow. An illustration of such a code is provided in Figure 10. It is possible that during further iterations it would be reasonable to break some of the code once again apart (as it was during the first iterations). By now all calculation inaccuracy problems are probably solved. Thus, the initial problem of changing partly duplicate code in multiple places can have less problematic effects. However, the project is still lacking broad enough code base for basic operations, which is one of the prerequisites to be able to break the code more easily apart. The current development has followed the principle of producing minimal necessary specialized code which has some drawback for the described situation.

```
private void riskAndReturnCalculation(Double portfolioSize, double returnFactor, Date startDate, Date endDate, double[] values, double[] dates,
    if (portfolioSize > 1) {
        Calendar cal1 = Calendar.getInstance();
        cal1.setTime(startDate);
        int year = cal1.get(Calendar.YEAR);
        cal1.set( year: year + 1,  month: 0,  date: 1);
        prevEndDate = cal1.getTime();
    } else {
        prevEndDate = null;
    }
    Double portfolioBeta = 0.0;
    for (ReturnCashFlowTable th : cashFlowRows) {
        if (daysPortfolio != 0) {
            portfolioBeta += th.beta * (double) ((double) th.numberOfDays / (double) daysPortfolio);
        }
    }
    investorRiskReturnEntity.setPortfolioBeta(portfolioBeta);
    Calendar cal1 = Calendar.getInstance();
    cal1.setTime(startDate);
    investorRiskReturnEntity.setYear(cal1.get(Calendar.YEAR));

    if (cashFlowRows.size() > 0) {
        int days = Days.daysBetween(new DateTime(startDate), new DateTime(endDate)).getDays();
        double annualized = Math.pow(returnFactor, 365.0 / (double) daysPortfolio) - 1.0;
        double annualizedIndex = Math.pow(indexReturn, 365.0 / (double) daysPortfolio) - 1.0;
        XIRRData data = new XIRRData(values.length,  guess: 0.1, values, dates);
        double xirrValue = XIRR.xirr(data);

        if (!Double.isNaN(xirrValue) && !Double.isInfinite(xirrValue)) {
            investorRiskReturnEntity.setMwr(xirrValue);
        }
        if (!Double.isNaN(annualized) && !Double.isInfinite(annualized)) {
            investorRiskReturnEntity.setTwr(annualized);
        }
        if (!Double.isNaN(returnFactor) && !Double.isInfinite(returnFactor)) {
            investorRiskReturnEntity.setReturnFactor(returnFactor - 1.0);
        }
```

Figure 10. Extract of Java code for calculating returns

5.6 Other data transformation procedures

In addition to portfolio fact generation and return calculations, other procedures had to be run first. This was necessary to have the data in the first place which could be then used for the previously described calculations. This includes recording stock prices; stock index values; calculation of stock betas; and recording participation in initial public offerings (IPOs) and adjusting price tables to reflect such information. As it turned out, otherwise return calculations do not yield accurate results. All of the generator classes which were created during the project are shown in Figure 11. The figure also lists methods of those classes except for generators which have been presented in more details already on previous figures.

The project includes also a number of entity classes. Figure 12 shows the most important of those by grouping them according to the functional area of the program. The high level view of the entity classes is provided for illustrating the separate purposes of creating different entity classes. Helper entity classes are omitted from the figure.

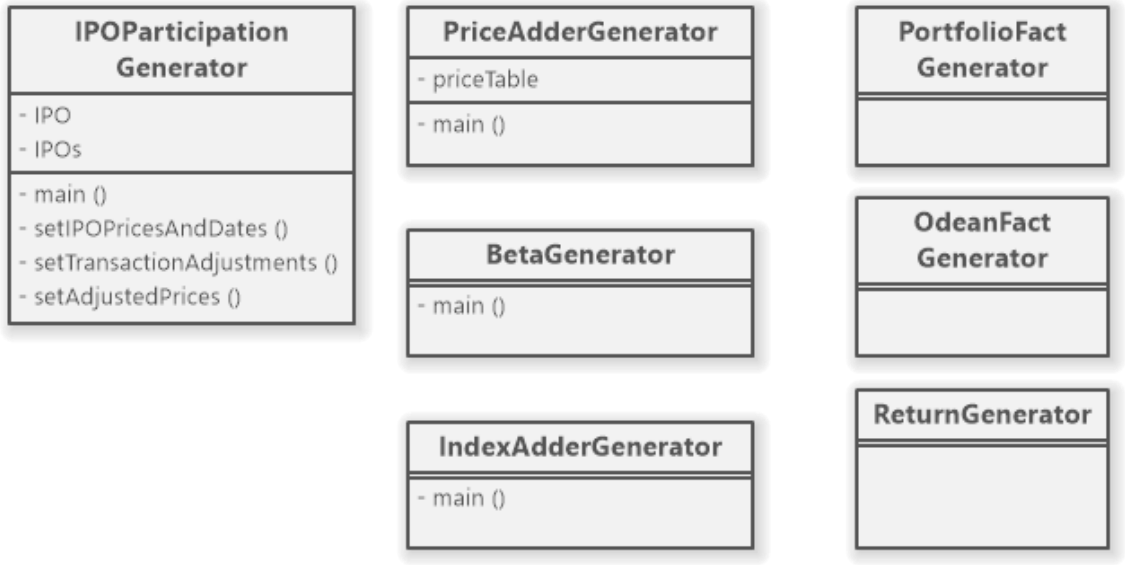


Figure 11. Class diagram of all generator classes.

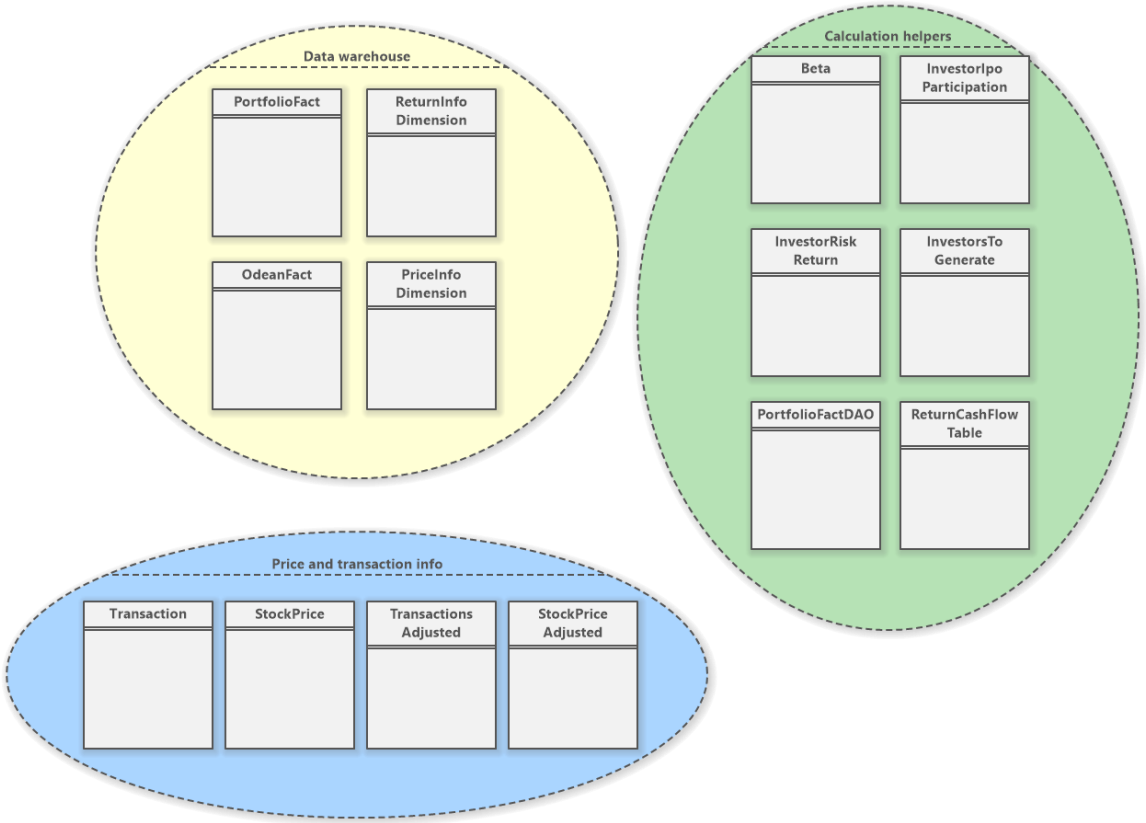


Figure 12. Class diagram of all entity classes grouped by relation to the functional area of the program.

5.7 Testing

The project includes unit tests for testing return calculations. The project does not include any functional or acceptance tests. Unit tests cover the return calculation functionality and no unit tests were developed for portfolio fact generation. Part of the portfolio fact generation process uses the same methods which are used for return calculations but no separate unit tests are written for individual methods. This is one of the limitations of the project.

The project has been thoroughly manually tested during all iterations. The output results of the program have been examined and verified by multiple finance researchers who use the output for their research. Bugs have been reported and corrected numerous times. In addition, the output results will be used during simulations described in the next chapter to provide verification that the program works and produces meaningful and useful results.

```
@Test
public void test_10086316() {
    String isik = "10086316";
    ReturnCalculator rc = new ReturnCalculator(isik, priceAdjustment, distinctDates, startDate, limitDate, endOfMonthDates, endOfYearDates,
    rc.generate(connection, session, saveToDatabase: false);
    InvestorRiskReturn it = rc.investorRiskReturnEntity;
    assertEquals( expected: 1.6829918716301617, it.getTwr(), EPSILON);
    assertEquals( expected: 0.5376614242040201, it.getMwr(), EPSILON);
}

@Test
public void test_10163903() {
    String isik = "10163903";
    ReturnCalculator rc = new ReturnCalculator(isik, priceAdjustment, distinctDates, startDate, limitDate, endOfMonthDates, endOfYearDates,
    rc.generate(connection, session, saveToDatabase: false);
    InvestorRiskReturn it = rc.investorRiskReturnEntity;
    assertEquals( expected: 0.23591892026646777, it.getTwr(), EPSILON);
    assertEquals( expected: 0.23591892026646777, it.getMwr(), EPSILON);
}

@Test
public void test_10119422() {
    String isik = "10119422";
    ReturnCalculator rc = new ReturnCalculator(isik, priceAdjustment, distinctDates, startDate, limitDate, endOfMonthDates, endOfYearDates,
    rc.generate(connection, session, saveToDatabase: false);
    InvestorRiskReturn it = rc.investorRiskReturnEntity;
    assertEquals( expected: -0.8667061097101663, it.getTwr(), EPSILON);
    assertEquals( expected: -0.8667061097101663, it.getMwr(), EPSILON);
}

@Test
public void test_10091174() {
    String isik = "10091174";
    ReturnCalculator rc = new ReturnCalculator(isik, priceAdjustment, distinctDates, startDate, limitDate, endOfMonthDates, endOfYearDates,
    rc.generate(connection, session, saveToDatabase: false);
    InvestorRiskReturn it = rc.investorRiskReturnEntity;
    assertEquals( expected: -2.7720528751329354E-6, it.getTwr(), EPSILON);
    assertEquals( expected: -2.7720528814612067E-6, it.getMwr(), EPSILON);
}
```

Figure 13. Extract of Java code for testing return calculations

The developed unit tests for return calculations include 27 different test cases for calculating returns. Those test cases range from simple cases to various edge and borderline test cases. Some test cases require the use of a small number of observations, and some require a large

number of observations. The test cases are chosen to include both positive and negative results and similar and greatly varying results dependent on the calculation methodology. An extract of the code that includes the unit tests in provided in Figure 13.

Further manual testing was conducted using MS Excel to check the results of risk-adjusted returns obtained by using different models.

6 Simulations regarding the disposition effect

To verify the data transformation procedures, simulations are used to create artificial trade data. Statistical software (Stata 16) is used to measure the disposition effect with simulated data and compare obtained portfolio return information. The simulations include using a real transaction data set described earlier and simulating artificial limit loss or limit profit type of orders. The chapter briefly summarizes the results that have been published by the author in [31].

6.1 Simulation set up

I use the real data set of transactions provided by the stock exchange. The total number of investors in nearly 34 thousand and in total they have made over 1,3 million trades during the period from 2004-2012. There are slightly more purchase transactions than sale transactions. Descriptive statistics of the data set can be seen from Table 11.

Table 11. Descriptive statistics

Type of investor	Number of investors	Number of trades	Number of buys	Number of sells	Average number of trades per year
Individual	28 957	553 848	294 357	259 491	4.1
Corporate	4 792	536 194	262 282	273 912	9.2
Client accounts	43	236 252	119 504	116 748	137.6
Investment funds	24	1 053	401	652	8.7
Government related	27	332	190	142	3.4
Total	**33 843**	**1 327 679**	**676 734**	**650 945**	**4.9**

I run simulations to create artificial sale decisions with different scenarios. Each scenario includes certain threshold levels such as 5%, 10%, 15%, 20% or 25% level stop loss orders. If a position that an investor hold has reached that level of loss, it will be automatically sold by the simulation. From the investor perspective such a simulation is retrospective and if an actual investor would know that a stock position at a certain loss level would be automatically sold by the broker, it would affect investors real life decisions as well.

The simulation scenario is the following. An investor opens a stock position. If the market price of the stock falls below the stop loss threshold level (which is 5-25%, depending on

the scenario), the simulation sells the stock position using the closing price of the day as the transaction price. The stock position is then considered to be completely closed out. Any subsequent purchases of the same stock by the same investor are regarded to be a new position with a new purchase for which the same stop loss principles apply. In real life, the subsequent purchase and sale decisions of an investor would be affected by such transactions, but this cannot be considered in the simulations. When the real transactions show that an investor is selling a position which has already been sold by the simulation, those real-life transactions will be discarded as the investor does not have any stock in the virtual portfolio of the simulation.

The motivation of running such simulations comes from the work of [47] who show in their experimental study that when investors use stop loss orders, their disposition effect can be smaller. Similar findings are provided by [48].

As I record a row in the data base for each stock position of every investor on every trading day, the initial number of transactions of about 1,3 million increases to over three hundred million observations in total. This includes all various scenarios. A logit methodology would use all of the observations to estimate the disposition effect. I use the Cox proportional hazard model, which means that I am using about 3.4 million observations, which are records of selling transactions, and already include the purchase time and price as well.

6.2 Simulation results

After running the simulations and generating the data for various scenarios, I use Stata 16 to estimate the Cox proportional hazard model to get estimates of the disposition effect. I am interested in the hazard ratios for the trading loss indicator (TLI). I report the results in Figure 14. The figure shows the hazard ratios for the TLI of different scenarios. If the hazard ratio of TLI is below one, it is considered that an investor is exhibiting a disposition effect as the probability of selling a losing position is then lower than the probability of selling a winning position. If the hazard ratio of the TLI is larger than one, an investor is not considered to be affected by the disposition effect.

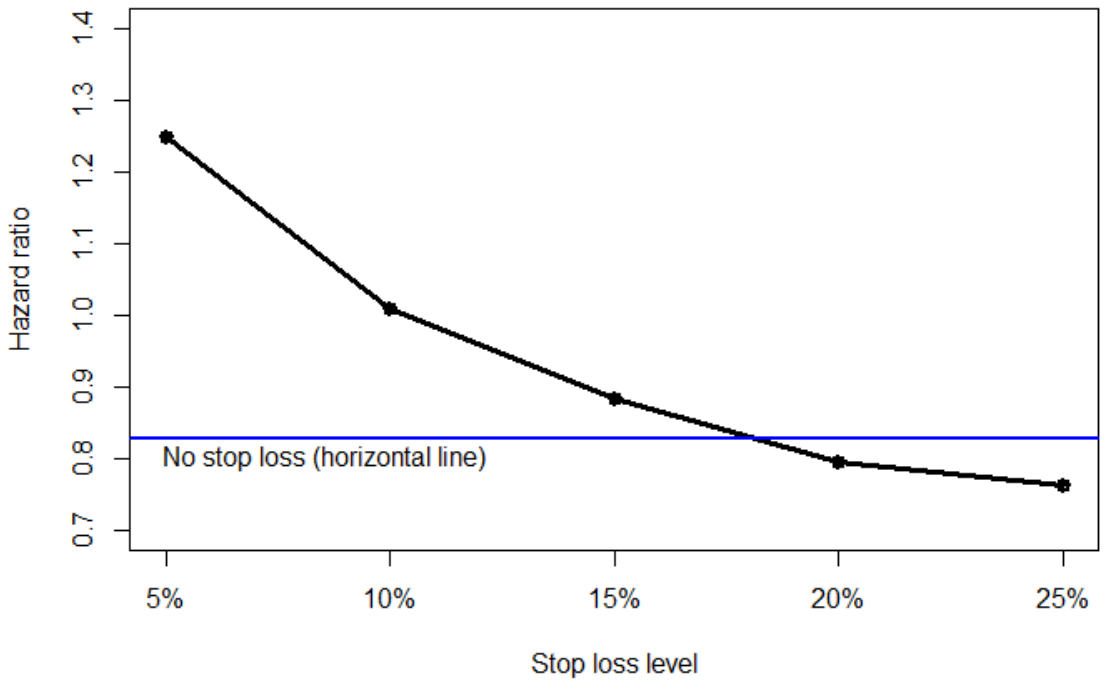
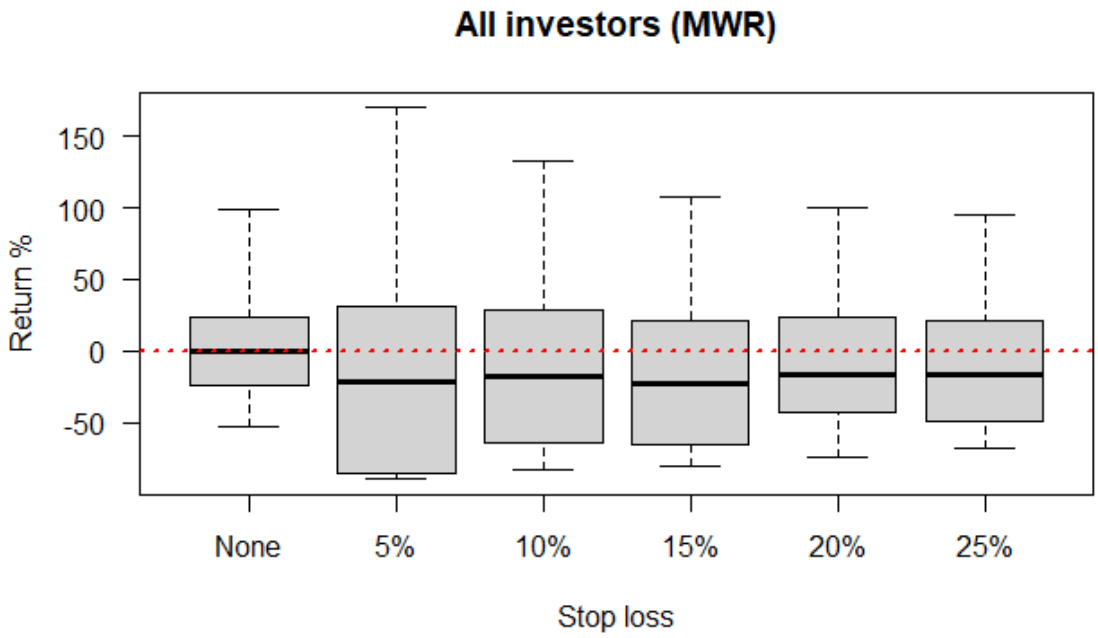


Figure 14. Hazard ratios obtained from Cox proportional hazard model by simulating scenarios with stop loss threshold from 5-25%.

As the Figure 14 shows, the disposition effect estimations change when simulation scenarios change. The horizontal line in the figure shows the hazard ratio for the dataset without using simulated stop loss orders. The obtained estimation of the hazard ratio without simulated stop loss orders is at 0.827, which is at a comparable level of similar results provided in the disposition effect literature. The aim of the thesis is not to analyse the results of the disposition effect estimation (which is provided by the author in [31]) but to develop the necessary procedures and show that the procedures work and are useful.

I also calculate the distribution of average returns of investors using different simulation scenarios Figure 15 presents the results of calculation of annualized money-weighted returns (MWR) and holding period returns (HPR) with different simulation scenarios. The main body of the box shows the upper and lower quartiles of the average returns. The whiskers represent the upper and lower 10% of the average returns and the black line inside the box the median of annualized average returns. Red dotted lines show the median and blue dotted lines the upper and lower quartile of the returns when no simulated stop loss orders are used.



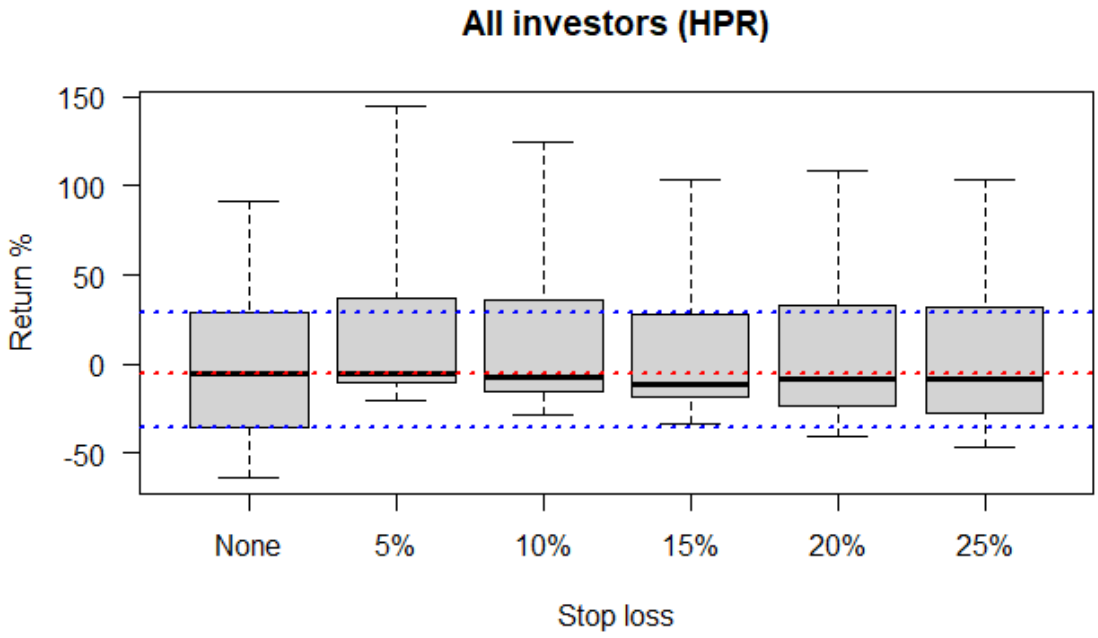


Figure 15. Investor annualized returns obtained by simulating scenarios with stop loss threshold from 5-25%.

The simulations show that the developed data models helped to transform the data into a format which could be easily used by other statistical software. Current simulations do not take advantage of the full capabilities that the developed data warehouse type of approach

offers. It would be possible to merge other data set with the existing data which is the case for results obtained for [31]. Running simple join queries within statistical software enables to query for exactly the right amount of data which corresponds to the necessary conditions set in the SQL query. Current simulations demonstrate the use of various scenarios and only the necessary portion of the data (one scenario at a time) was loaded to the statistical software to save more computer resources for running the estimations.

The simulations help to verify that the data transformation procedures developed in the thesis helped to obtain the necessary data and enabled to make quite complicated and extensive calculations in batch. This made getting descriptive information easy when analysing the results of simulations. The code necessary for running simulations was incorporated into fact table generation procedures. It enables to create observations without using any simulated stop loss order. But at the same time included checks which allow using various different approaches to compare the stock purchase price to a market price or deliberately setting a dynamic price (as e.g. different stop loss levels). This corresponds to the requirements described in Chapter 2.

.

7 Conclusions

The aim of the thesis was to design a data warehouse model and develop data transformation and calculation procedures and algorithms that enable to transform the raw data into the format that enables to easily conduct estimations of the disposition effect. The requirements for the data transformation and storage were developed and described in Chapter 2 based on the disposition effect literature and taking into account various possible other requirements based on real transaction data.

The necessary data warehouse was developed in Chapter 4 using a data model based on star schema. Detailed structure of the physical data model was described and implemented using MariaDB. Data transformation procedures and calculation algorithms were developed in Chapter 5 and were written in Java. The data transformation procedures enabled to generate data for the fact table and dimension tables which are part of the proposed data warehouse system; and to conduct the necessary financial calculations as well as aggregations.

I describe the simulations that I used to validate the suitability of the developed database structure and data transformation procedures in Chapter 6. The simulations use original transaction data set provided by the stock exchange and generate over three hundred million rows in the fact table with various hypothetical scenarios. The simulation results and analysis of the simulation results (which was done by using Stata statistical software) showed that the proposed data models and their physical implementation was appropriate for the task and the developed data generation and calculation procedures worked as expected.

Although efforts were made to follow the best practices of software development, a weakness of the developed solution is that only part of the developed code is covered by automatic tests, although all of the code is thoroughly manually tested. A better unit test coverage is expected to be achieved during later iterations that will extend the basic code base so that current special purpose code can be split into smaller and more general-purpose pieces. Further refactoring of the code will be done during next iterations.

Overall, the aim of the thesis was achieved. The developed data models and data transformation procedures helped to get valuable simulation results. The developed procedures and details of data structure can be helpful to other researchers who possess similar transaction data sets. The most straightforward application is in various cases of academic research, especially in the context of using probabilistic regression models for further analysis of the

data. However, the developed calculation procedures can be useful outside the specific domain as some calculation algorithms can be used as part of a banking or investment evaluation system. Due to the complexity of making portfolio return calculations many of similar systems avoid presenting investors with their return data.

Additionally, the developed procedures enabled to gain novel insights about stop-loss orders and their effect on the disposition effect and investor portfolio performance. This led to the publishing the results in [31]. The current work can be extended by introducing more data to the database and combining various data sets with the current generated data. It is quite easily achieved by the current data warehouse architecture. Further development of various calculation algorithms is possible as well. It is possible to create specialized command line or graphical interfaces for running various data generation and calculation procedures automatically. Such a functionality was not deemed necessary under the current scope of the project.

8 References

[1] H. Shefrin and M. Statman, "The Disposition to Sell Winners Too Early and Ride Losers Too Long: Theory and Evidence," *The Journal of Finance*, vol. 40, 1985, pp. 777–790.

[2] A. Seru, T. Shumway, and N. Stoffman, "Learning by Trading," *Review of Financial Studies*, vol. 23, 2010, pp. 705–739.

[3] M. Kaustia, "Prospect Theory and the Disposition Effect," *Journal of Financial and Quantitative Analysis*, vol. 45, 2010, pp. 791–812.

[4] L. Feng and M.S. Seasholes, "Do Investor Sophistication and Trading Experience Eliminate Behavioral Biases in Financial Markets?," *Review of Finance*, vol. 9, 2005, pp. 305–351.

[5] M. Grinblatt and M. Keloharju, "What Makes Investors Trade?," *The Journal of Finance*, vol. 56, 2001, pp. 589–616.

[6] P.R. Locke and S.C. Mann, "Professional trader discipline and trade disposition," *Journal of Financial Economics*, vol. 76, 2005, pp. 401–444.

[7] T. Odean, "Are Investors Reluctant to Realize Their Losses?," *The Journal of Finance*, vol. 53, 1998, pp. 1775–1798.

[8] Z. Shapira and I. Venezia, "Patterns of behavior of professionally managed and independent investors," *Journal of Banking & Finance*, vol. 25, 2001, pp. 1573–1587.

[9] D. Kahneman and A. Tversky, "Prospect Theory: An Analysis of Decision under Risk," *Econometrica*, vol. 47, 1979, pp. 263–292.

[10] N. Barberis and W. Xiong, "What Drives the Disposition Effect? An Analysis of a Long-Standing Preference-Based Explanation," *The Journal of Finance*, vol. 64, 2009, pp. 751–784.

[11] T. Hens and M. Vlcek, "Does Prospect Theory Explain the Disposition Effect?," *Journal of Behavioral Finance*, vol. 12, 2011, pp. 141–157.

[12] B.M. Barber and T. Odean, "The courage of misguided convictions," *Financial Analysts Journal*, vol. 55, 1999, pp. 41–55.

[13] M. Grinblatt and M. Keloharju, "The investment behavior and performance of various investor types: a study of Finland's unique data set," *Journal of Financial Economics*, vol. 55, 2000, pp. 43–67.

[14] J. Lakonishok and S. Smidt, "Volume for Winners and Losers: Taxation and Other Motives for Stock Trading," *The Journal of Finance*, vol. 41, 1986, pp. 951–974.

[15] B.M. Barber and T. Odean, "Boys will be boys: Gender, overconfidence, and common stock investment," *The Quarterly Journal of Economics*, vol. 116, 2001, pp. 261–292.

[16] M. Grinblatt, M. Keloharju, and J.T. Linnainmaa, "IQ, trading behavior, and performance," *Journal of Financial Economics*, vol. 104, 2012, pp. 339–362.

[17] M. Grinblatt and M. Keloharju, "How distance, language, and culture influence stockholdings and trades," *The Journal of Finance*, vol. 56, 2001, pp. 1053–1073.

[18] R. Garvey and A. Murphy, "The profitability of active stock traders," *Journal of Applied Finance*, vol. 15, 2005.

[19] M.S. Haigh and J.A. List, “Do professional traders exhibit myopic loss aversion? An experimental analysis,” *The Journal of Finance*, vol. 60, 2005, pp. 523–534.

[20] G. Chen, K.A. Kim, J.R. Nofsinger, and O.M. Rui, “Trading performance, disposition effect, overconfidence, representativeness bias, and experience of emerging market investors,” *Journal of Behavioral Decision Making*, vol. 20, 2007, pp. 425–451.

[21] L. Feng and M.S. Seasholes, “Individual investors and gender similarities in an emerging stock market,” *Pacific-Basin Finance Journal*, vol. 16, 2008, pp. 44–60.

[22] T.Y. Cheng, C.I. Lee, and C.H. Lin, “An examination of the relationship between the disposition effect and gender, age, the traded security, and bull/bear market conditions,” *Journal of Empirical Finance*, vol. 21, 2013, pp. 195–213.

[23] H. Choe and Y. Eom, “The disposition effect and investment performance in the futures market,” *Journal of Futures Markets*, vol. 29, 2009, pp. 496–522.

[24] R. Dhar and N. Zhu, “Up Close and Personal: Investor Sophistication and the Disposition Effect,” *Management Science*, vol. 52, 2006, pp. 726–740.

[25] P. Brown, N. Chappel, R. da Silva Rosa, and T. Walter, “The reach of the disposition effect: Large sample evidence across investor classes,” *International Review of Finance*, vol. 6, 2006, pp. 43–78.

[26] C.C. Leal, M.J.R. Armada, and J.L.C. Duque, “Are All Individual Investors Equally Prone to the Disposition Effect All the Time? New Evidence from a Small Market,” *Frontiers in Finance and Economics*, vol. 7, Oct. 2010, pp. 38–68.

[27] A. Kumar, “Who gambles in the stock market?,” *The Journal of Finance*, vol. 64, 2009, pp. 1889–1933.

[28] A. Krause, K.J. Wei, and Z. Yang, “Behavioral Bias of Traders: Evidence for the Disposition and Reverse Disposition Effect,” *EFM 2006: Symposium Behavioral Finance Discussion Paper*, 2006.

[29] D.R. Cox, “Partial likelihood,” *Biometrika*, vol. 62, 1975, pp. 269–276.

[30] D.R. Cox and D. Oakes, *Analysis of survival data*, Chapman and Hall/CRC, 2018.

[31] T. Talpsepp and T. Vaarmets, “The disposition effect, performance, stop loss orders and education,” *Journal of Behavioral and Experimental Finance*, vol. 24, 2019, p. 100240.

[32] W.F. Sharpe, “Mutual fund performance,” *The Journal of Business*, vol. 39, 1966, pp. 119–138.

[33] F. Modigliani and L. Modigliani, “Risk-adjusted performance,” *The Journal of Portfolio Management*, vol. 23, 1997, pp. 45–54.

[34] M.C. Jensen, “The performance of mutual funds in the period 1945-1964,” *The Journal of finance*, vol. 23, 1968, pp. 389–416.

[35] R. Kimball and M. Ross, *The data warehouse toolkit: the complete guide to dimensional modeling*, John Wiley & Sons, 2013.

[36] W.H. Inmon, *Building the data warehouse*, John wiley & sons, 2005.

[37] S. Rizzi, A. Abelló, J. Lechtenbörger, and J. Trujillo, “Research in data warehouse modeling and design: dead or alive?,” *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, 2006, pp. 3–10.

[38] S. Hoberman, *Data modeling made simple: a practical guide for business and IT professionals*, Technics Publications, 2015.

[39] P. Ponniah, *Data warehousing fundamentals for IT professionals*, John Wiley & Sons, 2011.

[40] A. Zheng and A. Casari, *Feature engineering for machine learning: principles and techniques for data scientists*, O"Reilly Media, Inc., 2018.

[41] "Hibernate."

[42] R.C. Martin, J. Grenning, and S. Brown, *Clean architecture: a craftsman's guide to software structure and design*, Prentice Hall, 2018.

[43] R.C. Martin, *The clean coder: a code of conduct for professional programmers*, Pearson Education, 2011.

[44] M. Fowler, *Refactoring: improving the design of existing code*, Addison-Wesley Professional, 2018.

[45] G. Satpathy, "JXIRR."

[46] "The Apache Commons Mathematics Library."

[47] U. Fischbacher, G. Hoffmann, and S. Schudy, "The causal effect of stop-loss and take-gain orders on the disposition effect," *The Review of Financial Studies*, vol. 30, 2017, pp. 2110–2129.

[48] D.W. Richards, J. Rutterford, D. Kodwani, and M. Fenton-O'Creevy, "Stock market investors' use of stop losses and the disposition effect," *The European Journal of Finance*, vol. 23, 2017, pp. 130–152.

Appendix

I. Glossary

Disposition effect Tendency of investors to sell winning positions earlier and hold on to losing positions for longer.
Hazard ratio A model parameter estimate obtained from the Cox proportional hazard model
Hibernate An object–relational mapping tool for the Java
JPA Jakarta Persistence is an application programming interface specification that describes the management of relational data in Java
MWR Money-weighted investment return
POJO Plain Old Java Object is a simple Java class used for mapping data entities in the current context
RAP Risk adjusted performance
Stata Software for statistics and data science
TGI Trading gain indicator
TLI Trading loss indicator
TWR Time-weighted investment return

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Tõnn Talpsepp,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Data Analytics for the Estimating the Disposition Effect,
(title of thesis)

supervised by Rajesh Sharma.
(supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Tõnn Talpsepp

22/12/2021

III. Structure of the investor aggregate info dimension table

Attribute	Type	Default	Comment
id	int	None	Identifier of the investor
ipo_tallinna_vesi	tinyint(1)	0	Participated in IPO Tallinna Vesi
ipo_starman	tinyint(1)	0	Participated in IPO Starman
ipo_tallink	tinyint(1)	0	Participated in IPO Tallink
ipo_nordecon	tinyint(1)	0	Participated in IPO Nordecon
ipo_olympic	tinyint(1)	0	Participated in IPO Olympic
ipo_ekspress	tinyint(1)	0	Participated in IPO Ekspress Grupp
ipo_arco_vara	tinyint(1)	0	Participated in IPO Arco Vara
ipo_premia	tinyint(1)	0	Participated in IPO Premia
ipo_any	tinyint(1)	0	Participated in any IPO
r_all_withoutTC_twr	double	None	2004-2012 without transaction cost time weighted return
r_all_withoutTC_portf_beta	double	None	2004-2012 without transaction cost portfolio beta
r_all_with-outTC_std_portf_ret	double	None	2004-2012 without transaction cost std portfolio return
r_all_withoutTC_std_ex-cess_ret	double	None	2004-2012 without transaction cost std excess return
r_all_withoutTC_std_mar-ket_ret	double	None	2004-2012 without transaction cost std market return
r_all_withoutTC_market_ret	double	None	2004-2012 without transaction cost market return
r_all_withoutTC_hold-ing_per_d	int	None	2004-2012 without transaction cost holding period days
r_all_withTC_twr	double	None	2004-2012 with transaction cost time weighted return
r_all_withTC_portf_beta	double	None	2004-2012 with transaction cost portfolio beta
r_all_withTC_std_portf_ret	double	None	2004-2012 with transaction cost std portfolio return
r_all_withTC_std_excess_ret	double	None	2004-2012 with transaction cost std excess return
r_all_withTC_std_market_ret	double	None	2004-2012 with transaction cost std market return
r_all_withTC_market_ret	double	None	2004-2012 with transaction cost market return
r_all_withTC_holding_per_d	int	None	2004-2012 with transaction cost holding period days
r_2007_withoutTC_twr	double	None	2004-6.02.2007 without transaction cost time weighted return
r_2007_with-outTC_portf_beta	double	None	2004-6.02.2007 without transaction cost portfolio beta
r_2007_with-outTC_std_portf_ret	double	None	2004-6.02.2007 without transaction cost std portfolio return
r_2007_withoutTC_std_ex-cess_ret	double	None	2004-6.02.2007 without transaction cost std excess return
r_2007_withoutTC_std_mar-ket_ret	double	None	2004-6.02.2007 without transaction cost std market return
r_2007_withoutTC_mar-ket_ret	double	None	2004-6.02.2007 without transaction cost market return
r_2007_withoutTC_hold-ing_per_d	int	None	2004-6.02.2007 without transaction cost holding period days
r_2007_withTC_twr	double	None	2004-6.02.2007 with transaction cost time weighted return
r_2007_withTC_portf_beta	double	None	2004-6.02.2007 with transaction cost portfolio beta
r_2007_withTC_std_portf_ret	double	None	2004-6.02.2007 with transaction cost std portfolio return
r_2007_withTC_std_ex-cess_ret	double	None	2004-6.02.2007 with transaction cost std excess return
r_2007_withTC_std_mar-ket_ret	double	None	2004-6.02.2007 with transaction cost std market return
r_2007_withTC_market_ret	double	None	2004-6.02.2007 with transaction cost market return
r_2007_withTC_hold-ing_per_d	int	None	2004-6.02.2007 with transaction cost holding period days
r_2009_withoutTC_twr	double	None	7.02.2007-9.03.2009 without transaction cost time weighted re-turn
r_2009_with-outTC_portf_beta	double	None	7.02.2007-9.03.2009 without transaction cost portfolio beta

r_2009_with-outTC_std_portf_ret	double	None	7.02.2007-9.03.2009 without transaction cost std portfolio re-turn
r_2009_withoutTC_std_ex-cess_ret	double	None	7.02.2007-9.03.2009 without transaction cost std excess return
r_2009_withoutTC_std_mar-ket_ret	double	None	7.02.2007-9.03.2009 without transaction cost std market return
r_2009_withoutTC_mar-ket_ret	double	None	7.02.2007-9.03.2009 without transaction cost market return
r_2009_withoutTC_hold-ing_per_d	int	None	7.02.2007-9.03.2009 without transaction cost holding period days
r_2009_withTC_twr	double	None	7.02.2007-9.03.2009 with transaction cost time weighted return
r_2009_withTC_portf_beta	double	None	7.02.2007-9.03.2009 with transaction cost portfolio beta
r_2009_withTC_std_portf_ret	double	None	7.02.2007-9.03.2009 with transaction cost std portfolio return
r_2009_withTC_std_ex-cess_ret	double	None	7.02.2007-9.03.2009 with transaction cost std excess return
r_2009_withTC_std_mar-ket_ret	double	None	7.02.2007-9.03.2009 with transaction cost std market return
r_2009_withTC_market_ret	double	None	7.02.2007-9.03.2009 with transaction cost market return
r_2009_withTC_hold-ing_per_d	int	None	7.02.2007-9.03.2009 with transaction cost holding period days
r_2012_withoutTC_twr	double	None	10.03.2009-28.12.2012 without transaction cost time weighted return
r_2012_with-outTC_portf_beta	double	None	10.03.2009-28.12.2012 without transaction cost portfolio beta
r_2012_with-outTC_std_portf_ret	double	None	10.03.2009-28.12.2012 without transaction cost std portfolio re-turn
r_2012_withoutTC_std_ex-cess_ret	double	None	10.03.2009-28.12.2012 without transaction cost std excess re-turn
r_2012_withoutTC_std_mar-ket_ret	double	None	10.03.2009-28.12.2012 without transaction cost std market re-turn
r_2012_withoutTC_mar-ket_ret	double	None	10.03.2009-28.12.2012 without transaction cost market return
r_2012_withoutTC_hold-ing_per_d	int	None	10.03.2009-28.12.2012 without transaction cost holding period days
r_2012_withTC_twr	double	None	10.03.2009-28.12.2012 with transaction cost time weighted re-turn
r_2012_withTC_portf_beta	double	None	10.03.2009-28.12.2012 with transaction cost portfolio beta
r_2012_withTC_std_portf_ret	double	None	10.03.2009-28.12.2012 with transaction cost std portfolio return
r_2012_withTC_std_ex-cess_ret	double	None	10.03.2009-28.12.2012 with transaction cost std excess return
r_2012_withTC_std_mar-ket_ret	double	None	10.03.2009-28.12.2012 with transaction cost std market return
r_2012_withTC_market_ret	double	None	10.03.2009-28.12.2012 with transaction cost market return
r_2012_withTC_hold-ing_per_d	int	None	10.03.2009-28.12.2012 with transaction cost holding period days
MWR_all_no_transac-tion_cost	double	None	Money weighted return 2004-2012_no_transaction_cost
MWR_all_transaction_cost	double	None	Money weighted return 2004-2012_transaction_cost
MWR_2012_no_transac-tion_cost	double	None	Money weighted return 10.03.2009-28.12.2012_no_transac-tion_cost
MWR_2012_transaction_cost	double	None	Money weighted return 10.03.2009-28.12.2012_transac-tion_cost
MWR_2009_no_transac-tion_cost	double	None	Money weighted return 7.02.2007-9.03.2009_no_transac-tion_cost
MWR_2009_transaction_cost	double	None	Money weighted return 7.02.2007-9.03.2009_transaction_cost
MWR_2007_no_transac-tion_cost	double	None	Money weighted return 2004-6.02.2007_no_transaction_cost
MWR_2007_transaction_cost	double	None	Money weighted return 2004-6.02.2007_transaction_cost
number_of_sales_counted	int	None	Number of sales for which we know also buys (max 1 sale per stock per day)
number_of_buys_counted	int	None	Number_of_buys_counted (max 1 sale per stock per day)

portf_smallgain_ever	tinyint(1)	0	Has portfolio ever been in gain
portf_largegain_ever	tinyint(1)	0	Has portfolio ever been in more than 20% gain
portf_verylargegain_ever	tinyint(1)	0	Has portfolio ever been in more than 100% gain
portf_smallloss_ever	tinyint(1)	0	Has portfolio ever been in loss
portf_largeloss_ever	tinyint(1)	0	Has portfolio ever lost more than 20%
portf_verylargeloss_ever	tinyint(1)	0	Has portfolio ever lost more than 50%
fakesale_ever	tinyint(1)	0	Has any stocks been ever forcefully bought out from portfolio
active2004_2005	tinyint(1)	0	Investor active in 2004-2005 (at least 3 counted trades)
active2006_2007	tinyint(1)	0	Investor active in 2006-2007 (at least 3 counted trades)
active2008_2009	tinyint(1)	0	Investor active in 2008-2009 (at least 3 counted trades)
active2010_2012	tinyint(1)	0	Investor active in 20010-2012 (at least 3 counted trades)
total_stock_days_portf	int	None	(number of days stocks in portfolio * number of stocks in portfolio for each specific day). It can overestimate the number if sales were conducted in multiple parts (different days).
avg_portf_size	double	None	Average portfolio size in EUR (average size before each sale transaction)
avg_stocks_portf	int	None	How many different stocks in portfolio on average (average number before each sale transaction)
avg_holding_per_d	int	None	How many days each position was held in portfolio (counted by every sale transaction)
portf_turnover_rate	double	None	Totals_stock_days_portfolio/avg_holding_peridod_days. It should indicate how many times the stocks were changed during the period.
number_of_sale_transactions	int	None	Number_of_sale_transactions (each transaction is counted which can overestimate the number if one trade had multiple counterparties)
turnover_sales	double	None	The sum of all sale transactions EUR
avg_sum_of_sale_understating	double	None	Average sale transaction size EUR (turnover_sales/number_of_sale_transactions)
avg_sum_of_sale_overstating	double	None	Average sale transaction size EUR (turnover_sales/number_of_sale_transactions (max 1 per day))
number_of_buy_transactions	int	None	Number_of_buy_transactions (each transaction is counted which can overestimate the number if one trade had multiple counterparties)
turnover_buys	double	None	Turnover of buy transactions
avg_sum_of_buy_understated	double	None	Average buy transaction size EUR (turnover_buys/number_of_buy_transactions)
avg_sum_of_buy_overstated	double	None	Average buy transaction size EUR (turnover_buys/number_of_buy_transactions (max 1 per day))
number_of_net_sales	int	None	Number_of_net_sales (number of sales where same stock transactions are netted for the day (meaning max 1 per day))
turnover_net_sales	double	None	The sum of net sale transactions EUR
avg_sum_of_net_sale	double	None	Average size of net sale transaction EUR
number_of_net_buys	int	None	Number_of_net_buys (number of buys where same stock transactions are netted for the day (meaning max 1 per day))
turnover_net_buys	double	None	The sum of net buy transactions EUR
avg_sum_of_net_buy	double	None	Average size of net buy transaction EUR
total_num_transactions	int	None	Number of net sales plus number of net buys
avg_net_transaction_size	double	None	Avg_net_transaction_size EUR
stock_hansa	tinyint(1)	0	Has ever owned stock_hansa
stock_kaubamaja	tinyint(1)	0	Has ever owned stock_kaubamaja
stock_saku	tinyint(1)	0	Has ever owned stock_saku
stock_estiko	tinyint(1)	0	Has ever owned stock_estiko
stock_farmaatsia	tinyint(1)	0	Has ever owned stock_farmaatsia
stock_silvano	tinyint(1)	0	Has ever owned stock_silvano
stock_norma	tinyint(1)	0	Has ever owned stock_norma
stock_kalev	tinyint(1)	0	Has ever owned stock_kalev
stock_rakvere	tinyint(1)	0	Has ever owned stock_rakvere
stock_trigon	tinyint(1)	0	Has ever owned stock_trigon
stock_merko	tinyint(1)	0	Has ever owned stock_merko
stock_baltika	tinyint(1)	0	Has ever owned stock_baltika

stock_harju_elekter	tinyint(1)	0	Has ever owned stock_harju_elekter
stock_tallink	tinyint(1)	0	Has ever owned stock_tallink
stock_telekom	tinyint(1)	0	Has ever owned stock_telekom
stock_starman	tinyint(1)	0	Has ever owned stock_starman
stock_ekspress	tinyint(1)	0	Has ever owned stock_ekspress
stock_tallinna_vesi	tinyint(1)	0	Has ever owned stock_tallinna_vesi
stock_arco_vara	tinyint(1)	0	Has ever owned stock_arco_vara
stock_nordecon	tinyint(1)	0	Has ever owned stock_nordecon
stock_olympic	tinyint(1)	0	Has ever owned stock_olympic
stock_viisnurk	tinyint(1)	0	Has ever owned stock_viisnurk
stock_premia_foods	tinyint(1)	0	Has ever owned stock_premia_foods
tr_2003_Q4	int	None	Number of trades in 2003_Q4
tr_2004_Q1	int	None	Number of trades in 2004_Q1
tr_2004_Q2	int	None	Number of trades in 2004_Q2
tr_2004_Q3	int	None	Number of trades in 2004_Q3
tr_2004_Q4	int	None	Number of trades in 2004_Q4
tr_2005_Q1	int	None	Number of trades in 2005_Q1
tr_2005_Q2	int	None	Number of trades in 2005_Q2
tr_2005_Q3	int	None	Number of trades in 2005_Q3
tr_2005_Q4	int	None	Number of trades in 2005_Q4
tr_2006_Q1	int	None	Number of trades in 2006_Q1
tr_2006_Q2	int	None	Number of trades in 2006_Q2
tr_2006_Q3	int	None	Number of trades in 2006_Q3
tr_2006_Q4	int	None	Number of trades in 2006_Q4
tr_2007_Q1	int	None	Number of trades in 2007_Q1
tr_2007_Q2	int	None	Number of trades in 2007_Q2
tr_2007_Q3	int	None	Number of trades in 2007_Q3
tr_2007_Q4	int	None	Number of trades in 2007_Q4
tr_2008_Q1	int	None	Number of trades in 2008_Q1
tr_2008_Q2	int	None	Number of trades in 2008_Q2
tr_2008_Q3	int	None	Number of trades in 2008_Q3
tr_2008_Q4	int	None	Number of trades in 2008_Q4
tr_2009_Q1	int	None	Number of trades in 2009_Q1
tr_2009_Q2	int	None	Number of trades in 2009_Q2
tr_2009_Q3	int	None	Number of trades in 2009_Q3
tr_2009_Q4	int	None	Number of trades in 2009_Q4
tr_2010_Q1	int	None	Number of trades in 2010_Q1
tr_2010_Q2	int	None	Number of trades in 2010_Q2
tr_2010_Q3	int	None	Number of trades in 2010_Q3
tr_2010_Q4	int	None	Number of trades in 2010_Q4
tr_2011_Q1	int	None	Number of trades in 2011_Q1
tr_2011_Q2	int	None	Number of trades in 2011_Q2
tr_2011_Q3	int	None	Number of trades in 2011_Q3
tr_2011_Q4	int	None	Number of trades in 2011_Q4
tr_2012_Q1	int	None	Number of trades in 2012_Q1
tr_2012_Q2	int	None	Number of trades in 2012_Q2
tr_2012_Q3	int	None	Number of trades in 2012_Q3
tr_2012_Q4	int	None	Number of trades in 2012_Q4
pure_transactions	int	None	Number of trades for whole period
first_trade_date	date	None	Date of the first trade made
r_all_without_RAP	double	None	RAP model 2004-2012 without transaction cost
r_all_with_RAP	double	None	RAP model 2004-2012 with transaction cost
r_2007_without_RAP	double	None	RAP model 2004-6.02.2007 without transaction cost
r_2007_with_RAP	double	None	RAP model 2004-6.02.2007 with transaction cost
r_2009_without_RAP	double	None	RAP model 7.02.2007-9.03.2009 without transaction cost
r_2009_with_RAP	double	None	RAP model 7.02.2007-9.03.2009 with transaction cost
r_2012_without_RAP	double	None	RAP model 10.03.2009-28.12.2012 without transaction cost
r_2012_with_RAP	double	None	RAP model 10.03.2009-28.12.2012with transaction cost
r_all_without_Beta_model	double	None	Beta model 2004-2012 without transaction cost
r_all_with_Beta_model	double	None	Beta model 2004-2012 with transaction cost

r_2007_without_Beta_model	double	None	Beta model 2004-6.02.2007 without transaction cost
r_2007_with_Beta_model	double	None	Beta model 2004-6.02.2007 with transaction cost
r_2009_without_Beta_model	double	None	Beta model 7.02.2007-9.03.2009 without transaction cost
r_2009_with_Beta_model	double	None	Beta model 7.02.2007-9.03.2009 with transaction cost
r_2012_without_Beta_model	double	None	Beta model 10.03.2009-28.12.2012 without transaction cost
r_2012_with_Beta_model	double	None	Beta model 10.03.2009-28.12.2012with transaction cost
r_all_without_Sharpe	double	None	Sharpe ratio 2004-2012 without transaction cost
r_all_with_Sharpe	double	None	Sharpe ratio 2004-2012 with transaction cost
r_2007_without_Sharpe	double	None	Sharpe ratio 2004-6.02.2007 without transaction cost
r_2007_with_Sharpe	double	None	Sharpe ratio 2004-6.02.2007 with transaction cost
r_2009_without_Sharpe	double	None	Sharpe ratio 7.02.2007-9.03.2009 without transaction cost
r_2009_with_Sharpe	double	None	Sharpe ratio 7.02.2007-9.03.2009 with transaction cost
r_2012_without_Sharpe	double	None	Sharpe ratio 10.03.2009-28.12.2012 without transaction cost
r_2012_with_Sharpe	double	None	Sharpe ratio 10.03.2009-28.12.2012with transaction cost

IV. Developed Java code for running data generation and calculation procedures

The code can be viewed on:

https://gitfront.io/r/thesis-se2022/b23ce3169de31e3ca4b0adbd0c14344863854f07/master-thesis-public/