FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Implementierung der Konservativen Formel nach David Blitz und Pim van Vliet

Eine Portfoliomanagement Strategie

BACHELORARBEIT

von

Andreas Michael Buchner
Matrikelnummer 11811860

Betreuung: Dipl.-Ing. Mag. Dr. Thomas Neubauer

Wien, 31.05.2021

Erklärung zur Verfassung der Arbeit

Andreas Michael Buchner
Glaubackerstraße 4, 4040 Linz

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

Herzlichen Dank an meinen Betreuer Dipl.-Ing. Mag. Dr. Thomas Neubauer, der mir bei der Verfassung und Themenwahl dieser Arbeit viele Freiheiten gelassen hat und mich tatkräftig unterstützt hat.

Weiters möchte ich mich bei meinen Eltern und meinem Bruder für ihre generelle Unterstützung und anregenden Diskussionen bedanken.

Ausnahmsweise darf ich mich auch ein wenig für die Corona-Krise bedanken, durch die mir reichlich Zeit gegeben wurde, mich mit diesem Thema zu beschäftigen und das Programm zu entwickeln.

Schlussendlich möchte ich mich auch bei den Entwicklern der Konservativen Formel bedanken, die den wirtschaftlichen und strategischen Grundstein für diese Arbeit gelegt haben.

Abstract

Today's stock markets are characterized by an unusual high volatility. This volatility has brought many new traders into the stock market. To trade successfully in the stock market for a longer time span a clearly defined strategy is essential.

The aim of this work is to present an existing strategy and to implement a program that trades stocks automatically based on this strategy.

To achieve this goal first the strategy, the so-called Conservative Formula, and its used indicators are being presented. Following on from this a program is being implemented, that deploys the strategy. The implementation and the used external libraries are being explained in detail. Open issues that appeared during implementing, which can be solved in further academic work, are being listed.

The finished program will then be tested in real time with virtual money and the results of this testing phase are being presented. On average the results show that the Conservative Formula has an edge on today's stock market. At least in the tested timeframe from 11.02.2021 to 31.5.2021 the implementation significantly outperformed the most common Benchmark ETF with the symbol „SPY“, that supposedly tracks the S&P 500 Index.

Kurzfassung

Aktienmärkte sind in der heutigen Zeit von einer ungewöhnlich hohen Volatilität geprägt. Diese Volatilität hat das Interesse vieler neuer Anleger geweckt. Um in den Aktienmärkten über längere Zeit erfolgreich zu sein, ist eine klar definierte Strategie unerlässlich.

Ziel der folgenden Arbeit ist, eine existierende Strategie vorzustellen und in Folge diese Strategie mittels einem Programm am heutigen Aktienmarkt zu automatisieren.

Um dieses Ziel zu erreichen wird zuerst die zugrundeliegende Strategie, die sogenannte Konservative Formel, und die darin verwendeten Indikatoren vorgestellt. Darauf basierend wird ein Programm entwickelt, das die vorgestellte Strategie umsetzt. Die Implementierung dieses Programms sowie die dafür benötigten externen Bibliotheken werden detailliert erklärt. Auch Probleme, die im Laufe der Implementierung auftraten und möglicherweise durch weiterführender Arbeiten gelöst werden können, werden aufgelistet.

Das erstellte Programm wird dann in Echtzeit mit virtuellem Geld getestet und die Ergebnisse dieser Testphase präsentiert. Im Schnitt zeigt sich dabei, dass die Konservative Formel am heutigen Aktienmarkt zumindest im betrachteten Zeitraum vom 11.02.2021 bis zum 31.5.2021 immer noch seine Gültigkeit besitzt und sich deutlich besser entwickelt, als der meist verwendete amerikanische Benchmark ETF mit dem Kürzel „SPY “, der den S&P 500 Index abzubilden versucht.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Forschungsfragen . . . 1
- 1.3 Methoden . . . 2
- 1.4 Aufbau der Arbeit . . . 2

2 Konservative Formel **3**
- 2.1 Motivation der Konservativen Formel . . . 3
- 2.2 Verwendete Indikatoren . . . 3
- 2.3 Interpretation der Indikatoren . . . 5
- 2.4 Die Strategie . . . 6
- 2.5 Ergebnisse der Konservativen Formel . . . 7

3 Programmarchitektur und Design **9**
- 3.1 UML Diagramme . . . 9
- 3.2 Datenstruktur . . . 11

4 Implementierung **14**
- 4.1 Python . . . 14
- 4.2 API . . . 14
- 4.3 Verwendete Externe Bibliotheken . . . 15
- 4.4 Persistente Datenspeicherung . . . 18
- 4.5 Alerter . . . 19
- 4.6 Bewertungsalgorithmus . . . 19
- 4.7 Portfolio-Konstruktion . . . 23
- 4.8 Algorithmus zur Neuausrichtung des Portfolios . . . 26
- 4.9 Neuausrichtung jedes Quartal . . . 28
- 4.10 Umgang mit Errors . . . 29

5 Validierung und Diskussion **32**
- 5.1 Resultate . . . 32
- 5.2 Probleme bei der Umsetzung . . . 33

6 Zusammenfassung **35**
- 6.1 Optimierungsvorschläge . . . 35
- 6.2 Ausblick . . . 35

Literaturverzeichnis **37**

Kapitel 1

Einleitung

1.1 Motivation

In Zeiten einer globalen Pandemie, in der die Aktienmärkte eine überdurchschnittlich hohe Volatilität aufweisen [1], ist das Investieren riskanter geworden. Volatilität ist eine bekannte Kennzahl, die die Stärke der Kursschwankungen eines Aktienkurses beschreibt. Diese hohe Volatilität, welche die Kurse innerhalb kürzester Zeiträume in neue Höhen, aber auch Tiefen treibt, hat das Interesse vieler neuen Investoren geweckt [12]. Vor allem jüngere Generationen, die die Aktienmärkte erkunden wollen und auf der Suche nach dem schnellen Geld sind, werden in den Bann gezogen. Dazu als Vergleich der Anstieg der Nutzerzahlen des Subreddits „r/wallstreetbets“, ein Kanal der sozialen Plattform „Reddit“, der seit Beginn der Corona-Pandemie im Jahr 2020 von rund 800.000 auf knappe zehn Millionen Nutzer im April 2021 gewachsen ist [34]. Dieser Kanal dient, wie der Name bereits vermuten lässt, hauptsächlich dazu, Diskussionen über hochspekulativen Handel von Aktien und Optionen zu führen [33].

Kurzfristigen Spekulationen, vor allem den Handel mit Optionen betreffend, ist ein hohes Risiko charakteristisch, welche im schlimmsten Fall zum Totalverlust führen können [14]. Häufig werden Spekulationen auf Basis nicht begründbarer Gefühle getätigt. Um aber über längere Zeit am Kapitalmarkt erfolgreich zu sein, ist eine klar definierte Strategie unerlässlich, denn Glück und Gefühl sind auf lange Sicht nicht die besten Berater. Es gibt unzählig viele Investment-Strategien und jeder kann auch eine eigene Strategie entwickeln. Wichtig ist, dass die gewählte Strategie eine Entscheidungsgrundlage dafür ist, warum welcher Handel getätigt wird. So wird der Faktor des Glücks und des Gefühls minimiert und aufgrund objektiver Signale gehandelt. Ein weiterer Vorteil einer klar definierten Strategie liegt darin, dass die Effektivität quantifiziert werden kann. Dadurch ist eine Vergleichbarkeit der Strategien untereinander möglich bzw. die objektiv bessere Strategie kann evaluiert werden [8].

Diese Arbeit wird eine Strategie vorstellen und implementieren, die den Anspruch stellt, über eine längere Zeit eine bessere Performance als der zugrundeliegende Markt zu erreichen [2]. Diese Strategie wird von ihren Entwicklern David Blitz und Pim van Vliet die „Konservative Formel“genannt.

1.2 Forschungsfragen

Die Forschungsfragen dieser Arbeit beschränken sich im Grunde auf folgende:

1. Was ist die Konservative Formel und welche Ziele verfolgt sie?

   Dabei wird untersucht, wie die Strategie der Konservativen Formel aussieht und weshalb diese Strategie besser sein soll, als der zugrundeliegende Markt. Es werden die verwendeten Indikatoren einzeln vorgestellt und erklärt.

2. Wie kann die Konservative Formel am heutigen Aktienmarkt umgesetzt werden?

Diese Frage ist technischer Natur und stellt das eigentliche Ziel des praktischen Teils dieser Arbeit dar: die Implementierung der Konservativen Formel und das Testen dieser am Aktienmarkt in Echtzeit.

3. Wie gut funktioniert die Konservative Formel am heutigen Aktienmarkt?

   Die Ergebnisse sollen zeigen, ob die Konservative Formel am heutigen Aktienmarkt den gewünschten Erfolg bringt.

1.3 Methoden

Dieses Kapitel beschreibt, welche Methoden und Vorgehensweisen zur Verfassung dieser Arbeit verwendet werden.

Literaturrecherche

Als erste Methode wird die Literaturrecherche angewendet. Dabei ist die Konservative Formel entdeckt worden und darauf basieren auch die theoretischen Aspekte dieser Arbeit, die zum Verständnis der Konservativen Formel und dessen Indikatoren nötig sind. Im Zuge dieser Methode wird versucht, einen groben Überblick über Teilaspekte des Aktienmarkts zu geben und auf weiterführende Literatur zu verweisen.

Implementierung

Die Implementierung stellt den größten Teil dieser Arbeit dar und umfasst die Umsetzung der Konservativen Formel in ein fertiges Programm, welches autonom danach handelt. Dazu gehören auch die Planung und das Design.

1.4 Aufbau der Arbeit

Der Aufbau gliedert sich nach der Einleitung in folgende Kapitel: Zuerst werden die Konservative Formel, deren Ziele und zugrundeliegenden Berechnungsmethoden der verwendeten Indikatoren vorgestellt. Die historischen Ergebnisse der Konservativen Formel werden erläutert und diskutiert.

Im nächsten Kapitel werden das geplante Design und die Architektur des Programms beschrieben, bevor schließlich der Hauptteil, die Implementierung selbst, folgt. Nach dem Hauptteil werden in der Validierung und Diskussion die Resultate der implementierten Strategie anhand der Performance des erstellten Portfolios der Entwicklung des bekanntesten Benchmark-ETF, mit dem Kürzel „SPY “, der den S&P 500 Index abbilden soll, gegenübergestellt und erläutert, wo es bei der Implementierung Schwierigkeiten gab. Am Schluss dieser Arbeit befindet sich eine kurze Zusammenfassung, in der auch Optimierungsvorschläge angeführt werden sowie ein genereller Ausblick gemacht wird.

Kapitel 2

Konservative Formel

2.1 Motivation der Konservativen Formel

Das folgende Kapitel fasst die Motivation hinter der Entwicklung Konservativen Formel [2] zusammen.

Die Konservative Formel erschien im Jahr 2018 unter anderem in der Zeitschrift: „The Journal of Portfolio Management“. Sie wurde von David Blitz und Pim van Vliet entwickelt.

Bei der Konservativen Formel handelt es sich um eine Portfoliomanagement-Strategie, die darauf abzielt, das Risiko zu minimieren. Laut dem in den 1960er Jahren entwickelten Capital Pricing Model, zu Deutsch Kapitalgutpreismodell, ist die empirische Relation von Risiko und Erfolg eine flache Gerade [23]. Das bedeutet, dass ein erhöhtes Risiko im Schnitt zu einem unterproportional großem Erfolg führt. Somit führt das doppelte Risiko erfahrungsgemäß nicht zu einem doppelt so großen Gewinn. Diese Eigenschaft wird „low-risk effect“genannt, wodurch sich, historisch gesehen, Portfolios mit geringem Risiko im Durchschnitt besser entwickelten als solche, mit höherem Risiko [3].

Ein weiterer wichtiger Aspekt der Konservativen Formel ist, dass diese relativ einfach umzusetzen ist. Viele andere wissenschaftlich entwickelte Strategien sehen zwar auf dem Papier gut aus, verlieren in der Realität aber ihre Gültigkeit. Entweder wird bei diesen Strategien durch Steuern oder Gebühren, die bei jedem Kauf bzw. Verkauf zu zahlen sind, der Gewinn geschmälert, oder es wird von schlichtweg unrealistischen Marktbedingungen, wie z.B. unendlich großes Angebot oder Nachfrage, ausgegangen. Die Autoren verweisen in ihrer Kritik an anderen Strategien auf einen Artikel namens „Replicating Anomalies“ [18], der 2018 in „The Review of Financial Studies“ erschienen ist.

Am Aktienmarkt können nicht nur einfache Aktien gehandelt werden. Es gibt unter anderem auch die Möglichkeit mit Optionen zu handeln. Eine Option bedeutet im Grunde, dass sich der Anleger vertraglich zusichert, zu einem späteren Zeitpunkt zu einem gewissen Kurs zu kaufen bzw. zu verkaufen. Dabei wird entweder auf steigende oder auf fallende Kurse gesetzt. In vielen auch akademisch entwickelten Strategien wird laut [18] von einem unendlichen Angebot an Optionsscheinen ausgegangen, was der Realität aber keinesfalls entspricht.

Die Konservative Formel setzt auf eine „Long-only“ Strategie. Das bedeutet, dass keine Optionen gehandelt werden und Aktien auf herkömmliche Art gekauft und zu einem späteren Zeitpunkt, sofern gewisse strategische Rahmenbedingungen zutreffen, wieder verkauft werden.

2.2 Verwendete Indikatoren

Bevor die Konservative Formel im Detail weiter erläutert werden kann, müssen einige Indikatoren, die darin verwendet werden, erklärt werden:

Marktkapitalisierung [4]

Die Marktkapitalisierung beschreibt den Wert, den ein Unternehmen aktuell auf dem Aktienmarkt hat (Börsenwert). Dieser berechnet sich aus dem Aktienpreis x in der jeweiligen Währung, multipliziert mit

der absoluten Anzahl an Aktien y. Somit ergibt sich die Marktkapitalisierung M aus $M = x * y$.

Hat also ein Unternehmen z.B. 1000 Aktien am Markt, die aktuell um 5 $ pro Stück gehandelt werden, ergibt die Berechnung der Marktkapitalisierung einen Börsenwert von 5000 $.

Volatilität [6]

Unter Volatilität versteht man die Schwankungsbreite eines Aktienkurses. Berechnet wird sie mithilfe der Standardabweichung der Kursveränderungen.

In der Konservativen Formel wird die Volatilität basierend auf den letzten 36 Monaten der täglichen Kursänderung berechnet. Je kleiner die resultierende Volatilität, umso stabiler war der Kursverlauf der Aktie im betrachteten Zeitraum.

Beispiele der Berechnung einer 3-Tage Volatilität fiktiver Aktienkurse sind in Tabelle 2.1 dargestellt:

Aktie	Tag 1	Tag 2	Veränderung 2:1	Tag 3	Veränderung 3:2	Volatilität
Aktie 1	100 $	120 $	1.2	144 $	1.2	$\sigma([1.2, 1.2]) = 0$
Aktie 2	100 $	110 $	1.1	99 $	0.9	$\sigma([1.1, 0.9]) = 0.1$
Aktie 3	100 $	100 $	1	200 $	2	$\sigma([1, 2]) = 0.5$
Aktie 4	100 $	500 $	5	50 $	0.1	$\sigma([5, 0.1]) = 2.45$

Tabelle 2.1: Beispiel für Berechnung der Volatilität

Momentum [5]

Als Momentum bezeichnet man die Veränderungsrate einer Aktie über einen gewissen Zeitraum. Bei dem Beispiel zur Berechnung der Volatilität in Tabelle 2.1 wurde bereits das Momentum über einzelne Tage berechnet. Die Formel zur Berechnung des Momentums im Zeitraum eines Jahres lautet kurz $M = x/y$, wobei M das Momentum, x der momentane Aktienkurs und y der Aktienkurs von vor einem Jahr ist.

Ein Kursverlauf von 100 $ am 1. Januar 2020 auf 120 $ am 1. Januar 2021 resultiert nach dieser Formel also beispielsweise in einem Momentum von 1.2.

Nettoauszahlungsrendite [36]

Die Nettoauszahlungsrendite, englisch Net Payout Yield oder auch Total Yield, ist die Summe der Rückkaufrendite und der Dividendenrendite, welche in den folgenden Absätzen erklärt werden.

Rückkaufrendite [13]

Die Rückkaufrendite sagt aus, wie viele eigene Aktien von einer Aktiengesellschaft zurückgekauft werden. Die Formel zur Berechnung der Rückkaufrendite gestaltet sich folgendermaßen: $R = x * k/y * 100$, wobei R die resultierende Rückkaufrendite in % ist, x die Anzahl an zurückgekauften Aktien, k der Kurs zum Zeitpunkt des Rückkaufes in der jeweiligen Währung, und y die Marktkapitalisierung darstellt. Es werden dabei nur Rückkäufe betrachtet, die in einem definierten Zeitraum, z.B. zwei Jahre, liegen.

Somit hat ein Unternehmen, das vor zwei Jahren 100 Anteile am Markt hatte, innerhalb dieser zwei Jahre 10 Aktien zu einem Preis von je 10 $ von Anlegern zurückgekauft hat, 100 $ an eigenen Aktien zurückgekauft. Die Rückkaufrendite berechnet sich nun mithilfe der Marktkapitalisierung des Unternehmens, nehmen wir dafür 1000 $ an. Somit ergibt sich eine Rückkaufrendite von 10%. In die Formel eingesetzt sieht das so aus: $(10Stk. * 10\$)/1000\$ * 100 = 10\%$.

Die Rückkaufrendite kann auch negativ sein. In diesem Fall hat die Aktiengesellschaft neue Anteile auf dem Markt zum Verkauf angeboten. Gründe dafür können vielseitig sein, meist wird neues Geld von Investoren benötigt.

Dividendenrendite [20]

Die Dividendenrendite beschreibt, wie viel an Dividende ein Unternehmen seinen Anleger:innen in Relation zum Aktienkurs ausgezahlt hat. Dividenden sind Gewinnausschüttungen, die direkt an die Beteiligten der Aktiengesellschaft, den Aktionär:innen, gehen. Berechnet wird die Dividendenrendite durch den Anteil, der innerhalb des letzten Jahres an die Anleger:innen ausgeschüttet wurde zum aktuellen Preis der Aktie. Die Formel lautet also $D = x/y * 100$, wobei D die Dividendenrendite in % ist, x die Summe des ausgeschütteten Gewinns innerhalb des letzten Jahres pro Aktie und y den aktuellen Kurs der Aktie darstellt.

Eine Aktiengesellschaft, die im letzten Jahr 1 $ pro Aktie an Dividende ausgezahlt hat und deren Kurs momentan bei 10 $ liegt, hat somit eine Dividendenrendite von 10%.

Die Dividendenrendite kann nur positive Zahlen annehmen, weil es negative Dividendenauszahlungen nicht gibt. Im geringsten Fall ist die Dividendenrendite also gleich 0, was bedeutet, dass im beobachteten Zeitraum keine Dividende ausgezahlt wurde.

2.3 Interpretation der Indikatoren

Die vorgestellten Indikatoren beschreiben unterschiedliche Eigenschaften eines Aktienkurses bzw. einer Aktiengesellschaft. Wie diese in der Konservativen Formel interpretiert werden, wird in den folgenden Absätzen geklärt:

Marktkapitalisierung

Marktkapitalisierung drückt den Börsenwert einer Aktiengesellschaft aus. Ein großer Börsenwert lässt auf ein großes Unternehmen schließen, was tendenziell eine sicherere Investition darstellt.

In der Konservativen Formel werden bei der Erstellung des Portfolios ausschließlich Aktien aus den nach Marktkapitalisierung größten 1000 berücksichtigt.

Volatilität

Volatilität wird primär als Indikator für das Risiko einer Aktie verwendet. Hohe Volatilität steht für große Kursschwankungen und damit verbunden eine größere Unsicherheit und somit auch ein größeres Risiko.

Es werden somit in der Konservativen Formel nur Aktien mit geringer Volatilität im Verlauf der letzten drei Jahren berücksichtigt. Auch wenn historische Volatilität nicht zwangsweise ein Indikator für die zukünftige Volatilität sein muss, zeigen die Ergebnisse von Blitz und Van Vliet [2], siehe dazu auch Kapitel 2.5, dass historische Volatilität eine gewisse Aussagekraft über zukünftige Volatilität besitzt.

Momentum

In der Konservativen Formel wird das 12 - 1 Monate Momentum verwendet, das heißt, der aktuelle Kurs wird mit dem Kurs von vor einem Jahr verglichen. Hier gilt in der resultierenden Strategie, dass ein größeres Momentum besser ist, weil dadurch der Aktienkurs große Gewinne verbucht hat und eine Fortsetzung dieses Trends erwartet wird.

Rückkaufrendite [35]

In der Konservativen Formel wird bei der Berechnung der Rückkaufrendite der Zeitrahmen der letzten zwei Jahre berücksichtigt.

Laut der Konservativen Formel ist eine hohe Rückkaufrendite gut, ob aber jeder Aktienrückkauf ausschließlich positive Folgen hat, ist, wie z.B. der Artikel [35] zeigt, umstritten. Oft werden durch größere Rückkäufe beispielsweise die Kurse in die Höhe getrieben, was das Unternehmen überdurchschnittlich viel für seine eigenen Anteile bezahlen lässt. Es kann aber auch sein, dass die Aktiengesellschaft ihre Anteile über Kredite zurückkauft, um einer Übernahme entgegenzuwirken und die Folgen erst bei Fälligkeit dieser Kredite ersichtlich werden. Eine überdurchschnittlich hohe Rückkaufrendite kann zudem

auch negativ interpretiert werden, wenn die Aktiengesellschaft keine Investitionsmöglichkeiten in ihrem eigentlichen Kerngeschäft sieht, daher viele eigene Aktien zurückkauft und in Folge der fehlenden Investitionen in Zukunft weniger Wachstum aufweisen wird als vergleichbare Unternehmen mit einer geringeren Rückkaufrendite.

Als positive Aspekte einer hohen Rückkaufrendite sind anzuführen, dass Aktienrückkäufe normalerweise den Kurs stabilisieren und die zukünftigen Gewinne auf weniger Aktionäre verteilt werden müssen, was zu einer besseren Dividendenrendite führen kann.

Dividendenrendite

Eine hohe Dividendenrendite bedeutet eine hohe Ausschüttung an Dividende an die Anleger:innen. Somit wird in der Konservativen Formel eine hohe Dividendenrendite positiv interpretiert.

Nettoauszahlungsrendite

Die Nettoauszahlungsrendite wird, weil Dividendenrendite und Rückkaufrendite positiv interpretiert werden, ebenfalls positiv interpretiert. Somit ist es in der Konservativen Formel wahrscheinlich, dass Aktiengesellschaften mit hoher Nettoauszahlungsrendite im Portfolio landen.

2.4 Die Strategie

Hier kann mit der Erklärung der Konservativen Formel fortgesetzt und, da nun die benötigten Indikatoren sowie deren Interpretation vorgestellt wurden, die eigentliche Strategie erklärt werden.

Bei der Konservativen Formel handelt es sich um eine Portfoliomanagement Strategie. Das bedeutet, dass in ein Portfolio bestehend aus mehreren Aktien investiert wird. Der Sinn eines Portfolios gegenüber Einzelaktien liegt darin, das Risiko zu streuen und somit weniger abhängig gegenüber einzelnen Kursschwankungen zu sein. Mehr Informationen über Streuung, auch Diversifikation genannt, findet man beispielsweise in „Equity Portfolio Diversification“, erschienen im Journal „Review of Finance“ [15].

Die Konservative Formel setzt auf ein Portfolio von insgesamt 100 verschiedenen Aktien. In diese 100 Aktien wird der zu investierende Betrag gleichmäßig gewichtet aufgeteilt und investiert. Um die 100 gewünschten Aktien zu eruieren wird nun folgendes Prozedere durchgeführt:

1. Das gesamte Universum von bekannten Aktien, die länger als drei Jahre an der Börse sind, wird betrachtet.

2. Die Marktkapitalisierung dieser bekannten Aktien wird berechnet und das Universum wird auf jene 1000, mit der größten Marktkapitalisierung, reduziert.

3. Die Volatilität der letzten drei Jahre von jeder der 1000 Aktien wird berechnet. Die Resultate werden aufsteigend sortiert und es werden von nun an bloß die 500 Aktien, mit geringer Volatilität betrachtet.

4. Das 12 - 1 Monate Momentum jeder einzelnen dieser 500 Aktien wird berechnet. Nach dieser Berechnung wird absteigend sortiert und jeder Aktie wird der Rang nach dieser Sortierung zugewiesen. Die Aktie mit dem größten Momentum bekommt den Rang 1, die Aktie mit dem geringsten Momentum den Rang 500.

5. Die Nettoauszahlungsrendite jeder einzelnen Aktie aus dem Universum von 500 Stück wird berechnet. Nun wird dasselbe Bewertungsschema wie bei dem Momentum angewendet: Absteigend sortieren und einen Rang zuweisen.

6. Es resultiert eine Liste von 500 Aktien, denen jeweils zwei verschiedene Bewertungs-Platzierungen zugewiesen wurden. Die finale Bewertung wird aus dem Durchschnitt dieser Platzierungen berechnet. Eine Aktie, die in der Reihung nach Momentum auf Rang 13 liegt und bei der Reihung nach Nettoauszahlungsrendite auf Rang 15 liegt, bekommt als finale Bewertung eine 14 zugewiesen.

Diese 14 bedeutet aber nicht, dass diese Aktie am Ende auch den 14. Rang belegt, es ist lediglich die finale Bewertung.

7. Nach dieser finalen Bewertung werden die 500 Aktien aufsteigend sortiert. Als gewünschtes Portfolio werden nun die besten 100 Aktien, mit der niedrigsten finalen Bewertung, ausgewählt.

Jetzt muss in diese 100 resultierende Aktien noch annähernd gleich verteilt investiert werden. Dieser Bewertungsprozess wird jedes Quartal, also alle drei Monate, wiederholt und das bestehende Portfolio auf das neue Ergebnis angepasst. [2]

Wir sehen also, dass von keinerlei unrealistischen Marktbedingungen wie unbegrenzte Verfügbarkeit von Optionsscheinen oder ähnlichem ausgegangen wird. Somit haben die Entwickler dieser Strategie mit ihrer Behauptung, eine realistisch umsetzbare Strategie entwickelt zu haben, recht.

2.5 Ergebnisse der Konservativen Formel

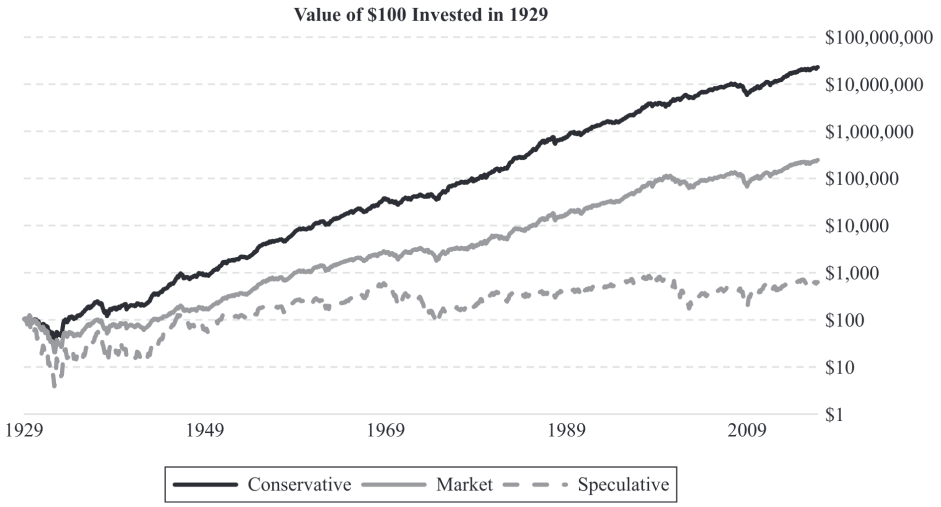

Abbildung 2.1: Entwicklung von 1929 bis 2016 [2]

Die Ergebnisse der Konservativen Formel sehen folgendermaßen aus: der amerikanische Aktienmarkt stieg im Zeitraum von 1929 bis einschließlich 2016 im Durchschnitt um 10,6% pro Jahr. Wenn in dieser Periode durchgehend gleich verteilt in den Markt investiert worden wäre, hätte der Markt unter Berücksichtigung des Zinseszins-Effekts eine Steigerung von rund 9,3% pro Jahr erfahren. Eine Investition von 100 $ im Jahre 1929 hätte, laut den Autoren der Konservativen Formel [2], inklusive Zinseszinsen im Jahre 2016 rund 246,000 $ ergeben.

Das Portfolio der Konservative Formel hingegen erbrachte im gleichen Zeitraum einen durchschnittlichen Erfolg von 15,5% pro Jahr. Über den gesamten Zeitraum, abermals unter Berücksichtigung des Zinseszins-Effekts, betrug der Erfolg 15,1% pro Jahr. Das ist deutlich näher am durchschnittlichen Jahreserfolg, was bedeutet, dass die Erfolge über die Jahre weniger Schwankung aufwiesen als der Markt. Um es wieder durch die 100 $ Investition im Jahre 1929 bis 2016 auszudrücken: aus 100 $ die nach dieser Strategie 1929 angelegt worden wären, wären 2016 rund 20 Mio. $ geworden.

Diese Ergebnisse wurden allerdings ohne Steuern und Gebühren berechnet, somit wäre in der Realität der Zinseszins-Effekt viel abgeschwächter. Selbiges gilt auch für das Beispiel des Markts.

In der Arbeit von Blitz und Van Vliet [2] wurde ebenfalls mit einem sogenannten „spekulativem“ Portfolio verglichen. Dabei wurde die gleiche Berechnung wie bei der konservativen Formel durchgeführt, mit dem Unterschied, dass die volatilere Hälfte betrachtet wurde und von diesen in die *schlechtesten* anhand der Reihung nach Momentum und Nettoauszahlungsrendite investiert wurde. Dieses spekulative Portfolio entwickelte sich deutlich schlechter als der Markt, mit einem Erfolg von bloß 2.1% im gleichen Zeitraum unter Berücksichtigung des Zinseszins-Effekts. Der Vergleich der Entwicklung des

spekulativen Portfolios mit dem optimalen Portfolio laut Strategie zeigt, dass die Formel eine Aussagekraft besitzt.

Eine weitere Anmerkung zu den Ergebnissen ist die Inflationsrate in den USA im betrachteten Zeitraum von 1929 bis 2016. Diese ist in den oben genannten Beispielen nämlich auch nicht berücksichtigt worden. 100 $ aus dem Jahr 1929 entsprechen einer Kaufkraft von rund 1400 $ im Jahre 2016 [7]. Somit können die 100 $ von 1929 nicht direkt mit den 246,000 oder 20 Mio. $ im Jahre 2016 verglichen werden. Es ist aber dennoch ersichtlich, dass ein enormer Erfolg erzielt worden wäre.

Abbildung 2.1 veranschaulicht die Ergebnisse der Konservativen Formel im Vergleich zur Marktentwicklung und der Entwicklung des spekulativen Portfolios. Interessant ist, dass das spekulative Portfolio inflationsbereinigt am Ende des Betrachtungszeitraums weniger Kaufkraft aufweist als zu Beginn der Investition.

Kapitel 3

Programmarchitektur und Design

Ziel dieser Arbeit ist nun die Implementierung eines Programms. das die vorgestellte Strategie automatisiert umsetzt. Bevor mit der Entwicklung gestartet werden kann, muss ein Plan erstellt werden. In diesem Kapitel wird mithilfe von UML Diagrammen die Architektur und Funktionsweise des Programms erklärt. Weiters wird die Datenbank und die persistente Datenspeicherung in tabellarischer Form erläutert.

3.1 UML Diagramme

UML Diagramme sind ein bewährtes Konzept um Programmarchitekturen und -flüsse zu beschreiben. UML ist ein Akronym für Unified Modeling Language, zu Deutsch vereinheitliche Modellierungssprache. Dabei handelt es sich um eine Standardsprache, die unter anderem zur Analyse und zum Design objektorientierter Anwendungen verwendet wird. Es gibt mehrere verschiedene Diagrammtypen, die jeweils unterschiedliche Schwerpunkte haben und so verschiedene Aspekte einer Software besser modellieren können. In diesem Kapitel werden das Klassen- und Aktivitätsdiagramm verwendet. Mithilfe dieser zwei Diagramme kann bereits ein relativ umfangreicher Einblick in die Architektur und Funktionsweise des resultierenden Programms gegeben werden. [19]

Klassendiagramm

Ein Klassendiagramm ist nützlich zur Modellierung von Beziehungen der unterschiedlichen Programmklassen und deren Objekte. Es ist hilfreich, um den logischen Aufbau bzw. die Architektur eines Systems zu verstehen.

Die Funktionsweise und Vorteile von objektorientierter Programmierung genauer vorzustellen würde den Rahmen dieser Arbeit sprengen, daher werden in den folgenden Zeilen bloß die Grundkonzepte kurz erläutert. Im Prinzip ist jede Klasse eine Vorlage für Objekte. Ein Objekt ist eine Instanz einer Klasse, wobei jede Klasse normalerweise durch beliebig viele Objekte instantiiert werden kann. Jedes Objekt einer gewissen Klasse enthält genau die Attribute (Variablen) und die verfügbaren Funktionen (Methoden) die in der Klasse spezifiziert wurden. So können gewisse Funktionsweisen eines Programms besser abstrahiert und mithilfe eigener Klassen abgebildet werden. Im UML Diagramm findet man im oberen Teil einer modellierten Klasse die Attribute und im unteren Teil die Methoden. Beziehungen zwischen Klassen werden mit Linien gekennzeichnet.

In vielen Softwarearchitekturen findet man komplexe Beziehungen und Vererbungen zwischen Klassen vor, in der Implementierung dieser Arbeit wurde jedoch die Architektur sinngemäß relativ einfach gehalten. Es existiert eine Main Klasse, die für die Durchführung der Gesamtlogik und Verteilung der Aufgaben verantwortlich ist. Diese Main Klasse wird beim Ausführen des Programms bloß ein einziges Mal als Objekt instantiiert. Durch das Aufrufen der run() Methode des Main Objekts wird dann der gesamte Programmablauf gestartet. Das Main Objekt besitzt eine Referenz auf drei verschiedene Klassen: den Datafetcher, Alerter und das Portfolio. Diese Klassen werden ebenfalls bloß ein einziges Mal instantiiert. Die Aufgaben, die diese Klassen jeweils erfüllen, sind unterschiedlicher Natur:

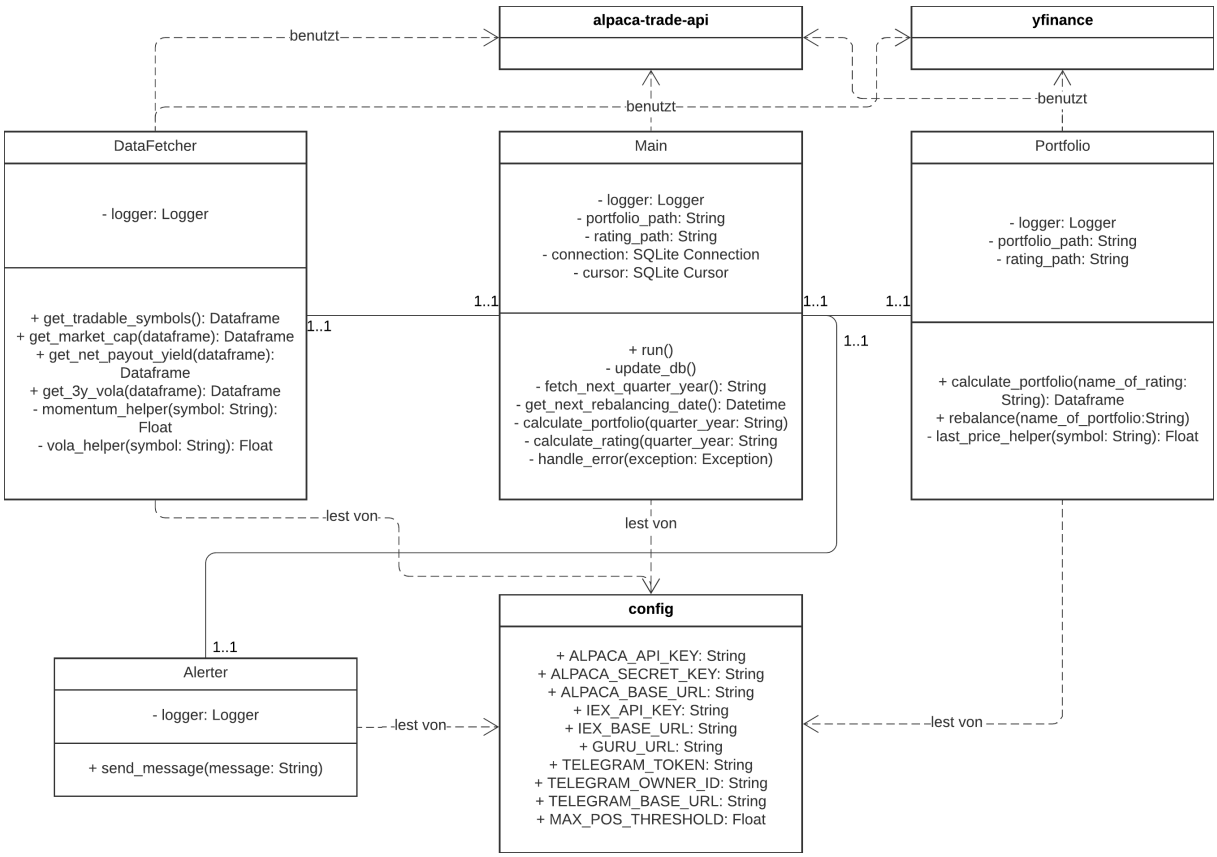

Abbildung 3.1: UML Klassendiagramm der Implementierung

- DataFetcher:

  Diese Klasse ist dafür zuständig, die benötigten Daten von externen APIs (Programmschnittstellen) zu holen. Die Hauptaufgabe liegt in der Berechnung für die neue Bewertung der Aktien.

- Portfolio:

  Diese Klasse deckt die vollständige Funktionsweise ab, die benötigt wird, um einen Portfoliovorschlag basierend auf dem Rating zu erstellen und sodann auch mithilfe des verwendeten Brokers umzusetzen.

- Alerter:

  Der Alerter dient dazu, Nachrichten, die wichtige Informationen über den Zustand des Programms enthalten, über einen Telegram Bot zu senden.

Weiters existiert eine config Datei, die wichtige Konfigurationen wie beispielsweise URLs oder Keys, die für den Zugriff auf externe APIs nötig sind, enthält. Diese Informationen sind für alle Klassen zugänglich und können gelesen werden. Der Sinn, diese Informationen in einer eigenen Datei auszulagern, ist der, dass man diese, wenn nötig, an einer zentralen Stelle schnell adaptieren kann.

Die extern verwendeten APIs, welche eine eigene verwendbare Bibliothek anbieten, werden im Klassendiagramm ebenfalls als Schnittstellen modelliert, wobei aber auf die Modellierung der Attribute und Funktionen aufgrund ihres großen Umfangs verzichtet wird. Die verwendeten Bibliotheken werden in Kapitel 4.3 im Detail einzeln näher erklärt.

Das resultierende Klassendiagramm kann in Abbildung 3.1 nachgelesen werden.

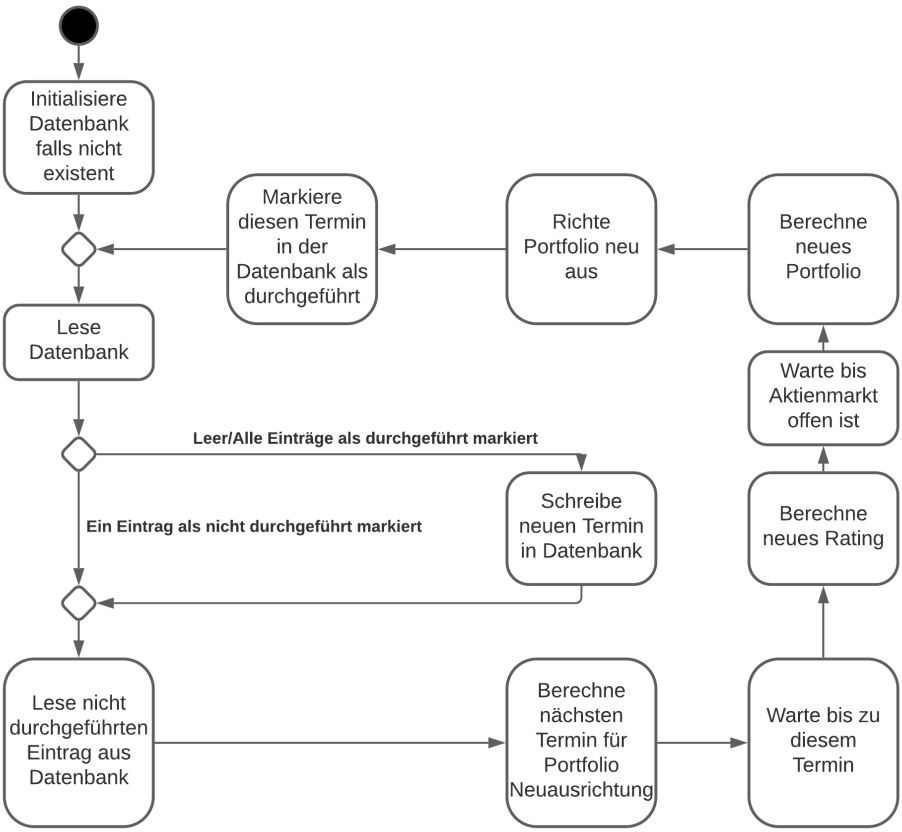

Abbildung 3.2: UML Aktivitätsdiagramm der Implementierung

Aktivitätsdiagramm

Ein Aktivitätsdiagramm beschreibt, wie der Name bereits vermuten lässt, die Aktivitäten, die ein Programm durchführt. Es ist nützlich, um einen Überblick über den Programmfluss zu bekommen. In der fertigen Implementierung gibt es natürlich viele verschiedene Szenarien, die man mithilfe eines Aktivitätsdiagramms beschreiben könnte. Dabei könnte auch beliebig tief ins Detail gegangen werden. Ziel des Aktivitätsdiagramms in dieser Phase der Projektbeschreibung ist aber nicht, Entscheidungen des Programms auf unterster Ebene zu modellieren, sondern die Funktionsweise des Gesamtablaufs zu modellieren.

Aktivitätsdiagramme sind relativ einfach zu lesen, wodurch auf eine detaillierte Erklärung verzichtet wird. Der schwarze Punkt ist der Startpunkt und die Vierecke mit Text sind einzelne Aktivitäten. Bei den kleineren Rauten handelt es sich um Verzweigungen/Zusammenführungen, die den Programmfluss für unterschiedliche Zustände modellieren. Aktivitätsdiagramme besitzen meist noch mehr Elemente, die aber für die Zwecke dieser Modellierung nicht erforderlich sind.

Das resultierende Aktivitätsdiagramm ist in Abbildung 3.2 dargestellt.

3.2 Datenstruktur

Die folgenden Absätze erklären die verwendete Struktur der Datenbank und der tabellarischen Speicherung.

Datenbank

Die Datenbankplanung ist sehr simpel gehalten, weil sie im Grunde bloß eine Funktion erfüllt: die Speicherung der Termine, an denen das Portfolio neu ausgerichtet wird.

Im Grunde hätte man für diese simple Funktionsweise auf eine Datenbank verzichten können, im Falle von Erweiterungen ist aber eine bereits vorhandene Datenbank hilfreich.

Die Datenbank enthält eine Tabelle, die folgende Struktur aufweist: Eine Spalte mit Textwerten für das Quartal und Jahr, in dem das Portfolio neu ausgerichtet werden muss, bzw. worden ist. Eine zweite Spalte, die angibt, ob das Portfolio an diesem Termin bereits neu ausgerichtet worden ist. Ein Beispiel wie diese Tabelle zum Stand April 2021 ausgesehen hat, findet man in Tabelle 3.1.

quarter_year	rebalanced
Q3_2021	0
Q2_2021	1
Q1_2021	1

Tabelle 3.1: Datenbank Tabelle für Neuausrichtungstermine

Tabellarisch

Es gibt alternativ zu Datenbanken auch andere Methoden um Daten in tabellarischer Form speichern bzw. verwenden zu können. Dabei geht aber meist der Vorteil von Datenbanken, Relationen zwischen Einträge verschiedener Tabellen effizient abbilden und abfragen zu können, verloren. Im Gegenzug dazu sind aber alternative Methoden oftmals leichter umzusetzen und für kleinere Projekte durchaus ausreichend. Eine Methode, die in diesem Projekt verwendet wird, ist die Speicherung als .csv Datei. Dabei sind die Einträge der Tabellen in jeweils einer neuen Zeile gespeichert, wobei die einzelnen Spalten durch Beistriche getrennt sind. Daher kommt auch der Name der Dateiendung .csv, kurz für comma-seperated-values.

In diesem Programm werden tabellarische Datenstrukturen ständig verwendet, z.B. bei der Berechnung des Ratings oder beim Konstruieren des Portfolios. Das erleichtert die Programmierung um ein Vielfaches. Zur Verwendung von tabellarischen Datenstrukturen, auch Dataframes genannt, wird die externe Bibliothek Pandas verwendet. Eine genauere Erklärung dazu findet man in Kapitel 4.3.

Das Format der verwendeten Tabellen ist unterschiedlich, ein Beispiel für einen Ausschnitt eines fertig berechnetes Rating kann man in Tabelle 3.2 nachlesen.

Symbol	Momentum	Market Cap	Volatility	Momentum Ranking
AAPL	1.694016	2272940317440	0.022213725	15
GOOGL	1.3833369	1414264082964	0.0195211	61
⋮	⋮	⋮	⋮	⋮
EBAY	1.7231048	42405380199	0.01921186	12
Symbol	**NPY**	**NPY Ranking**	**Rating**	**Final Ranking**
AAPL	3.95	116	65.5	8
GOOGL	2.21	238	149.5	68
⋮	⋮	⋮	⋮	⋮
EBAY	12.96	2	7.0	1

Tabelle 3.2: Tabellarische Speicherung eines Ratings

Einen Ausschnitt eines fertigen Portfolios, für das ebenfalls ein Pandas Dataframe zur Speicherung verwendet wird, findet man in Tabelle 3.3

Symbol	**Price**	**Shares**	**Total**
AAPL	134.81	8	1078.48
GOOGL	2095.13	1	2095.13
⋮	⋮	⋮	⋮
EBAY	62.01	17	1054.17

Tabelle 3.3: Tabellarische Speicherung eines Portfolios

Kapitel 4

Implementierung

Nun zum Hauptteil, der Entwicklung des Programms.

Dieses Kapitel beschreibt die verwendete Programmiersprache Python, die in der Implementierung verwendeten externen Bibliotheken, die Umsetzung der Datenspeicherung und die genaue Funktionsweise der wichtigsten verwendeten Algorithmen.

Weiters wird erklärt, wodurch im Programm verhindert wird, dass bei einem allfälligen Auftreten von externen oder internen Fehler bzw. Errors, das Programm instabil wird oder aber unbemerkt abstürzt.

4.1 Python

Als Sprache zur Umsetzung wurde Python gewählt. Bei Python handelt es sich um eine höhere Programmiersprache, die bereits ein sehr großes Level an Abstraktion beinhaltet. Das Ziel von Python ist, einen gut lesbaren und verständlichen Programmcode zu erschaffen. Python wird im Gegensatz zu anderen bekannten Sprachen wie Java oder C nicht kompiliert, sondern zur Laufzeit von einem Interpreter in Maschinencode umgewandelt. Das und die Eigenschaft, bei Variablendeklarationen keinen konkreten Typ angeben zu müssen, machen Python als Programmiersprache in der Laufzeit etwas langsamer als kompilierte Sprachen. Diese schlechtere Performance wird aber durch die Einfachheit der Sprache kompensiert. [32]

Python erfreut sich einer stark wachsenden Community und ist laut PYPL [31] die beliebteste Programmiersprache 2021. Das war aber bei der Wahl der Programmiersprache für die Umsetzung dieses Projektes nur nebensächlich. Der Hauptgrund, warum die Wahl auf Python fiel, wo auch Alternativen wie beispielsweise Java die Aufgabe sehr gut erfüllt hätten, lag auf der Verfügbarkeit der externen Bibliotheken. Pandas, alpaca-trade-api und yfinance, welche in den folgenden Absätzen 4.3 noch vorgestellt werden, kennen kaum vergleichbaren Alternativen in anderen Programmiersprachen.

Bei der Vorstellung der Implementierung wird von einem gewissen Grundverständnis der Programmierung in Python ausgegangen. Auf die Erklärung der Syntax und Semantik der Programmiersprache wird in dieser Arbeit verzichtet.

4.2 API

In diesem Absatz muss für das weitere Verständnis der Begriff API kurz geklärt werden. API ist eine Abkürzung für Application Programming Interface, zu Deutsch Programmierschnittstelle. Eine Programmierschnittstelle ermöglicht das Verbinden von Programmen, indem es eine Kommunikation zwischen diesen ermöglicht. Diese Kommunikation findet meist über HTTP statt. HTTP steht für HyperText Transfer Protocol und wird im Web ständig verwendet. [10]

Für dieses Projekt sind APIs vonnöten, um die benötigten Daten zu den jeweiligen Aktien zu bekommen und um das resultierende Portfolio bei einem Broker abzubilden.

4.3 Verwendete Externe Bibliotheken

In den folgenden Absätzen werden die verwendeten externen Bibliotheken einzeln im Detail vorgestellt und erklärt, wofür sie zur Umsetzung des gesamten Programms benötigt werden.

Pandas

Pandas ist eine Bibliothek, die vor allem dafür da ist, sogenannte Dataframes als Datenstruktur anzubieten. Diese Dataframes sind eine tabellarische Datenstruktur und erfüllen die Aufgaben, die in Absatz 3.2 vorgestellt wurden. Pandas bietet außerdem eine simple Möglichkeit an, diese Dataframes als .csv Dateien persistent zu speichern und unkompliziert wieder zu laden. [29]

Weiters erleichtert Pandas die Manipulation der Daten in tabellarischer Form, was bei der Umsetzung des in Absatz 4.6 erklärten Bewertungsalgorithmus von Vorteil ist.

Numpy

Numpy wird vor allem im Data Science und Machine Learning Bereich genutzt. Es bietet performante Datenstrukturen wie Arrays und Matrizen. In diesem Projekt wird Numpy lediglich zur genaueren Berechnung größerer Divisionen bzw. der Standardabweichung für eine Liste von Zahlen verwendet. [22]

Yahoo Finance API (yfinance)

Die Yahoo Finance API oder auch yfinance genannt ermöglicht das Abrufen von Daten der Website Yahoo Finanzen. Yahoo Finanzen bietet einen kostenlosen Zugang zu historischen Preisen von Aktien. Somit ist yfinance die verwendete Bibliothek, um mittels Programmcode an diese Daten zu kommen. Diese Daten sind bei der Umsetzung der Konservativen Formel zur Berechnung des Momentums und der Volatilität erforderlich. Die Yahoo Finance API ist keine offiziell von Yahoo entwickelte Programmierschnittstelle, was bei der Implementierung zu gelegentlichen Problemen führte. Mehr dazu in Kapitel 5.2. [30]

Ein großer Vorteil von yfinance für die Umsetzung dieses Projekt ist, dass die angefragten Daten direkt als Pandas Dataframe, also in der gewünschten tabellarischen Datenstruktur, retourniert werden.

Alpaca Trade API (alpaca-trade-api)

Diese Bibliothek ist zur einfachen Verwendung der Alpaca API entwickelt worden. Alpaca Markets ist der Broker, der zur Abwicklung der Aktienkäufe verwendet wird. Es handelt sich dabei um einen der wenigen Broker, die das Handeln mit virtuellem Geld, Paper Trading genannt, ermöglichen und gleichzeitig auch eine gut dokumentierte API haben, auf die kostenlos zugegriffen werden kann. [21]

Die Bibliothek alpaca-trade-api bietet nun eine Vereinfachung der API, indem sie die Funktionen der Alpaca Markets API in Python direkt verfügbar macht und aufwändige Prozesse wie Authentifizierung der Anfragen an die API und Kodierung der Daten in ein passendes Format übernimmt. [26]

requests

Requests ermöglicht HTTP requests in Python zu senden.

Das wird benötigt, um externe APIs verwenden zu können, die keine eigene Bibliothek haben, um deren Funktionsweise in Python abstrahiert verfügbar zu machen.

Im Projekt werden nahezu alle requests aber nicht über die requests Bibliothek gesendet, sondern über aiohttp, welche eine asynchrone Durchführung von requests ermöglicht. Dazu wird noch näher eingegangen werden.

Zur Umsetzung der Konservativen Formel sind bereits die historischen Preisdaten und die Abwicklung der Käufe und Verkäufe durch die vorher vorgestellten Bibliotheken abgedeckt. Es fehlen noch APIs, die die Marktkapitalisierung und Nettoauszahlungsrendite einer Aktiengesellschaft liefern.

Für die Marktkapitalisierung wird über HTTP eine Anfrage an IEX Cloud [9] gesendet. Dabei sind 50.000 Anfragen pro Monat an diese API kostenlos verfügbar, was für die Zwecke dieses Projekts mehr als ausreichend ist.

Für die Nettoauszahlungsrendite muss, weil kein anderer erschwinglicher Anbieter gefunden werden konnte, eine Website maschinell ausgelesen werden. Mehr dazu in der Vorstellung von beautifulsoup4.

beautifulsoup4

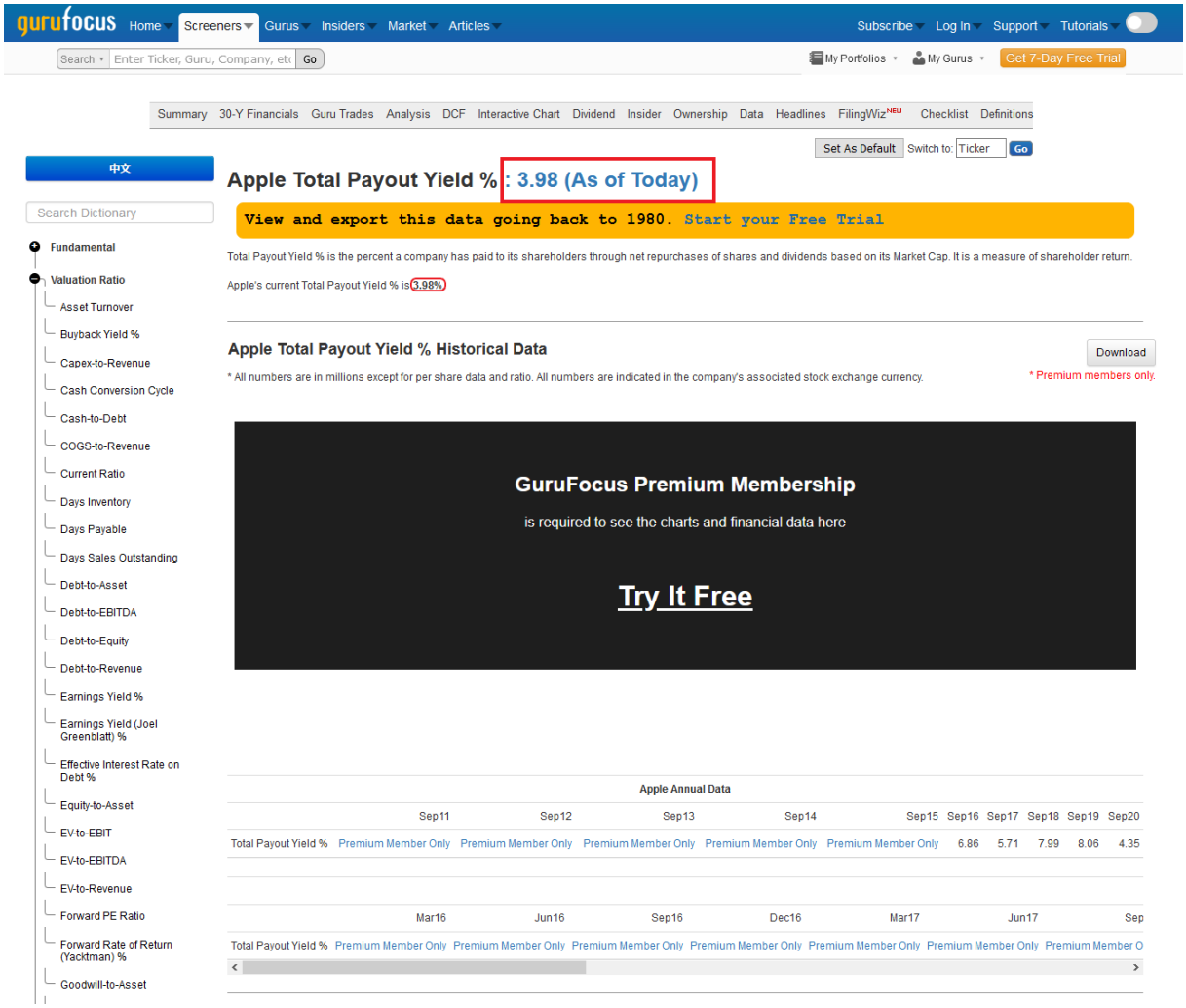

Abbildung 4.1: Nettoauszahlungsrendite von Apple auf gurufocus.com [16]

Diese Bibliothek ermöglicht das Filtern von Daten von einer HTML Website, sodass aus den vielen Elementen des Quellcodes einer Website die gewünschte Information heraus gelesen werden kann. [28]

Für dieses Projekt wird beautifulsoup4 zum Beschaffen der Daten zur Nettoauszahlungsrendite verwendet. Dazu werden gewisse Endpunkte der Website `https://www.gurufocus.com` geladen und in Folge die gewünschte Information extrahiert. [16]

Möchte man zum Beispiel die Nettoauszahlungsrendite für Apple herausfinden, so wird die URL `https://www.gurufocus.com/term/TotalPayoutYield/aapl/` aufgerufen. Das Resultat kann man in Abbildung 4.1 nachlesen. Dabei ist die gewünschte Information in Blau geschrieben relativ weit oben in der Mitte zu finden, in der Abbildung ist sie mit einem roten Viereck umrandet. Um diese Information aus dem Quelltext der Website auszulesen, ist beatifulsoup4 das richtige Werkzeug. Dabei muss eine Eigenschaft gefunden werden, die genau auf und nur auf diese Information passt. In diesem Fall ist die Nettoauszahlungsrendite die einzige Information die auf dieser Website von einem strong tag (3.98%) umgeben ist und innerhalb dieses strong tags auch ein „%“ vorkommt. Genau

genommen extrahieren wir hierbei nicht den blau gedruckten Text, sondern den fett gedruckten Text, der einen Absatz darunter zu finden ist. Dieser ist in der Abbildung mit einem runden, roten Rechteck markiert. Die enthaltene Information ist aber dieselbe.

So sieht diese beschriebene Extraktion der Information aus der Website in Programmcode aus:

```
r = requests.get("https://www.gurufocus.com/term/TotalPayoutYield/aapl/")
soup = BeautifulSoup(r, 'html.parser')
soup = soup.findAll("strong")
for s in soup:
    if '%' in s.next:
        return float(s.next[:-1])
return numpy.NAN
```

Dieses Prozedere muss für jede einzelne Aktie durchgeführt werden, wobei selbstverständlich der URL jedes Mal ein anderer ist. Welche URL für welche Aktie aufzurufen ist, kann mit dem Kürzel der jeweiligen Aktie herausgefunden werden. Die Aktie von Apple hat beispielsweise „AAPL“ als Abkürzung, sodass die aufzurufende URL lautet: `https://www.gurufocus.com/term/TotalPayoutYield/aapl/`. Sollte eine Aktie nicht auf gurufocus gefunden werden, so würde unser Programm numpy.NAN zurückgeben, was angibt, dass diese Aktie keine verfügbaren Daten hat.

Wie man auf die einzelnen Kürzel der jeweiligen Aktien kommt, wird in Kapitel 4.6 näher erklärt.

asyncio

Wie bereits beschrieben, müssen sehr viele Anfragen an externe APIs hintereinander gesendet werden. Um diesen Prozess zu beschleunigen und nicht jede einzelne Anfrage iterativ durchführen zu müssen, wird asyncio, eine Bibliothek die Asynchrone Programmierung in Python ermöglicht, verwendet.

Asyncio ermöglicht, mehrere Aufgaben parallel durchzuführen und bei der Ausführung der nächsten Aufgabe nicht zwangsmäßig auf den Abschluss der Vorherigen warten zu müssen. [27]

aiohttp

Um nun HTTP requests asynchron zu senden, wird eine neue Bibliothek benötigt, die asynchrone requests unterstützt. Die ursprünglich vorgestellte requests Bibliothek kann für asynchrone Programmierung nicht verwendet werden. Als Ersatz wird aiohttp verwendet. [25]

Eine asynchrone Durchführung von mehreren requests zur Ermittlung der Nettoauszahlungsrendite sieht folgendermaßen aus:

```
symbols = ["AAPL", "GOOG", "EBAY", ..., "TSLA"]
results = []

async def fetch_one(url, session):
    async with session.get(url) as response:
        return await response.read()

async def fetch_all(symbol_list):
    url = "https://www.gurufocus.com/term/TotalPayoutYield/{}/"
    tasks = []
    # Fetch all responses within one Client session,
    # keep connection alive for all requests.
    async with aiohttp.ClientSession() as session:
        for s in symbols:
            task = asyncio.ensure_future(
                        fetch_one(
                            url.format(s.lower()),
```

```
                            session)
                        )
            tasks.append(task)

        responses = await asyncio.gather(*tasks)
        for r in responses:
            soup = BeautifulSoup(r, 'html.parser')
            soup = soup.findAll("strong")
            found = False
            for s in soup:
                if '%' in s.next:
                    results.append(float(s.next[:-1]))
                    found = True
                    break
            if not found:
                results.append(np.NAN)
loop = asyncio.get_event_loop()
future = asyncio.ensure_future(fetch_all(symbols))
loop.run_until_complete(future)

return results
```

4.4 Persistente Datenspeicherung

Dieses Kapitel beschreibt kurz, welche Bibliotheken zur persistenten Datenspeicherung verwendet werden.

Datenbank

Für die Datenbank wird sqlite3 verwendet. Dabei handelt es sich um eine Bibliothek, die das schnelle Implementieren von Datenbanken ermöglicht. Sqlite3 ist bereits in Python als Standardbibliothek enthalten.

Um nun die Datenbank für dieses Projekt zu initialisieren wird folgender Code ausgeführt:

```
import sqlite3
conn = sqlite3.connect("rebalancing_dates.db"))
cursor = conn.cursor()

cursor.execute("""CREATE TABLE
                  IF NOT EXISTS
                  dates (
                        quarter_year text PRIMARY KEY NOT NULL,
                        rebalanced INTEGER
                            CHECK (rebalanced >= 0 AND rebalanced <= 1)
                            NOT NULL
                        );
               """)
conn.commit()
```

Das erstellte cursor Objekt ermöglicht SQL Befehle auf der Datenbank auszuführen. Da aber diese Änderungen zu Beginn bloß auf diesem cursor Objekt existieren, müssen diese mittels unserer Verbindung mit der Datenbank (conn Objekt) bestätigt werden. Das passiert mit der $commit()$ Methode des

conn Objekts. Erst nach dieser Bestätigung sind die tatsächlichen Änderungen auf der Datenbank sichtbar.

SQL ist eine Sprache zur Verwendung von Datenbanken. Auf die Erklärung dieser Sprache wird in dieser Arbeit verzichtet.

Tabellarisch, .csv

Zur tabellarischen Speicherung als .csv Dateien wird, wie bereits erwähnt, die externe Bibliothek Pandas verwendet.

Gespeicherte .csv Dateien können folgendermaßen in das Programm geladen, ausgegeben und unter anderem Namen als Kopie gespeichert werden:

```
import pandas as pd

rating = pd.read_csv("Q1_2021.csv")
print(rating)
rating.to_csv("Q1_2021_Kopie.csv", index=False)
```

Die Möglichkeit Dataframes problemlos als .csv Datei zu speichern und zu laden ist hilfreich, um Datenverluste bei Stromausfälle so gering wie möglich zu halten. Mehr dazu in Kapitel 4.10

4.5 Alerter

Hier wird kurz die verwendete Alerter Klasse vorgestellt. Im Prinzip ermöglicht diese Klasse, Textnachrichten über Telegram zu versenden. Telegram ist ein Messenger Service, vergleichbar mit Whatsapp. Zum Versenden von Nachrichten mittels einem Bot muss zuerst ein Telegram Bot erstellt und dessen ID vermerkt werden. Mittels dieser ID kann man nun über die Telegram API Nachrichten an beliebige Personen über den erstellten Bot senden.

Der Alerter erfüllt bei der Implementierung einen simplen Zweck: Er soll über den Status des Programms informieren. Wenn das Portfolio neu ausgerichtet wird, soll eine Nachricht gesendet werden. Wenn ein Fehler passiert, den das Programm von alleine nicht beheben kann, soll der Programmierer ebenfalls benachrichtigt werden. Der Programmcode, um eine Nachricht über den erstellten Bot zu senden, sieht folgendermaßen aus:

```
def send_message(self, message: str) -> None:
    url = self.base_url + self.token + \
          '/sendMessage?chat_id=' + self.owner + \
          '&parse_mode=Markdown&text=' + message
    requests.get(url)
```

Wie die benötigten IDs von Benutzer und Bot herausgefunden werden können, kann man in diesem Artikel [17] nachlesen. Diese Klasse ist die einzige Art von User Interface, die bei Auftreten eines Fehlers den Programmierer informiert. Informationen bezüglich der Entwicklung des Portfolios können direkt beim Broker Alpaca Markets über dessen Website [21] erlangt werden.

4.6 Bewertungsalgorithmus

Dieses Kapitel beschreibt das Prozedere, das für die Berechnung einer neuen Bewertung durchgeführt wird.

Erstellen eines Universums an Aktien

Zu Beginn muss ein Universum an Aktien, die auch gehandelt werden können, erstellt werden. Das passiert mittels der API unseres Brokers, alpca-trade-api:

```
symbols = alpaca_api.list_assets(status='active')
```

Dieser Programmcode liefert uns eine Liste von Symbolen. Es handelt sich hierbei um eigene Objekte, die unter anderem ein Attribut namens symbol haben. Das symbol Attribut eines Symbols ist eine Abkürzung, die jede Aktie eindeutig beschreibt. Zwecks Einfachheit werden alle Aktien der Liste von nun an mithilfe dieses Kürzels gespeichert, denn alle benötigten Tätigkeiten können mithilfe dieses Kürzels später durchgeführt werden. Apple besitzt beispielsweise das Kürzel „AAPL“, d.h. es würde in unserer fertigen Liste als AAPL aufscheinen.

Nun reicht es aber nicht, ein Universum von Aktien zu haben, die auch aktiv bei dem Broker gehandelt werden können, wir müssen ebenfalls Daten wie historische Preise und Marktkapitalisierung zu diesen Aktien bekommen. Um die Aktien zu beseitigen, von denen wir diese Daten nicht bekommen, rufen wir alle verfügbaren Aktien auf, die der Plattform IEX Cloud bekannt sind. Das passiert mittels simplem HTTP request:

```
iex_symbols = requests.get(config.IEX_BASE_URL +
              "ref-data/iex/symbols?token=" +
              config.IEX_KEY)
```

Das Resultat dieser API Anfrage ist aber im JSON Format, deshalb muss die gewünschten Informationen noch in eine normale Liste formatiert werden:

```
iex_symbols = [s['symbol'] for s in iex_symbols]
```

Nun wird das Universum auf jene Aktien, die in beiden Listen vorkommen, reduziert. Weiters müssen die Aktien entfernt werden, die per Definition keine direkten Aktien einer Aktiengesellschaft darstellen, sondern selbst bereits eigene Fonds, Indizes, ETFs oder Ähnliches sind. Einzelfälle, die manuell ausgenommen werden, sind GOOGL und LBTYK, weil diese Unternehmen mehrere unterschiedliche Arten von Aktien anbieten, abhängig davon, ob ein Stimmrecht bei deren Hauptversammlung existiert, oder nicht. Im resultierenden Universum soll jedes Unternehmen bloß ein einziges Mal vorkommen, ansonsten könnte es passieren, dass in ein Unternehmen wie Google beispielsweise zweimal investiert werden würde, was natürlich nicht erwünscht ist. Der Programmcode zur Reduktion des Universums an Aktien nach den beschriebenen Kriterien sieht folgendermaßen aus:

```
symbols = [s.symbol for s in symbols if s.tradable
            and not any(word.upper() in s.name for word in
                        ["FUND", "ETF", "TRUST", "ETC", "ETN", "INDEX"])
            and s.symbol in iex_symbols
            and s.symbol != 'GOOGL'
            and s.symbol != 'LBTYK']
```

Die finale Liste an Symbolen wird in einer tabellarischen Datenstruktur, einem Pandas Dataframe, gespeichert.

```
df = pd.DataFrame(symbols, columns=['Symbol'])
```

Berechnung der Marktkapitalisierung

Als nächster Schritt wird für jede dieser Aktien die Marktkapitalisierung berechnet. In der konservativen Formel werden nämlich bloß die 1000 größten Unternehmen nach Marktkapitalisierung betrachtet. Das ursprüngliche Universum an Aktien, die gehandelt werden können und zu denen Daten verfügbar sind, umfasst normalerweise rund 7000 Symbole. Das wird nun auf die größten 1000 reduziert.

Um die Marktkapitalisierung einer Aktiengesellschaft zu erhalten, wird für jede Aktie ein HTTP request an IEX Cloud gesendet. Das passiert auf asynchrone Weise, um diesen Prozess zu beschleunigen. Die URL, an die der API request gesendet wird, lautet:
„https://cloud.iexapis.com/stable/stock/<Symbol der Aktie>/stats/marketcap?token=<API Token>“
Das asynchrone Absenden der requests funktioniert analog wie in Absatz 4.3 vorgestellt. Das retournierte Datenformat ist hierbei wieder in JSON, erfordert aber keine weitere Decodierung, weil bloß eine einfache Zahl als Resultat retourniert wird. Eine einzelne asynchrone API Anfrage zur Ermittlung der Marktkapitalisierung sieht folgendermaßen aus:

```
url = self.iex_base_url +
      "stock/{}/stats/marketcap?token=" +
      self.iex_key

# url wird in fetch_all Funktion formattiert um
# das Symbol der Aktie zu enthalten

async with session.get(url) as response:
    return await response.json()
```

Nach dem asynchronen Absenden aller Anfragen wird, wie in Absatz 4.3, eine Liste an Resultaten erhalten. Diese werden in der Tabelle in der Spalte 'Market Cap' hinzugefügt:

```
df['Market Cap'] = results
```

Weil aber nun unser Universum auf die größten 1000 Aktien reduzieren werden soll, muss die Tabelle noch dementsprechend manipuliert werden:

```
df.sort_values(by='Market Cap', ascending=False, inplace=True)
df.reset_index(drop=True, inplace=True)
df = df[:1000]  # größten 1000 werden betrachtet
```

Berechnung der Volatitlität

Nächster Schritt in der Konservativen Formel ist die Volatilität der letzten drei Jahre jeder Aktie zu berechnen. Daraus folgend wird dann nur noch die Hälfte der Aktien betrachtet, die die geringste Volatilität aufweisen.

Um die Volatilität einer Aktie der letzten drei Jahre zu berechnen, werden die jeweiligen Kursverläufe der letzten drei Jahre benötigt. Diese werden über die Yahoo Finance API (yfinance) ermittelt.

Dieser Prozess wird rein iterativ durchgeführt, weil die Bibliothek yfinance keine asynchrone Programmierweise unterstützt. Da es sich aber hierbei bloß noch um 1000 Aktien handelt, ist der zeitliche Aufwand vertretbar.

Zum Berechnungsmodus ist anzumerken, dass hierbei historische Daten der letzten fünf Jahre anstatt der benötigten drei Jahren geladen werden. Einerseits weil Daten von etwas mehr als drei Jahre benötigt werden und andererseits weil yfinance als passendes Zeitintervall bloß zwei bzw. fünf Jahre oder maximale Dauer, für die Yahoo Finance Daten besitzt, als Input akzeptiert. Um die Volatilität einer einzelnen Aktie zu berechnen, dient folgende Funktion:

```
def vola_helper(symbol: str) -> float:
    history = yf.Ticker(symbol).history(
                                period="5y",
                                interval="1d",
                                actions=False
                                )

    # Länge der Einträge muss größer als 757 sein
```

```
    # 252 Tage wird durchschnittlich im Jahr gehandelt
    # 252 * 3 = 756, wir wollen auch die
    # Änderungsrate des ersten Tages --> 757
    if len(history) < 757:
        return np.NAN

    # entferne nicht benötigte Spalten
    history.drop(['Open', 'High', 'Low', 'Volume'],
                 inplace=True, axis=1)

    # betrachte nur gewünschten Zeitraum
    history = history[-757:]

    # berechne tägliche Änderungsrate
    history['ROC'] = history.pct_change(1)

    # entferne ersten Eintrag weil dieser NAN ist
    history = history[1:]

    # retourniere die Standardabweichung = Volatilität
    return history['ROC'].std()
```

Diese Funktion muss nun für jedes einzelne Symbol in unserem Dataframe aufgerufen werden. Dafür eignet sich die apply Funktion von Pandas:

```
dataframe['Volatility'] = dataframe.apply(
                          lambda row: self.vola_helper(row.Symbol),
                          axis=1)
```

Um nun die Hälfte an Aktien mit der geringsten Volatilität zu betrachten, werden folgende Manipulationen am Dataframe durchgeführt:

```
dataframe.dropna(inplace=True)
dataframe.sort_values(by='Volatility', ascending=True, inplace=True)
dataframe.reset_index(drop=True, inplace=True)
dataframe = dataframe[:int(len(dataframe) / 2 + 1)]
```

Dabei ist es möglich, dass nicht mehr ganze 500 Aktien in dem Universum sind, weil manche erst zu kurz am Markt sind und dadurch aussortiert worden sind.

Berechnung und Reihung nach Momentum

Zur Berechnung des 12-1 Monate Momentums wird ebenfalls yfinance verwendet. Die Funktionsweise ist im Prinzip gleich wie jener der Berechnung der Volatilität. Der einzige Unterschied liegt darin, dass zwei Jahre an Daten angefordert werden und als Rückgabewert die Division des letzten bekannten Werts durch jenem von vor 252 Tagen geliefert wird.

Am Ende werden dann folgende Manipulationen am Dataframe durchgeführt, um die Reihung nach Momentum zu erhalten (Indizes starten in der Regel bei 0 anstatt bei 1, daher muss bei der Berechnung der Reihung eins zum Index der jeweiligen Reihung in der Sortierung hinzufügt werden):

```
dataframe.sort_values(by='Momentum', ascending=False, inplace=True)
dataframe.reset_index(drop=True, inplace=True)
dataframe['Momentum Ranking'] = dataframe.index + 1
```

Berechnung und Reihung nach Nettoauszahlungsrendite

Wie die Nettoauszahlungsrendite für jede einzelne Aktie berechnet wird, wurde bereits in Kapitel 4.3, bei der Vorstellung der Bibliothek aiohttp, behandelt. Es werden die Daten von der Website gurufocus.com extrahiert.

Die Resultate dieses Prozedere werden wiederum in der verwendeten Tabelle in einer neuen Spalte gespeichert und dieselbe Reihung durchgeführt wie zuvor nach Momentum.

Berechnung der finalen Reihung

Nun sind in der Tabelle alle benötigten Spalten zur Berechnung der finalen Reihung enthalten. Zuerst wird die Bewertung der einzelnen Aktien berechnet. Diese setzt sich aus dem Durchschnitt der Reihung nach Momentum und Nettoauszahlungsrendite zusammen. Im Programm sieht das folgendermaßen aus:

```
dataframe['Rating'] =   (dataframe['Momentum Ranking']
                        +
                        dataframe['NPY Ranking'])
                        / 2
```

Nach dieser Bewertung wird nun ebenfalls wieder sortiert und eine Reihung vergeben. In diesem Fall wird aber aufsteigend sortiert, weil eine geringe Bewertung besser ist als eine hohe:

```
dataframe.sort_values(by='Rating', ascending=True, inplace=True)
dataframe.reset_index(drop=True, inplace=True)
dataframe['Final Ranking'] = dataframe.index + 1
```

Das Resultat ist eine fertige Bewertung in tabellarischer Form. In der Spalte 'Final Ranking' findet sich die jeweilige Platzierung der Aktien. Investiert wird dann in die besten 100 nach dieser Reihung.

4.7 Portfolio-Konstruktion

Der Algorithmus zur Konstruktion des Portfolios berechnet ein fertiges Portfolio basierend auf einer existierenden Bewertung. Es ist der nächste Schritt, der durchgeführt werden muss.

Die Bedingung, die der Algorithmus erfüllen soll, ist hauptsächlich, eine annähernd gleich verteilte Investition in die einzelnen Aktien zu erreichen. Im fertigen Programm wird diese Aufgabe von der Portfolio-Klasse erfüllt.

Zu Beginn der Prozedur existiert ein Dataframe, welches die Bewertung enthält. Von dieser Tabelle werden die besten 100 betrachtet und auf die Spalte mit den Symbolen der Aktien reduziert:

```
df = pd.read_csv(os.path.join(self.rating_path, name_of_rating + ".csv"))
df = df[:100]
df = df.filter(['Symbol'])
```

Um nun eine annähernd gleiche Verteilung zu berechnen, müssen zuerst die aktuellen Preise der einzelnen Aktien berechnet werden. Das funktioniert ähnlich der Berechnung des Momentums oder der Volatilität im Bewertungsalgorithmus. Es wird für jede Zeile des Dataframes eine Funktion ausgeführt, die den letzten bekannten Preis der Aktie retournieren soll:

```
def last_price_helper(self, symbol: str) -> float:
    price = self.alp_api.get_last_trade(symbol)
    return price.price

df['Price'] = df.apply(
              lambda row: self.last_price_helper(row.Symbol),
              axis=1)
```

Jetzt muss noch herausgefunden werden, wie viel Geld für die Investition zur Verfügung steht. Das passiert mittels der alpaca-trade-api:

```
equity = float(self.alp_api.get_account().equity)
```

Da aber die Kurse der Aktien nicht in Stein gemeißelt sind und sich bis zum eigentlichen Kauf noch verändern können, ist es sinnvoll, nicht 100% planmäßig zu investieren, sondern einen kleinen Puffer zu lassen. In diesem Projekt wurde eine Investition von 99% angestrebt, was unter normalen Marktbedingungen genügend Sicherheit bietet, alle Aktien schlussendlich in der gewünschten berechneten Anzahl kaufen zu können. Somit wird der eigentliche Betrag, der investiert werden soll, folgendermaßen berechnet:

```
# equity ist bereits vorhanden
cash = float(self.alp_api.get_account().cash)
if cash < (equity*(1-config.MAX_POS_THRESHOLD)):
    equity = equity - (equity*(1-config.MAX_POS_THRESHOLD)-cash)

equity = equity*config.MAX_POS_THRESHOLD
```

Dabei ist in der Konfigurationsdatei config ein Wert gespeichert, der diese 99% repräsentiert, genannt MAX_POS_THRESHOLD. In der if Abfrage zuvor wird verglichen, ob das momentan verfügbare Cash die gewünschte Quote von 1% im Verhältnis zum Gesamtvermögen erfüllt. Wenn nicht, bedeutet das, dass beim letzten Ausrichten des Portfolios der Puffer teilweise benötigt wurde. Um den Puffer groß genug zu halten, wird unser Gesamtvermögen so weit verringert, dass dieser 1% Puffer wiederhergestellt ist. Schlussendlich wird erneut der 1% Puffer mit einberechnet, was den eigentlichen Sicherheitsabstand für die kommende Neuausrichtung darstellt.

Nun muss berechnet werden, wie viel in eine einzelne Aktie investiert werden darf, um eine gleich verteilte Investition zu erreichen. Dafür wird das vorhandene Vermögen einfach durch die Gesamtzahl, in die investiert werden soll (100 Stück), dividiert. Das Resultat wird in der Variable pos_size gespeichert.

```
pos_size = numpy.divide(equity, len(df))
```

Da jedoch beim Broker Alpaca zur Zeit der Umsetzung bloß ganze Aktien gekauft werden können, wird es vorkommen, dass manche Aktienkurse teurer sind als die berechnete pos_size. Andere Aktien wiederum sind billiger, was zu einer Investition mehrerer Stücke führt, wobei nahezu niemals das verfügbare Budget pro Aktie optimal ausgeschöpft wird. Um dieses Problem zu lösen, wird ein relativ simpler Greedy Algorithmus konstruiert, der primär darauf abzielt, von jeder Aktie mindestens ein Stück zu kaufen und darauf folgend das vorhandene Gesamtvermögen so gut wie möglich auszuschöpfen.

Zu Beginn dieses Algorithmus wird zuerst die initiale Anzahl pro Aktie, in die investiert werden kann, berechnet. Dafür wird in der Spalte „Shares“ das abgerundete Resultat der Division von pos_size und Preis der Aktie gespeichert:

```
df['Shares'] = df.apply(
               lambda row: int(np.divide(pos_size, row.Price)),
               axis=1)
```

In der Spalte „Total“ wird gespeichert, wie viel gesamt in diese Aktie investiert werden würde, also $Total = Price * Shares$:

```
df['Total'] = df['Price'] * df['Shares']
```

Nun folgt der eigentliche Greedy Algorithmus:

```
def greedy_df(to_manipulate: pd.DataFrame) -> pd.DataFrame:
    to_manipulate.sort_values(by='Price', ascending=True, inplace=True)
    has_changed = False

    shares = to_manipulate['Shares'].tolist()
    manipulate = to_manipulate.copy()

    for i in range(len(shares)):
        # Garantiert dass in jede Aktie investiert wird
        if shares[i] == 0:
            shares[i] = 1
            has_changed = True
    manipulate['Shares'] = shares
    manipulate['Total'] = manipulate['Price'] * manipulate['Shares']

    # überprüfe ob wir über dem Vermögen sind nachdem
    # in jede Aktie min. einmal investiert wird
    # oder ob wir noch Potential nach oben haben
    if manipulate['Total'].sum() <= equity:
        downsizing = False
    else:
        downsizing = True

    for i in range(len(shares)):
        if downsizing and shares[i] > 1:
            shares[i] = shares[i] - 1
        elif not downsizing:
            shares[i] = shares[i] + 1

        # überprüfe in jeder Iteration ob Algorithmus fertig ist
        manipulate['Shares'] = shares
        manipulate['Total'] = manipulate['Price'] * manipulate['Shares']
        if not downsizing and not manipulate['Total'].sum() <= equity:
            shares[i] = shares[i] - 1
            manipulate['Shares'] = shares
            manipulate['Total'] = manipulate['Price'] * manipulate['Shares']

            # durch ein Stück pro Aktie bereits über Ziel hinausgeschossen
            # --> verringere Stückzahl anderer Aktien in neuer Iteration
            if has_changed:
                return greedy_df(manipulate)
            # ansonsten normal fertig geworden
            else:
                return manipulate
        elif downsizing and manipulate['Total'].sum() <= equity:
            return manipulate
        has_changed = True

    # Eine Iteration von Verkleinerungen reicht nicht
    if manipulate['Total'].sum() > equity:
        return greedy_df(manipulate)
```

```
    # Eine Iteration von Vergrößerungen reicht nicht
    else:
        return greedy_df(manipulate)
```

Die Funktionsweise ist folgende: Zu Beginn wird die Tabelle aufsteigend nach den Preisen sortiert. Dann wird über jede einzelne Aktie iteriert. Falls bei einer als zu investierende Anzahl eine Null steht, wird sie durch eine Eins ersetzt. Das garantiert, dass in jede Aktie investiert wird.

Nun wird mittels der Summe der 'Total' Spalte überprüft, ob durch diese Änderung das verfügbare Vermögen übertroffen wurde, oder ob noch ein Rest verfügbar ist. Diese Überprüfung ist ausschlaggebend, ob der Algorithmus in Folge versucht, mehr Aktien zu kaufen, um das Potenzial voll auszuschöpfen, oder weniger Aktien zu kaufen, um innerhalb des verfügbaren Rahmens zu gelangen.

Es wird über jede einzelne Aktie iteriert. Dabei wird versucht, entweder eine Aktie mehr oder weniger zu kaufen als ursprünglich geplant. Dann wird überprüft, ob das Ziel bereits erreicht ist, oder nicht. Falls das Ziel erreicht und mittels der Erhöhung der zu kaufenden Stück für diese konkrete Aktie beispielsweise das Vermögen voll ausgeschöpft ist, wird der Algorithmus abgebrochen. Sollte nach einer vollständigen Iteration das Ziel noch nicht erreicht worden sein, so wird eine erneute Iteration durchgeführt.

Dieser Greedy Algorithmus liefert zwar nahezu niemals eine perfekte Gleichverteilung oder Annäherung an das verfügbare Vermögen, es reduziert aber den Abstand zum gewünschten investierten Vermögen im Vergleich zur initialen Berechnung der Stückzahlen enorm. Für die Zwecke dieser Umsetzung ist dieser Algorithmus ausreichend.

Nach Aufruf dieses Greedy Algorithmus existiert ein fertiges Dataframe, welches das zu investierende Portfolio darstellt. Dieses Dataframe enthält drei Spalten: Symbol, Price und Total. Es besitzt exakt jene Struktur, wie in Kapitel 3.2 in Tabelle 3.3 vorgestellt. Aufgerufen wird der Algorithmus mit unserem Dataframe als Input folgendermaßen:

```
df = greedy_df(df)
```

4.8 Algorithmus zur Neuausrichtung des Portfolios

Als letztes Glied in der Kette muss das berechnete Portfolio auch beim verwendeten Broker Alpaca Markets umgesetzt werden. Dazu wird das in den folgenden Absätzen beschriebene Prozedere durchgeführt.

Zu Beginn werden alle allfällig noch offenen Käufe oder Verkäufe storniert, um Konflikte zwischen Aufträgen zu vermeiden. Im Normalfall sollten, weil dieser Prozess bloß alle drei Monate durchgeführt wird, sowieso keine noch offenen Aufträge existieren.

```
self.alp_api.cancel_all_orders()
```

Im Falle einer erstmaligen Investition in das Portfolio sind die weiteren Schritte trivial. Es werden von jeder Aktie genau so viele Stücke gekauft, wie im berechneten Portfolio veranschlagt. Dieses Prozedere soll aber auch funktionieren, wenn bereits in ein bestehendes Portfolio investiert wurde und dieses an das neu berechnete Portfolio angepasst werden soll.

Dazu muss zuallererst eine Liste von Positionen, in die bereits investiert wurde, vom Broker abgefragt werden. Als Datenstruktur zur Speicherung dieser Positionen wird ein Dictionary verwendet. Das speichert die Einträge als Kombination von Key und Value, also Schlüssel und Werten. Als Schlüssel wird das Symbol der Aktie verwendet und als Wert die Stückzahl:

```
positions = {}
pos_api = self.alp_api.list_positions()
for p in pos_api:
    positions[p.symbol] = int(p.qty)
del pos_api
```

Nun wird über alle Aktiensymbole, die zurzeit im gehaltenen Portfolio vorkommen, iteriert. Dabei wird überprüft, ob das Symbol im neuen Portfolio noch vorkommt. Wenn nicht, dann werden alle Stücke dieser Aktie verkauft. Sollte das Symbol im neuen Portfolio ebenfalls vorkommen, dann wird verglichen, ob im neuen Portfolio weniger Stück von dieser Aktie gehalten werden sollen und, sofern das der Fall ist, so viele Aktien verkauft bis die Stückzahlen übereinstimmen. Als erster Schritt der Neuausrichtung werden also alle Aktien verkauft, die im neuen Portfolio nicht vorkommen sollen. Das dient dazu, dass später, bei den Einkäufen, genügend Geld vorhanden ist, um auch alle gewünschten Aktien kaufen zu können. Im Programmcode sieht das beschriebene Prozedere folgendermaßen aus:

```
for s in positions.keys():
    if s not in portfolio.keys():
        self.alp_api.close_position(s)

    elif int(positions[s]) > int(portfolio[s]):
        diff = positions[s] - portfolio[s]
        self.alp_api.submit_order(
                    symbol=s,
                    qty=diff,
                    side='sell',
                    type='market',
                    # gtc = good till cancelled
                    # d.h. es kann nur manuell storniert werden
                    time_in_force='gtc'
                    )
```

Nach dem Verkauf der nicht benötigten Aktien müssen die Aktien gekauft werden, die im Portfolio enthalten sein sollen. Das passiert folgendermaßen:

```
 for s in portfolio.keys():
    if s not in positions.keys():
        self.alp_api.submit_order(
            symbol=s,
            qty=portfolio[s],
            side='buy',
            type='market',
            time_in_force='gtc'
        )
    else:
        diff = int(portfolio[s]) - int(positions[s])

        # die Anzahl passt, es wird nichts unternommen
        if diff <= 0:
            continue

        self.alp_api.submit_order(
            symbol=s,
            qty=diff,
            side='buy',
            type='market',
            time_in_force='gtc'
        )
```

Am Ende des Prozedere werden die Aufträge an den Broker übermittelt und das Portfolio sollte fertig ausgerichtet sein.

4.9 Neuausrichtung jedes Quartal

In den vorherigen Absätzen sind die einzelnen Algorithmen, die zur Berechnung, Erstellung und Neuausrichtung des Portfolios benötigt werden, vorgestellt worden. Nun fehlt noch der Teil des Programms, der gewährleistet, dass diese Algorithmen alle drei Monate, also zu Beginn eines Quartals, wiederholt werden. Diese Logik findet sich ausschließlich in der Main Klasse.

Um ein konkretes Quartal, in dem das Portfolio neu ausgerichtet werden soll, bzw. worden ist, zu speichern, wird eine aussagekräftige Abkürzung in der Datenbank verwendet. So bedeutet „Q1_2021“ das erste Quartal im Jahr 2021. Neben dem Texteintrag finden wir laut unserer Datenbankplanung eine 1 oder 0, je nachdem ob die Neuausrichtung für dieses Quartal stattgefunden hat oder nicht.

Zeitpunkte, die mit einem Quartal in Verbindung stehen, sind fest einprogrammiert: Das erste Quartal hat als Richtdatum den 1. Januar, das zweite Quartal den 1. April, das dritte Quartal den 1. Juli und das vierte Quartal den 1. Oktober. Richtdatum deshalb, weil nicht immer der Markt an diesen Tagen offen ist und die Neuausrichtung nur stattfinden kann, wenn der Aktienmarkt auch offen ist. Sollte der Markt zu diesem Termin geschlossen sein, so wird am nächst offenen Markttag neu ausgerichtet. Das Prozedere, für das jeweilige Quartal ein konkretes Datum zu bekommen, sieht folgendermaßen aus:

```
def get_next_rebalancing_date(self, quarter_year: str) -> datetime:
    months = {'Q1': 1, 'Q2': 4, 'Q3': 7, 'Q4': 10}

    month, year = quarter_year.split("_")
    month = months[month]
    ts = year + "-" + str(month) + "-1"

    calender = self.alp_api.get_calendar(start=ts)
    return calender[0].date.to_pydatetime().date()
```

Wann der Markt offen ist, wird direkt vom Broker erfahren. Dazu wird im obigen Code wieder die alpaca-trade-api Bibliothek verwendet.

Nun wird für ein konkretes Quartal das jeweilige Datum zur Neuausrichtung berechnet. Das Programm wartet nun bis zu diesem Datum mittels der $time.sleep()$ Methode. Wenn das Datum mit dem kalkulierten Tag übereinstimmt, wird der Bewertungsalgorithmus kurz nach Mitternacht gestartet. Nach Durchlaufen des Bewertungsalgorithmus wird mittels der Alerter Klasse eine kurze Textnachricht an den Programmierer gesendet, worin Durchführungszeit des Bewertungsalgorithmus und Zeitpunkt der Marktöffnung vermerkt sind. Erfahrungsgemäß dauert die Berechnung der neuen Bewertung mehrere Stunden, abhängig von der Stabilität der Internetverbindung.

Nach dem Bewertungsalgorithmus wird das Portfolio konstruiert, jedoch erst, wenn der Markt auch tatsächlich offen ist, damit die verwendeten Preise so aktuell wie möglich sind. Weil die Konstruktion bloß rund 30 Sekunden dauert, ist es verkraftbar, diese erst unmittelbar nach Marktöffnung durchzuführen. Wann genau der Markt öffnet, kann folgendermaßen ermittelt werden:

```
clock = self.alp_api.get_clock()
if not clock.is_open:
    to_open = clock.next_open - clock.timestamp
    to_open = to_open.total_seconds()
    hours, rem = divmod(to_open, 3600)
    minutes, seconds = divmod(rem, 60)
    message = "{:0>2}:{:0>2}:{:05.2f}".format(
                                        int(hours),
                                        int(minutes),
                                        seconds)
    self.alerter.send_message("Market will open in " + message)
    time.sleep(to_open)
```

Obiger Code zeigt auch, wie eine Nachricht mittels der Alerter Klasse gesendet wird.

Nach der Konstruktion des Portfolios wird das Portfolio beim Broker angepasst. Dieses Prozedere ist bereits in Absatz 4.8 vorgestellt worden.

Nach Vollendung der Neuausrichtung wird das Quartal in der Datenbank als fertig ausgerichtet markiert:

```
self.cursor.execute(f"UPDATE dates SET rebalanced = 1 "\
                    f"WHERE quarter_year = '{quarter_year}'")
self.conn.commit()
```

Dann wird ein neues Quartal in die Datenbank eingefügt. Auf Q1_2021 folgt Q2_2021. Das Programm befindet sich bis zum neuen Termin im Ruhezustand und sodann wird wiederum eine neue Bewertung, Konstruktion und Neuausrichtung durchgeführt. Somit befindet sich das Programm in einer Endlosschleife, bei der rund alle drei Monate das gesamte Prozedere durchgeführt wird, sofern keine unvorhergesehenen Fehler passieren.

4.10 Umgang mit Errors

Dieses Kapitel beschreibt wie in der Implementierung mit unterschiedlichen Errors umgegangen wird.

Stromausfall

Als Erstes wird ein möglicher Stromausfall behandelt. Weil die Implementierung aus Kostengründen nicht in der Cloud, sondern auf einem üblichen privaten Rechner läuft, müssen die Folgen eventueller Stromausfälle minimiert werden. Dazu wird in den kritischen Prozessen, hauptsächlich bei der Bewertung, weil diese so lange dauert, nach jedem Zwischenschritt die Bewertung als .csv Datei zwischengespeichert. Somit kann bei einem eventuellen Stromausfall bei dem Schritt fortgesetzt werden, wo der Ausfall passierte. Dafür wird nach der Berechnung des Universums an Aktien, der Berechnung der Marktkapitalisierung, Volatilität, Nettoauszahlungsrendite und des Momentums das verwendete Pandas Dataframe als .csv Datei gespeichert. Bei Neustart des Programms wird zuerst versucht, eine lokale Bewertung in Form einer .csv Datei zu lesen, bevor mit der weiteren Berechnung fortgesetzt wird.

Die Implementierung läuft auf einem Linux Rechner, was den Einsatz von sogenannten Cron Jobs ermöglicht. Mittels eines Cron Jobs wird jedes Mal, wenn der Rechner gestartet wird, das Programm ausgeführt. Wie ein Cron Job auf einem Linux Rechner eingerichtet werden kann, um ein Programm direkt nach Neustart bzw. Start des Rechners automatisch auszuführen, kann in diesem Artikel [11] nachgelesen werden.

Netzwerkausfall

Ein häufigeres Problem, wenn das Programm auf einem privaten Rechner und Netzwerk läuft, sind Netzwerkprobleme. Das Internet wird bei der Durchführung der API requests benötigt, kann aber kurzfristig nicht verfügbar sein. Um damit umzugehen, müssen die jeweiligen Fehlercodes abgefangen werden und eine Logik implementiert werden, die nach einer gewissen Zeit erneut versucht, die vorher gescheiterten API Anfrage durchzuführen.

Um diese Funktionalität zu erfüllen, enthält jede API Anfrage folgenden Programmcode:

```
while True:
    try:
        async with session.get(url) as response:
            return await response.read()
    except aiohttp.ClientConnectionError:
        await asyncio.sleep(5)
    except aiohttp.ServerDisconnectedError as e:
        await asyncio.sleep(5)
```

```
    except asyncio.TimeoutError:
        await asyncio.sleep(5)
```

Selbiges gilt für synchrone API Anfragen, wobei je nach verwendeter API eigene Fehlercodes abgefangen werden müssen. Hier noch ein Beispiel für eine sichere Anfrage an die Yahoo Finance API:

```
while True:
    try:
        df = yf.Ticker(symbol).history(period="2y",
                                       interval="1d",
                                       actions=False)
        if df.empty or df is None:
            return np.NAN
        break
    except requests.ConnectionError:
        time.sleep(5)
    except requests.Timeout:
        time.sleep(5)
    except requests.exceptions.ChunkedEncodingError:
        time.sleep(5)
    # normally weird Yahoo Finance is Down Exception
    except RuntimeError as e:
        time.sleep(10)
```

Im Prinzip wird jede Anfrage in einer Endlosschleife so oft ausgeführt, bis eine zufriedenstellende Antwort erhalten wird. Bevor eine Anfrage aber nach Erfolglosigkeit wiederholt wird, wird das Programm für ein paar Sekunden unterbrochen. Wenn nämlich zu viele Anfragen an eine externe API in zu kurzer Zeit gesendet werden, dann wird man von dessen Server gedrosselt oder aber ignoriert. So schützen externe APIs sich vor Überlastung.

Im fertigen Programm wird natürlich der Fehler mittels dem Standard logging Modul von Python in einer .log Datei vermerkt, um etwaige Auffälligkeiten nachlesen zu können.

Bei allen in dieser Arbeit vorgestellten Codefragmente wird auf die Sicherung der API Anfragen gegen Netzwerkfehler zur besseren Lesbarkeit der Code Stücke verzichtet.

Interne Errors

Trotz aller Versuche, ein Programm stabil zu entwickeln, können dennoch unvorhergesehene Fehler auftreten. Um bei solchen Fehlern informiert zu werden, wird folgende Sicherheitsmaßnahme eingebaut: Der initiale Aufruf der Main.run() Methode wird mittels eines try-except Block umgeben. Dabei werden alle möglichen Exceptions gefangen, und im Fall dieser wird der Programmierer mittels Alerter benachrichtigt. Im Programmcode der Main Datei sieht das folgendermaßen aus:

```
class Main:
    ...
    ...
    def handle_error(self, exception: Exception) -> None:
        self.alerter.send_message("There has been an Exception!")
        self.alerter.send_message(str(exception))
        self.alerter.send_message(str(
                                    traceback.extract_tb(
                                        exception.__traceback__)
                                    )
                                )
```

```
            self.logger.exception("Unrecoverable Exception Occured")
            self.logger.error(str(exception))
            self.logger.error(traceback.extract_tb(exception.__traceback__))

if __name__ == '__main__':
    mainObject = Main()
    try:
        mainObject.run()
    except Exception as e:  # handle every possible Exception
        mainObject.handle_error(e)
```

Kapitel 5

Validierung und Diskussion

Die fertige Implementierung wurde in Echtzeit getestet und die Resultate sowie Probleme der Testphase werden in diesem Kapitel vorgestellt.

5.1 Resultate

Der folgende Absatz stellt die Resultate der Implementierung im getesteten Zeitraum (11.2.2021 bis 31.5.2021) vor.

Während des getesteten Zeitraums wurde eine initiale Konstruktion eines Portfolios am 11.2. und eine Neuausrichtung dieses Portfolios an das am 1.4.2021 berechnete Portfolio durchgeführt. Dazu wurde ein fiktives Depot mit einem Wert von 100 000 $ erstellt. Dessen Performance nach Investition dieser 100 000 $ wurde dann aufgezeichnet und in Folge mit dem Erfolg eines Referenzportfolios, das ausschließlich in den S&P 500 ETF „SPY“ investierte, verglichen. Grundsätzlich ist das Programm ohne größere Probleme gelaufen und hat wie erwartet funktioniert. Die initiale Konstruktion und Neuausrichtung des Portfolios sind problemlos durchgeführt und die Entwicklung des erstellten Portfolios ist erfolgreich aufgezeichnet worden.

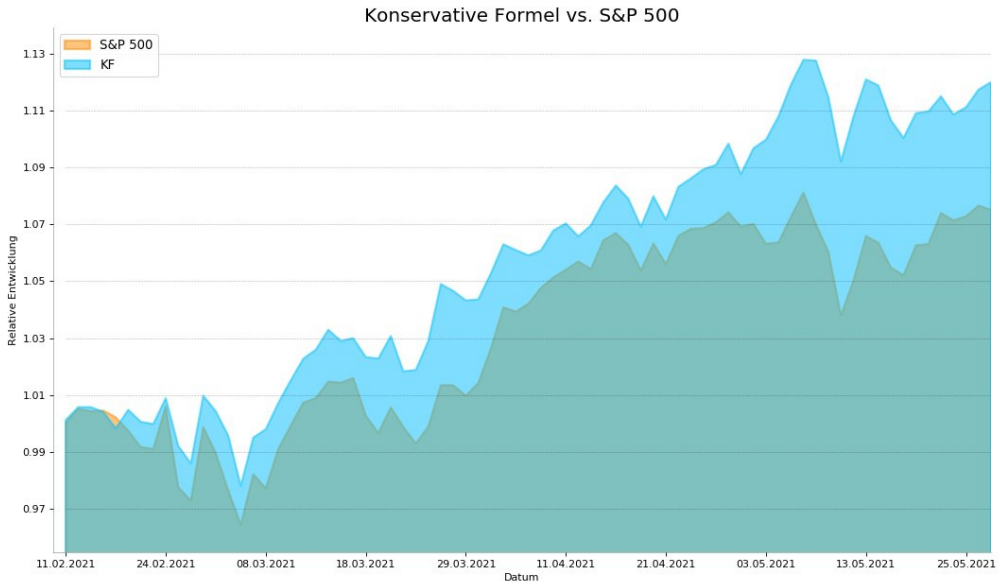

Abbildung 5.1: Entwicklung im getesteten Zeitraum

Die Resultate der Testphase sind in Abbildung 5.1 dargestellt. In Zahlen gefasst verbesserte sich die Implementierung der Konservativen Formel im betrachteten Zeitraum um 12%, während der S&P 500 ETF „SPY“ um 7,5% stieg. Somit ist die Implementierung besser als der Markt und validiert die tendenziell positive Fortsetzung der historischen Ergebnisse der Konservativen Formel nach Van Vliet und Blitz [2]. Die Entwicklung ist aber von ständigen Höhen und Tiefen geprägt, wobei die Gesamtentwicklung sehr stark mit der des S&P 500 korreliert, meist aber etwas besser ist und zu einem deutlich besseren Gesamtergebnis führt.

Die visualisierte Entwicklung berücksichtigt keine Dividendenausschüttungen. Im betrachteten Zeitraum wären im implementierten Portfolio 734,79 $ und vom S&P 500 ETF 327,17 $ ausgeschüttet. Dabei muss aber angemerkt werden, dass der S&P 500 ETF quartalsweise Dividende ausschüttet und im erstellten Portfolio die darin enthaltenen Aktien ihre Dividende an unterschiedlichen Stichtagen auszahlen. Alleine im ersten Quartal, vom 1.1. bis 1.4.2021, in dem aber nicht durchgehend investiert wurde, weil die Implementierung erst am 11.2. in die Testphase ging, hätte durch die Konservative Formel Dividende in Höhe von 365,98 $ erzielt werden können, wohingegen SPY bloß 327,17 $ an Dividende ausschüttete. Somit lässt sich mit relativer Sicherheit sagen, dass das Portfolio der Konservativen Formel auch im Punkt Dividendenrendite besser als SPY ist.

5.2 Probleme bei der Umsetzung

Die Umsetzung ist aber nicht immer reibungslos verlaufen und es galt einige Probleme zu lösen. Um auf möglichst viele Probleme so schnell wie möglich aufmerksam zu werden, ist in einer Testphase parallel zum eigentlichen Portfolio ein Testportfolio verwendet worden, das jeden Tag angepasst worden ist. D.h. Berechnung, Konstruktion und Ausrichtung des Portfolios sind täglich in diesem Testportfolio durchgeführt worden, um manuell auf allfällig auftretende Probleme zu testen. Die dabei entdeckten Fehler und deren Beurteilung werden in den folgenden Absätzen vorgestellt.

Probleme der Alpaca API

Dieser Absatz fasst die Probleme, die bei der Kommunikation mit der API des Brokers Alpaca Markets aufgetreten sind, zusammen.

- API liefert altes Datum für zuletzt durchgeführte Transaktionen:

  Dieses Problem trat am 2.3.2021 auf und war bei der Konstruktion des Portfolios störend, weil das Portfolio anhand falscher Preise konstruiert wurde. Ursache für dieses Problem war ein neues Feature von Alpaca Markets, nämlich ein eigenes Angebot von historischen Preisdaten. Durch die Veröffentlichung dieses Features wurden scheinbar andere Funktionen beeinträchtigt, unter anderem die Funktion, Daten bezüglich der zuletzt durchgeführten Transaktion zu bekommen.

- API lieferte falsche Daten bei Anfrage des Gesamtvermögens:

  Dieses Problem trat zwei Tage später, am 4.3.2021 auf. Dabei wurde eine inkorrekte Antwort erhalten was das gesamt verfügbare Vermögen betrifft. Störend war es in der Testphase wiederum bei der Konstruktion des Portfolios, wo von einem falschen Budget ausgegangen wurde.

- Splits werden ignoriert:

  Im Zuge der Testphase der Implementierung fanden vereinzelt sogenannte Aktiensplits statt. Aktiensplits bedeuten die Unterteilung einer Aktie in mehrere Teile, meist um den Preis einer Aktie zu verringern. Laut Alpaca Markets [21] werden Aktiensplits bewusst in Paper-Trading Accounts nicht berücksichtigt. Das führt aber zu einer Verfälschung der Ergebnisse, darum müssen diese Splits manuell eingerechnet werden.

- Dividenden werden ignoriert:

Dividenden werden in Paper-Trading Accounts ebenfalls vernachlässigt. Das Problem ist im Grunde analog zu dem der Aktiensplits. Einziger Unterschied ist jener, dass Dividenden keine so große Auswirkungen wie große Aktiensplits im Ergebnis haben.

Probleme mit der Yahoo Finanzen API

- Performance Probleme:

  Interessanterweise und ohne direkt nachvollziehbaren Grund wurden die API Anfragen über yfinance ab dem 12.3. merkbar langsamer. Die Anfragen dauerten in etwa doppelt so lange wie zuvor. Für die Zwecke dieser Implementierung ist Zeit noch kein Problem, weil es in erster Linie den Bewertungsalgorithmus verlangsamt, was egal ist, solange die Bewertung vor Marktöffnung fertiggestellt ist.

- Inkonsistenz der retournierten Daten:

  Vereinzelt waren Daten, die über die yfinance Bibliothek abgerufen wurden, nicht konsistent. Manchmal retournierte die API bei einer Spezifikation von einem Jahr bloß ein halbes Jahr an Daten. Dies konnte umgangen werden, indem immer zwei Jahre an Daten angefordert wurden. Weiters kann bei der Anfrage spezifiziert werden, ob man Daten zu Dividenden bzw. Aktiensplits beinhaltet haben möchte. Für diese Implementierung sollten diese Daten nicht enthalten sein. Bei der Neuausichtung des Portfolios am 1.4.2021 kam es ebenfalls zu Dividendenzahlungen bei diversen Aktien am selben Tag. Diese Aktien rutschten in der Bewertung raus, weil yfinance trotz klarer Spezifikation, dass diese Daten nicht enthalten sein sollten, Dividendendaten statt Preisdaten lieferte. Das führte zu einem Fehler im Programm, der zwar nicht zum Absturz führte, jedoch wurden dadurch diese Aktien für diese Neuausrichtung nicht weiter berücksichtigt.

Kapitel 6

Zusammenfassung

In dieser Arbeit wurden die Konservative Formel und die darin verwendeten Indikatoren detailliert vorgestellt. Darauf basierend ist ein Programm entwickelt worden, das nach dieser Formel automatisiert investiert. In einer Testphase vom 11.2.2021 bis zur Abgabe dieser Arbeit am 31.5.2021 ist die Entwicklung der Konservativen Formel aufgezeichnet und mit dem S&P 500 ETF „SPY“ verglichen worden. Die Ergebnisse zeigen, dass ein Portfolio mit dem im Rahmen dieser Arbeit erstellten Programm vollautomatisiert gemanagt werden kann. Dabei ist eine Rendite zu erwarten, die über dem marktüblichen Wert liegt.

6.1 Optimierungsvorschläge

Die folgenden Absätze stellen einige Verbesserungsmöglichkeiten zur Umsetzung der Konservativen Formel vor.

Teilaktien

Teilaktien ermöglichen den Kauf von Aktienanteilen. Dadurch fällt die Verpflichtung weg, gesamte Stücke an Aktien zu kaufen. Nützlich sind Teilaktien für die Umsetzung der Konservativen Formel bei der Berechnung und Ausrichtung des Portfolios. Teilaktien würden die Gleichverteilung des Vermögens auf die 100 Aktien trivialisieren. Der gesamte Prozess der Portfoliokonstruktion würde überflüssig werden und eine perfekte Gleichverteilung wäre einfach zu erreichen. Alpaca Markets führte am 4.3.2021 Teilaktien ein [21]. In diesem Angebot sind jedoch nicht alle Aktien zum anteiligen Kaufen verfügbar. Weiters steckt das Feature noch in Kinderschuhen, weshalb auf die Adaption der Implementierung verzichtet worden ist.

APIs

Um konsistente und verlässlichere Daten zu bekommen, wäre ein Wechsel der verwendeten APIs sinnvoll. Da aber qualitativ hochwertige Anbieter wie z.B. Polygon [24] relativ kostspielig sind, ist mit den bekannten APIs gearbeitet und versucht worden, dessen Fehler auszumerzen.

6.2 Ausblick

Aufgrund Zeit und Umfang dieser Arbeit sind noch einige Punkte offen, an die angeknüpft werden kann. Offene Verbesserungsmöglichkeiten aus technischer Sicht sind bereits vorgestellt worden.

Eine nähere Betrachtung, besonders bezüglich der Wirtschaftlichkeit der implementierten Strategie, ist vonnöten. Sind die präsentierten Ergebnisse auch nach Berücksichtigung von Steuern und Transaktionskosten noch immer besser als ein ETF wie der „SPY“ , der den S&P 500 Index abbildet? Bei der einzigen Neuausrichtung, die im Rahmen der Testphase vorgekommen ist, sind bereits 30 Aktien aus

dem Portfolio gefallen, das entspricht 30% des abgebildeten Portfolios. Eine Neuausrichtung wird planmäßig viermal im Jahr durchgeführt. Bei diesen Verkäufen sind Gebühren und Kapitalertragsteuern zu bezahlen, wohingegen diese bei einer Investition in einen ETF, der über längere Frist gehalten wird, von untergeordneter Bedeutung sind. Nichtsdestotrotz wäre es sinnvoll, die Berechnung der abzuführenden Steuern für getätigte Verkäufe und erhaltene Dividenden sowie realistische Gebühren im Programm zu integrieren. Dadurch könnte eine aus wirtschaftlicher Sicht genauere Prognose abgegeben werden bzw. die Resultate wären näher an der Realität.

Weiters wäre die Weiterentwicklung des Programms ein interessantes Ziel. Insbesondere die Entwicklung einer Webapplikation, die Benutzer:innen ermöglicht, ihr Geld nach dieser Strategie zu veranlagen.

Literaturverzeichnis

[1] Claudiu Tiberiu Albulescu. Covid-19 and the united states financial markets' volatility. *Finance Research Letters*, 38:101699, 2021.

[2] David Blitz and Pim van Vliet. Ipr: The conservative formula: Quantitative investing made easy. *The Journal of Portfolio Management*, 44(7):24–38, 2018.

[3] David Blitz, Pim van Vliet, and Guido Baltussen. The volatility effect revisited. *The Journal of Portfolio Management*, 2019.

[4] Börsenlexikon. `https://www.boerse.de/boersenlexikon/Marktkapitalisierung`. Accessed: 2021-04-06.

[5] Börsenlexikon. `https://www.boerse.de/boersenlexikon/Momentum`. Accessed: 2021-04-06.

[6] Börsenlexikon. `https://www.boerse.de/boersenlexikon/Volatilitaet`. Accessed: 2021-04-06.

[7] US Inflation Calculator. `https://trading-treff.de/wissen/aktienrueckkauf-wieso-unternehmen-aktien-zurueckkaufen`. Accessed: 2021-04-06.

[8] YAN LIU CAMPBELL R. HARVEY. Evaluating trading strategies. *The Journal of Portfolio Management*, 40:108–118, 2014.

[9] IEX Cloud. `https://iexcloud.io/docs/api/`. Accessed: 2021-04-28.

[10] ComputerWeekly. `https://www.computerweekly.com/de/definition/Programmierschnittstelle-API`. Accessed: 2021-04-29.

[11] Computerwoche. `https://www.tecchannel.de/a/cron-linux-tools-und-befehle-beim-boot-ausfuehren,3205434`. Accessed: 2021-05-03.

[12] Franz Nestler Daniel Mohr. `https://www.faz.net/aktuell/finanzen/deutschland-im-aktien-boom-wertpapier-kauf-steigt-waehrend-corona-17224396.html`. Accessed: 2021-04-06.

[13] Guru Focus. `https://www.gurufocus.com/term/buyback_yield/aapl/`. Accessed: 2021-04-06.

[14] Deutsche Terminbörse GmbH. *Risiken*, pages 87–90. Gabler Verlag, Wiesbaden, 1989.

[15] William N. Goetzmann and Alok Kumar. Equity Portfolio Diversification*. *Review of Finance*, 12(3):433–463, 03 2008.

[16] gurufocus. `https://www.gurufocus.com/`. Accessed: 2021-04-28.

[17] Christians Homepage. https://www.christian-luetgens.de/homematic/telegram/botfather/Chat-Bot.htm. Accessed: 2021-05-02.

[18] Kewei Hou, Chen Xue, and Lu Zhang. Replicating Anomalies. *The Review of Financial Studies*, 33(5):2019–2133, 12 2018.

[19] InfraSoft. https://www.infrasoft.at/images/downloads/uebersicht_der_uml_diagramme.pdf. Accessed: 2021-04-27.

[20] Investopedia. https://www.investopedia.com/terms/d/dividendyield.asp. Accessed: 2021-04-06.

[21] Alpaca Markets. https://alpaca.markets/. Accessed: 2021-04-28.

[22] NumPy. https://numpy.org/. Accessed: 2021-04-28.

[23] André F. Perold. The capital asset pricing model. *Journal of Economic Perspectives*, 18(3):3–24, September 2004.

[24] Polygon. https://polygon.io/pricing. Accessed: 2021-05-03.

[25] Pypi. https://pypi.org/project/aiohttp/. Accessed: 2021-04-28.

[26] Pypi. https://pypi.org/project/alpaca-trade-api/. Accessed: 2021-04-28.

[27] Pypi. https://pypi.org/project/asyncio/. Accessed: 2021-04-28.

[28] Pypi. https://pypi.org/project/beautifulsoup4/. Accessed: 2021-04-28.

[29] Pypi. https://pypi.org/project/pandas/. Accessed: 2021-04-28.

[30] Pypi. https://pypi.org/project/yfinance/. Accessed: 2021-04-28.

[31] PYPL. https://pypl.github.io/PYPL.html. Accessed: 2021-04-28.

[32] python.org. https://docs.python.org/3/faq/general.html#what-is-python-good-for. Accessed: 2021-04-27.

[33] Reddit. https://www.reddit.com/r/wallstreetbets/. Accessed: 2021-04-06.

[34] Subreddit Stats. https://subredditstats.com/r/wallstreetbets. Accessed: 2021-04-06.

[35] Trading Treff. https://www.usinflationcalculator.com/. Accessed: 2021-04-07.

[36] YCharts. https://ycharts.com/glossary/terms/net_payout_yield_ttm. Accessed: 2021-05-23.