

Das Projekt

Dieses Dokument dient dazu den aktuell stand des SWP- Projekts außenstehenden Personen näherzubringen. Dabei wird die eine Gliederung von

1. Verwendung/ Verwaltung API ->
2. Verwendung/ Verwaltung der Datenbank ->
3. Aufarbeitung der Daten für Basisprojekt ->
4. Basisprojekt aktueller Stand (nicht kombiniert)

2.tes Halbbahr:

1. Verwaltungstechnische Änderungen der Daten
2. Berechnung von Kennzahlen
3. Verbesserung der Visualisierung (nichts mehr sollte von der Konsole aus geschehen)

Inhalt

Das Projekt	1
Verwendung/ Verwaltung API (Download.Java)	2
Hier erfolgt die Dokumentation	3
Ausführung des Programmes in der Main	3
Interaktion mit dem Button	4
Work() Methode.....	4
Erstellung eines Datenbankschemas.....	5
Download der Daten	5
Daten in die Datenbank Laden	6
Der Datenbank manager	7
Errechnung der besten Kaufs bzw Verkaufszeiten.....	9
Umsetzung der Strategie.....	10
Visualisierung des Testergebnisses	10
Layout der Datenvisualisierung	11
Historie der Strategie (Areachart)	11
Bester Zeitpunkt für kauf und verkauf (BarChart).....	11
Historie der durchgeführten Trades (Tabelle).....	12
Textfeld mit wichtigen Kennzahlen	14
Zurück Button	15
Beispiel TLT (Do kauf Fr Verkauf)	16
Beispiel GLD (Fr kauf Mo Verkauf)	17

Eingehalten. Das Pflichtenheft für das Projekt finden Sie unter folgendem Link:

https://github.com/flostaudacher/ETF_TLT_Backtesting/blob/master/Mein%20Pflichtenheft.pdf

Verwendung/ Verwaltung API (Download.Java)

Als API wird Alpha Vantage herbeigezogen. Dabei handelt es sich um eine API zur Beziehung von Daten aus dem Aktienmarkt. In unserem Fall verwenden wir die Funktion Extended History von Alpha Vantage.

Intraday (Extended History)

This API returns historical intraday time series for the trailing 2 years, covering over 2 million data points per ticker. The intraday data is computed directly from the Securities Information Processor (SIP) market-aggregated data feed. You can query both raw (as-traded) and split/dividend-adjusted intraday data from this endpoint. Common use cases for this API include data visualization, trading simulation/backtesting, and machine learning and deep learning applications with a longer horizon.

API Parameters

■ Required: `function`

The time series of your choice. In this case, `function=TIME_SERIES_INTRADAY_EXTENDED`

■ Required: `symbol`

The name of the equity of your choice. For example: `symbol=IBM`

■ Required: `interval`

Time interval between two consecutive data points in the time series. The following values are supported: `1min`, `5min`, `15min`, `30min`, `60min`

■ Required: `slice`

Two years of minute-level intraday data contains over 2 million data points, which can take up to Gigabytes of memory. To ensure optimal API response speed, the trailing 2 years of intraday data is evenly divided into 24 "slices" - `year1month1`, `year1month2`, `year1month3`, ..., `year1month11`, `year1month12`, `year2month1`, `year2month2`, `year2month3`, ..., `year2month11`, `year2month12`. Each slice is a 30-day window, with `year1month1` being the most recent and `year2month12` being the farthest from today. By default, `slice=year1month1`.

■ Optional: `adjusted`

By default, `adjusted=true` and the output time series is adjusted by historical split and dividend events. Set `adjusted=false` to query raw (as-traded) intraday values.

■ Required: `apikey`

Hier bekommen wir Daten zu jedem Tag zu einer Vielzahl an Zeitpunkten zu jeder Aktie. Erhalten wird eine CSV Datei zu jedem Monat (bis zu 2 Jahren). Um die gewollten Daten zu beziehen muss eine URL aufgerufen werden. Da wir die komplette Historie beziehen wollen müssen wir also 24 verschiedene URLs aufrufen. Das heißt wir müssen uns verschiedene URLs in Java zusammensetzen die den benötigten URLs entsprechen. Hier benötigen wir also mehrere Methoden:

Hier erfolgt die Dokumentation

Die Dokumentation des Projektes erfolgt in der Reihenfolge in der gewisse Teile des Projektes auftreten.

Ausführung des Programmes in der Main

In der Main des Projektes wird in die Main der Visualisierung verwiesen, diese Main der Visualisierung, welche von Applikation erbt, wird wiederum auf die Methode launch verwiesen.

```
public static void main(String[] args) throws ClassNotFoundException, SQLException, ParseException {
    Datavizualise.main(args);
}
```

```
public static void main(String[] args) {
    launch(args);
}
```

Hier wird nun die von JavaFx vorgeschriebene Start Methode ausgeführt, welche die init Methode initialisiert.

```
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    init(primaryStage);
}
```

```
private void init(Stage primaryStage) {
    /**
     * Alles von hier bis zum nächsten komment behandelt den Backtest
     */
    st = primaryStage;
    st.setMaximized(true);
    sce1 = new Scene(layout1, 1900, 1000);

    /**
     * hier bis zum nächsten komment ist das auswahlmenü
     */
    Button DisplayBackTest = new Button("Commit the Backtest");
    GridPane.setConstraints(DisplayBackTest, 10, 10);
    GridPane layout2 = new GridPane();
    layout2.setPadding(new Insets(10, 10, 10, 10));
    layout2.setVgap(5);
    layout2.setHgap(5);

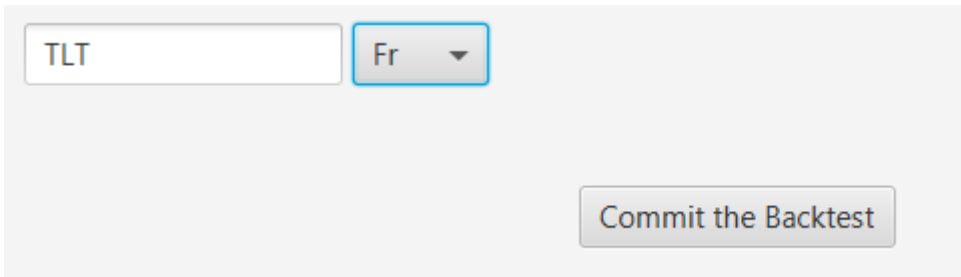
    final TextField Symbol = new TextField();
    Symbol.setPromptText("Enter the Stocksymbol you want to backtest");
    Symbol.setPrefColumnCount(10);
    Symbol.getText();
    GridPane.setConstraints(Symbol, 0, 0);

    ChoiceBox<String> dayOfBuy = new ChoiceBox<String>();
    dayOfBuy.getItems().addAll("Mo", "Di", "Mi", "Do", "Fr");
    dayOfBuy.setValue("Fr");
    GridPane.setConstraints(dayOfBuy, 1, 0);
    DisplayBackTest.setOnAction(e -> {
        try {
            getChoice(dayOfBuy, Symbol);
        } catch (ClassNotFoundException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (ParseException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    });
    layout2.getChildren().addAll(Symbol, dayOfBuy, DisplayBackTest);
    sce2 = new Scene(layout2, 500, 500);

    st.setScene(sce2);
    st.setTitle("Untersuchung der ausgewählten Aktie");
    st.show();
}
```

In diesem Teil des Projektes geschehen 2 Sachen. Es werden hier 2 Layouts erstellt, eines für die Auswahl eines Symbols und des Kauftages, und das andere für die Visualisierung des Backtests. Im

Layout für die Auswahlmöglichkeit wird auch ein Button erstellt bei dessen Interaktion einige Dinge passieren.

The image shows a Java Swing window with a light gray background. On the left, there is a text field containing the text 'TLT'. To its right is a dropdown menu with a blue border and a small downward arrow, currently displaying 'Fr'. Below these two elements, centered horizontally, is a button with a gray gradient and the text 'Commit the Backtest'.

Interaktion mit dem Button

Bei Interaktion mit dem Button wird die Methode `getChoice` aufgerufen.

```
private Object getChoice(ChoiceBox<String> dayOfBuy, TextField symbol) throws ClassNotFoundException, SQLException, ParseException {
    String ChosenDayOfBuy = dayOfBuy.getValue();
    String ChosenSymbol = symbol.getText();
    umsetzungStrategie.setDayofBuy(ChosenDayOfBuy);
    download.stockSymbol = ChosenSymbol;
    main.work();
    areaChart = visualization.areaChart();
    barchartMin = barChart.BarChart(1);
    barchartMax = barChart.BarChart(2);
    tabelle = table.tableCreat();
    kennzahlen = textfeld.showValues();
    GoBackToOptionsMenu.setOnAction(e -> goBack());
    layout1.setConstraints(areaChart, 0, 0);
    layout1.setConstraints(barchartMin, 2, 0);
    layout1.setConstraints(barchartMax, 1, 0);
    layout1.setConstraints(tabelle, 1, 1);
    layout1.setConstraints(kennzahlen, 0, 1);
    layout1.setConstraints(GoBackToOptionsMenu, 2, 1);
    layout1.getChildren().addAll(areaChart, barchartMin, barchartMax, tabelle, kennzahlen, GoBackToOptionsMenu);

    layout1.setVgap(100);
    layout1.setHgap(100);
    st.setScene(scel);
    return null;
}
```

Hier werden einige Werte gesetzt welche für die Visualisierung des Backtests eine Rolle spielen , außerdem wird die Methode `work()` aufgerufen, welche den Hauptteil des Projektes übernimmt.

Work() Methode

```
public static void work() throws ClassNotFoundException, SQLException, ParseException{
    download.deleteCurrentFiles();
    DBmanager db = new DBmanager();
    Connection con = db.getConnection();
    download.deleteCurrentFiles();
    if (db.stockAlreadyExistsInAktienList(con, download.stockSymbol) == false) {
        download.download();
        combination.VonCSVinDatenbank(con, db);
    }
    minimalValueTimeList = db.getTimeofMinOrMaxofDay(con, db.getIDfromStock(con, download.stockSymbol), 1); // 1 für min 2 für max
    String BuyTime = getBuyTime(1, minimalValueTimeList); // für min
    maximalValueTimeList = db.getTimeofMinOrMaxofDay(con, db.getIDfromStock(con, download.stockSymbol), 2);
    String SellTime = getBuyTime(2, maximalValueTimeList); // für max
    textfeld.setOptimaleZeiten(BuyTime, SellTime);
    ArrayList<aktie> eintrage = db.readStockValues(con, db.getIDfromStock(con, download.stockSymbol));
    umsetzungStrategie.handel(eintrage, BuyTime, SellTime);
}
```

In dieser Methode geschehen einige Sachen:

Erstellung eines Datenbankschemas

```
use aktien;  
create table aktienListe (  
  ID int (10) auto_increment primary key,  
  Symbol varchar(50)  
);  
create table aktie(  
  Symbol int (50) not null,  
  Datum date not null,  
  Zeitpunkt time ,  
  StockValue float (50) not null,  
  Weekday varchar (50) not null,  
  Primary Key(Symbol, Datum, Zeitpunkt)  
);
```

Als erstes musste ein Datenbank-Schema erstellt werden in dem die Daten gespeichert werden können

Download der Daten

Als erstes werden bereits heruntergeladene Dateien gelöscht, hierfür musste ein Ordner angelegt werden in dem die Daten auffindbar sind.

```
public static void deleteCurrentFiles() {  
    File location= new File ("C:\\Users\\flost\\eclipse-workspace\\ETF_BACKTESTING_BONUS\\data");  
    File[] files = location.listFiles();  
    for (File file : files) {  
        if (!file.delete()) {  
            System.out.println("failed to delete");  
        }  
    }  
}
```

In dieser Methode werden alle files aus der angegebenen Location gelöscht. Nun können die Daten von der API bezogen werden.

```
public static void download() {  
    fillMonthsArray();  
    String symbol = stockSymbol;  
    int zeit = 1;  
    for (int i = 0; i < PARTS_OF_API; i++) {  
        url[i] = createurl(i,symbol);  
        fileName[i] = fileNameBase + symbol + "_" + months[i] + ".csv";  
        try (BufferedInputStream in = new BufferedInputStream(new URL(url[i]).openStream());  
            FileOutputStream fileOutputStream = new FileOutputStream(fileName[i])) {  
            byte dataBuffer[] = new byte[1024];  
            int bytesRead;  
            while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {  
                fileOutputStream.write(dataBuffer, 0, bytesRead);  
            }  
        } catch (IOException e) {  
            System.out.println("");  
        }  
        if (i + 1 == 5*zeit) {  
            try{  
                zeit++;  
                Thread.sleep(60000);  
            }catch(InterruptedException e){  
                e.printStackTrace();  
            }  
        }  
    }  
    zeit = 0;  
}
```

In diesem Teil werden die Files in das filesystem heruntergeladen, dabei muss jedoch die „Downloadbegrenzung“ der Gratis Version der API beachtet werden. Der download setzt sich aus 24 verschiedenen CSV Dateien zusammen, welche alle durch eine Unterschiedliche API-Route

heruntergeladen werden. Diese verschiedenen URL müssen separat gebildet werden. Dies passiert durch folgende 2 Methoden:

Daten in die Datenbank Laden

Um die Daten in die Datenbank zu laden, werden zuerst alle Daten der CSV in eine Liste geladen. Dabei wird das Zeitzoneproblem gelöst.

- Das Zeitzoneproblem wird behandelt in dem die Zeitzone die die API liefert Eastern Time zur Central Europe Time zurückgewandelt wird.

```
static void VonCSVInDatenbank(Connection con, DBmanager db) throws SQLException, ParseException {
    if (db.stockAlreadyExistsInAktienList(con, download.stockSymbol) == false){
        aktienListe l = new aktienListe (1, download.stockSymbol);
        db.newAktieInAktienListe(con, l);
    }
    for (int i = download.fileName.length; i >= 0 ; i = i -1 ) {
        Scanner sc= null;
        try
        {
            sc= new Scanner (new BufferedReader(new FileReader(download.fileName[i])));
            sc.nextLine();
            while (sc.hasNextLine()) {
                String temp[] = new String[7];
                Inputline = sc.nextLine();
                Inputline = Inputline.replaceAll(" ", ",");
                String[] inArr = Inputline.split(",");
                for (int x = 0; x < inArr.length; x++) {
                    temp[x%(temp.length)]=inArr[x];
                }
                Rowc++;
                float val = Float.parseFloat(temp[4]);
                String s = aktie.correctTheDate(temp[0], temp[1]);
                s = s.replaceAll("Z",":00");
                String [] ar = s.split("T");
                String d = ar[0];
                String t = ar[1];
                aktie a = new aktie(db.getIDfromStock(con, download.stockSymbol), d, combination.toSQLTime(t), val);
                dataList.add(a);
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    pushCSVToDatabase(con,db);
}
```

Das zurückwandeln der Zeit erfolgt durch folgende Methode:

```
public static String correctTheDate(String falseDate, String falsetime) throws ParseException {
    String s = falseDate + " "+falsetime;
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    LocalDateTime parse = LocalDateTime.parse(s, formatter);
    ZoneId of = ZoneId.of("America/New_York");
    String trueDateUnedited= "" + parse.atZone(of).withZoneSameInstant(ZoneOffset.UTC);
    return trueDateUnedited;
}
```

Nachdem die Daten in eine ArrayList geladen wurden, wird nun die Methode pushCSVToDatabase() ausgeführt:

```
static void pushCSVToDatabase(Connection con, DBmanager db) throws SQLException, ParseException {
    int Passcounter = 0;
    int Errorcounter = 0;
    for (int Rowc = 0; Rowc < dataList.size(); Rowc++) {
        boolean okay = db.stockRowAlreadyExists(con,dataList.get(Rowc));
        if (okay == true) {
            if (dataList.get(Rowc).getTimestamp().before(combination.toSQLTime("16:00:00")) && dataList.get(Rowc).getTimestamp().after(combination.toSQLTime("08:00:00"))) {
                aktie temp = dataList.get(Rowc);
                db.saveNewSpecificStockValue(con,temp);
                Passcounter++;
            }
        }
        if (okay==false) {
            System.out.println(dataList.get(Rowc));
            Errorcounter++;
        }
    }
    System.out.println("Es wurden " + Passcounter + " neue Einträge eingetragen " + Errorcounter + " Einträge waren schon vorhanden");
}
```

In dieser Methode wird dann überprüft ob der Eintrag bereits in der Tabelle AktienListe vorhanden ist, wenn nicht wird dieser eingefügt, wenn schon geht es weiter und die noch nicht vorhandenen Einträge werden hinzugefügt.

Der Datenbank manager

Im Datenbankmanager findet man die wichtigsten Methoden für die Verbindung zur Datenbank.

SaveNewSpecific StockValue	<pre> public static void saveNewSpecificStockValue(Connection con, aktie stock) throws SQLException { String sql = "insert into aktie (Symbol,Datum,Zeitpunkt,StockValue,Weekday) values (?,?,,?,?)"; PreparedStatement stm = null; try { stm = con.prepareStatement(sql); Date date = Date.valueOf(stock.getDate()); // hier könnte stm.setInt(1, stock.getSymbol()); stm.setDate(2, date); stm.setTime(3, stock.getTimestamp()); stm.setFloat(4, stock.getValue()); stm.setString(5, stock.getWeekday()); stm.executeUpdate(); } finally { if (stm != null) stm.close(); } } </pre>
readStockValues	<pre> public ArrayList<aktie> readStockValues (Connection con, int Symbol) throws SQLException, ParseException{ PreparedStatement stm = null; ResultSet rs = null; ArrayList<aktie> result = new ArrayList<aktie>(); try { String sql = "select Symbol, Datum, Zeitpunkt, StockValue, Weekday from aktie where Symbol = ? order by Datum ASC"; stm = con.prepareStatement(sql); stm.setInt(1, Symbol); rs = stm.executeQuery(); while(rs.next()) { int id = rs.getInt(1); String Datum = rs.getString(2); Time Zeitpunkt = rs.getTime(3); Zeitpunkt.setHours(Zeitpunkt.getHours()-1); // lehren fragen wann es eine stunde vorsetzt Float StockValue = rs.getFloat(4); String Weekday = rs.getString(5); aktie a = new aktie(Symbol,Datum,Zeitpunkt,StockValue); a.setWeekday(Weekday); result.add(a); } } finally { if(stm != null) stm.close(); } System.out.println(result.size()); return result; } </pre>
stockRowAlreadyExist s	<pre> public boolean stockRowAlreadyExists(Connection con, aktie a) throws SQLException{ boolean result = false; PreparedStatement stm = null; ResultSet rs = null; try { String sql = "select count(*) from aktie where (Symbol = ? and Datum = ? and Zeitpunkt = ? and StockValue = ? and Weekday = ?)"; stm = con.prepareStatement(sql); stm.setInt(1, a.getSymbol()); stm.setString(2, a.getDate()); stm.setString(3, a.getTimestamp().toString()); stm.setFloat(4, a.getValue()); stm.setString(5, a.getWeekday()); rs = stm.executeQuery(); if (rs.next()) { int anzahl = rs.getInt(1); result = anzahl == 1; } } finally { if (stm != null) { stm.close(); } } return !result; } </pre>
newAktieninAktienlist e	<pre> public void newAktieInAktienListe(Connection con, aktienListe l) throws SQLException { String sql = "insert into aktienListe (symbol) values (?)"; PreparedStatement stm = null; try { stm = con.prepareStatement(sql); stm.setString(1, l.getSymbol()); stm.executeUpdate(); } finally { if (stm != null) stm.close(); } } </pre>

stockAlreadyexists inaktienLsite	<pre> public boolean stockAlreadyExistsInAktienList (Connection con, String symbol) throws SQLException { boolean result = false; PreparedStatement stm = null; ResultSet rs = null; try { String sql = "select count(*) from aktienListe where symbol = ?"; stm = con.prepareStatement(sql); stm.setString(1, symbol); rs = stm.executeQuery(); if (rs.next()) { int anzahl = rs.getInt(1); result = anzahl == 1; } } finally { if (stm != null) { stm.close(); } } return result; } </pre>
getIdfromStock	<pre> public int getIDfromStock(Connection con, String symbol) throws SQLException { PreparedStatement stm = null; ResultSet rs = null; int result = 0; try { String sql = "select ID from aktienListe where Symbol = ?"; stm = con.prepareStatement(sql); stm.setString(1, symbol); rs = stm.executeQuery(); while(rs.next()) { result = rs.getInt(1); } } finally { if(stm != null) stm.close(); } return result; } </pre>
getTimemin OrmaxFromDay	<pre> public ArrayList<String> getTimesOfMinOrMaxOfDay(Connection con, int symbol, int i) throws SQLException { PreparedStatement stm = null; String sql = ""; if (i == 1) { sql = "Select A.Datum, A.Zeitpunkt, B.minVal from aktie as A inner join (select Datum,min(StockValue) as minVal from aktie where (Symbol = ?) group by Datum) as B on A.StockValue = B.minVal and A.Datum = B.Datum"; } else { sql = "Select A.Datum, A.Zeitpunkt, B.maxVal from aktie as A inner join (select Datum,max(StockValue) as maxVal from aktie where (Symbol = ?) group by Datum) as B on A.StockValue = B.maxVal and A.Datum = B.Datum"; } ResultSet rs = null; String zeitpunkt = ""; String minStockValue = ""; ArrayList<String> result = new ArrayList<String> (); try { stm = con.prepareStatement(sql); stm.setInt(1, symbol); rs = stm.executeQuery(); while(rs.next()) { Date Datum = rs.getDate(); zeitpunkt = rs.getString(2); minStockValue = rs.getString(3); result.add(zeitpunkt); } } finally { if(stm != null) stm.close(); } return result; } </pre>
Release Connection	<pre> public void releaseConnection (Connection con) throws SQLException { if (con != null) con.close(); } </pre>

Errechnung der besten Kaufs bzw Verkaufszeiten

Um die bestmöglichen Zeiten zu erhalten wird anhand eines SQL statements zurückgegeben zu welcher Uhrzeit der minimal oder der maximal Wert am Jeweiligen Tag auftritt. Die Rückgabe wird dann in eine ArrayList gespeichert.

```
minimalValueTimeList = db.getTimeofMinOrMaxOfDay(con,db.getIDfromStock(con, download.stockSymbol), 1); // 1 für min 2 für max
String BuyTime = getBuyTime(1,minimalValueTimeList); // für min
maximalValueTimeList = db.getTimeofMinOrMaxOfDay(con,db.getIDfromStock(con, download.stockSymbol), 2);
String SellTime = getBuyTime(2,maximalValueTimeList); // für max
textfield.setOptimaleZeiten(BuyTime,SellTime);
```

In folgender Methode werden anhand einer Hashmap ausgewertet welcher Wert die größte Häufigkeit besitzt und somit der beste Zeitpunkt für einen Kauf/ Verkauf ist.

```
static String getBuyTime(int option, ArrayList<String> list) {
    // TODO Auto-generated method stub
    ArrayList<frequenzy> FList = new ArrayList<frequenzy>();
    for (String str : list) {
        Integer i = frequenzyMap.get( str);
        if ( i == null) {
            i = new Integer( 1);
        } else {
            i = new Integer( i.intValue()+1);
        }
        frequenzyMap.put( str, i);
    }
    for (String key : frequenzyMap.keySet()) {
        frequenzy f = new frequenzy(key,frequenzyMap.get(key));
        FList.add(f);
    }
    bestTime = sortForBestTime(FList);
    if (option == 1) {
        return bestTime.getTime();
    } else {
        return bestTime.getTime();
    }
}

private static frequenzy sortForBestTime(ArrayList<frequenzy> fList) {
    frequenzy temp;
    if (fList.size() > 1)
    {
        for (int x = 0; x < fList.size(); x++)
        {
            for (int i=0; i < fList.size() - x - 1; i++) {
                if (fList.get(i).compareTo(fList.get(i+1)) > 0)
                {
                    temp = fList.get(i);
                    fList.set(i,fList.get(i+1) );
                    fList.set(i+1, temp);
                }
            }
        }
    }
    return fList.get(fList.size()-1);
}
```

Umsetzung der Strategie

In dieser Klasse erfolgt der Trade des aktuell ausgewählten Symbols am ausgewählten Tag zum bestmöglichen Kaufs oder Verkaufszeitpunkt.

```
public static String handel(ArrayList<aktie> handel, String buyTime, String sellTime) {
    String dayOfBuy = null;
    boolean KaufExcepted = false;
    boolean VerkaufExcepted = false;
    int zaehlerVerkauf = 0;
    int zaehlerKauf = 0;
    int kaufe = 0;
    for (int Rowc = 0; Rowc < handel.size(); Rowc++) {
        KaufExcepted = handel.get(Rowc).getWeekday().equals(s) && toString(handel.get(Rowc).getTimestamp()).equals(buyTime);
        if (KaufExcepted == true && kaufe != 1) {
            dayOfBuy = handel.get(Rowc).getWeekday();
            buyStock(handel.get(Rowc));
            zaehlerKauf++;
            kaufe = 1;
        }
        VerkaufExcepted = (!handel.get(Rowc).getWeekday().equals(dayOfBuy)) && toString(handel.get(Rowc).getTimestamp()).equals(sellTime);
        if (VerkaufExcepted == true && kaufe == 1) {
            SellStock(handel.get(Rowc));
            dayOfBuy = null;
            zaehlerVerkauf++;
            kaufe = 0;
            textfeld.calcProfitFactor(stückzahl,buyValue,sellValue);
            Trade t = new Trade(handel.get(Rowc).getDate(),depotValue,stückzahl,buyValue,sellValue,buyValue < sellValue);
            History.add(t);
            stückzahl = 0;
        }
    }
    return s;
}
```

Kauf und Verkauf der Positionen erfolgt durch folgende zwei Methoden:

```
static void buyStock(aktie a) {
    double ValueBuy = a.getValue(); // ohne gebühren
    //double closingValueBuy = (Double.parseDouble(acc))*1.01; // mit gebühren.
    stückzahl = (int) ((depotValue - 5) / (ValueBuy)); // formel wo die gebühren hin gehören
    buyValue = ValueBuy;
    depotValue = depotValue - stückzahl * (ValueBuy);
}

static void SellStock(aktie a) {
    double ValueSell = a.getValue();
    depotValue = depotValue + stückzahl * (ValueSell);
    sellValue = ValueSell;
}
```

Visualisierung des Testergebnisses

All diese Dinge werden ab dem Betätigen des Buttons durchgeführt, außerdem findet auch die Visualisierung der Daten statt und die einzelnen Elemente werden in das Layout geladen.

```
private Object getChoice(ChoiceBox<String> dayOfBuy, TextField symbol) throws ClassNotFoundException, SQLException, ParseException {
    String ChosenDayOfBuy = dayOfBuy.getValue();
    String ChosenSymbol = symbol.getText();
    umsetzungStrategie.setDayofBuy(ChosenDayOfBuy);
    download.stockSymbol = ChosenSymbol;
    main.work();
    areaChart = vizualization.areachart();
    barchartMin = barChart.BarChart(1);
    barchartMax = barChart.BarChart(2);
    tabelle = table.tableCreat();
    kennzahlen = textfeld.showValues();
    GoBackToOptionMenu.setOnAction(e -> goBack());
    layout1.setConstraints(areaChart,0,0);
    layout1.setConstraints(barchartMin,2,0);
    layout1.setConstraints(barchartMax,1,0);
    layout1.setConstraints(tabelle,1,1);
    layout1.setConstraints(kennzahlen,0,1);
    layout1.setConstraints(GoBackToOptionMenu,2,1);
    layout1.getChildren().addAll(areaChart,barchartMin,barchartMax,tabelle,kennzahlen,GoBackToOptionMenu);

    layout1.setVgap(100);
    layout1.setHgap(100);
    st.setScene(scel);
    return null;
}
```

Layout der Datenvisualisierung

Die Visualisierung des Backtests ist in einem Gridpane angeordnet und besteht aus den folgenden Inhalten:

Historie der Strategie (Areachart)

Anhand einer Trading-Historie sollte der Erfolg einer Strategie für das ausgewählte Symbol dargestellt werden. Diese sollte einen AreaChart repräsentieren. Die Basis für den AreaChart stellt uns folgende Methode:

```
public static AreaChart<Number, Number> areachart() {  
  
    NumberAxis xAxis= new NumberAxis();  
    xAxis.setLabel("times traded");  
    NumberAxis yAxis= new NumberAxis();  
    yAxis.setLabel("Depot Value");  
    AreaChart<Number, Number> areaChart = new AreaChart<Number, Number>(xAxis,yAxis);  
    areaChart.setTitle("depot value in relation to times traded");  
  
    XYChart.Series<Number, Number> data = new XYChart.Series<Number, Number>();  
    data.setName("Backtesting result for last 2 years");  
  
    visualizingMethods.create(data);  
    areaChart.setCreateSymbols(false);  
    areaChart.getData().add(data);  
    return areaChart;  
}
```

Dieser AreaChart muss jetzt jedoch auch noch mit Daten befüllt werden. Dies geschieht in der Method create()

```
public class visualizingMethods {  
    public static void create(Series<Number, Number> data) {  
        createTheChart(data);  
    }  
    public static void createTheChart(Series<Number, Number> data) {  
        for (int Rowc = 1; Rowc < umsertzungStrategie.History.size(); Rowc++) {  
            data.getData().add(new XYChart.Data<Number, Number>( Rowc, umsertzungStrategie.History.get(Rowc).getDepotWert()));  
        }  
    }  
}
```

Bester Zeitpunkt für kauf und verkauf (BarChart)

Die Zeitpunkte für den besten Kauf und Verkaufszeitpunkt wird jeweils in einem Barchart dargestellt. Dabei werden wieder Grundlegende Punkte wie Beschriftung und Titel im unteren Code erledigt

```

public static BarChart<String, Number> Barchart(int Option) throws ParseException {
    CategoryAxis xAxis = new CategoryAxis();
    NumberAxis yAxis= new NumberAxis();
    BarChart<String, Number> barChart = new BarChart<>(xAxis,yAxis);
    XYChart.Series<Number, Number> data = new XYChart.Series<Number, Number>();
    if (Option == 1 ) {
        xAxis.setLabel("Times");
        yAxis.setLabel("Times of minval");
        barChart.setTitle("Time To Buy");
        data.setName("Analysation of data for best time to buy");
        barChart.getData().addAll(barChartMethods.createdforMin(new XYChart.Series<String,Number>()));
    }
    if (Option == 2) {
        xAxis.setLabel("Times");
        yAxis.setLabel("Times of Maxval");
        barChart.setTitle("Time To Sell");
        data.setName("Analysation of data for best time to Sell");
        barChart.getData().addAll(barChartMethods.createdforMax(new XYChart.Series<String,Number>()));
    }
    barChart.setCategoryGap(20);

    return barChart;
}

```

Die Befüllung des Charts mit Daten erfolgt dann in den Methoden „createdForMax“ oder „createdForMin“.

```

public static Map<String,Integer> frequenzyMap = new HashMap<String,Integer>();

public static Series<String, Number> createdforMin(Series<String, Number> series) throws ParseException {
    getTimes(main.minimalValueTimeList);
    for (String key : frequenzyMap.keySet()) {
        String s = textfeld.fixtime(key);
        series.getData().add(new XYChart.Data<String, Number>(s, frequenzyMap.get(key)));
    }
    return series;
}

public static Series<String, Number> createdforMax(Series<String, Number> series) throws ParseException {
    getTimes(main.maximalValueTimeList);
    for (String key : frequenzyMap.keySet()) {
        String s = textfeld.fixtime(key);
        series.getData().add(new XYChart.Data<String, Number>(textfeld.fixtime(key), frequenzyMap.get(key)));
    }
    return series;
}

public static void getTimes(ArrayList<String> list) {
    // TODO Auto-generated method stub
    ArrayList<frequenzy> FList = new ArrayList<frequenzy>();
    for (String str : list) {
        Integer i = frequenzyMap.get( str);
        if ( i == null) {
            i = new Integer( 1);
        } else {
            i = new Integer( i.intValue()+1);
        }
        frequenzyMap.put( str, i);
    }
    for (String key : frequenzyMap.keySet()) {
        frequenzy f = new frequenzy(key,frequenzyMap.get(key));
        FList.add(f);
    }
}

```

Historie der durchgeführten Trades (Tabelle)

Die Historie der Strategie wird zudem in einer Tabelle festgehalten. Hier wird jeder Trade mit den dazugehörigen Daten eingetragen, um den Erfolg der Strategie und die Visualisierung zu überprüfen.

Um eine Tabelle in JavaFx anlegen zu können, müssen als erstes die einzelnen Tabellenspalten angelegt werden, diese müssen dann auch noch mit einer brauchbare Klasse verleiht werden.

```
TableView<Tabellenklasse> table = new TableView<Tabellenklasse>();
final Label label = new Label("Trading History");
label.setFont(new Font("Arial", 20));
table.setEditable(false);
fill();
TableColumn Datum = new TableColumn("Datum");
Datum.impl_setReorderable(false);
Datum.setCellValueFactory(
    new PropertyValueFactory<Tabellenklasse, String>("Datum")
);
TableColumn Positionen = new TableColumn("Positionen");
Positionen.impl_setReorderable(false);
Positionen.setCellValueFactory(
    new PropertyValueFactory<Tabellenklasse, String>("Positionen")
);
TableColumn Kaufpreis = new TableColumn("Kaufpreis");
Kaufpreis.impl_setReorderable(false);
Kaufpreis.setCellValueFactory(
    new PropertyValueFactory<Tabellenklasse, String>("Kaufpreis")
);
```

Diese Tabellenklasse besteht in diesem Fall aus Variablen vom Typ SimpleStringProperty und den dazugehörigen Gettern und Settern, sowie einem Konstruktor.

```
public class Tabellenklasse {
    private final SimpleStringProperty Datum;
    private final SimpleStringProperty Positionen;
    private final SimpleStringProperty Kaufpreis;
    private final SimpleStringProperty Verkaufspreis;
    private final SimpleStringProperty Umsatz;
```

Bei der Spalte des Umsatzes müssen die Werte erst ausgewogen werden, ob positiv oder negativ.

```
TableColumn Umsatz = new TableColumn("Umsatz");
Umsatz.impl_setReorderable(false);
Umsatz.setCellValueFactory(
    new PropertyValueFactory<Tabellenklasse, String>("Umsatz")
);
Umsatz.setCellFactory(column -> {
    return new TableCell<Tabellenklasse, String>() {
        @Override
        protected void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);
            if (item == null || empty) {
                setText("");
                setStyle("");
            } else {
                if (item.equals("true")) {
                    setTextFill(Color.GREEN);
                    setText("positiver Trade");
                } else {
                    setTextFill(Color.RED);
                    setText("negativer Trade");
                    setStyle("");
                }
            }
        }
    };
});
```

Mit der Fill Methode wird die Tabelle dann mit den Daten befüllt.

```
public static void fill() {
    for (int Rowc = 1; Rowc < umsetzungStrategie.History.size(); Rowc++) {
        data.addAll(new Tabellenklasse(umssetzungStrategie.History.get(Rowc).getDatum(),
            umsetzungStrategie.History.get(Rowc).getPositionen(),
            umsetzungStrategie.History.get(Rowc).getKaufWert(), umsetzungStrategie.History.get(Rowc).getVerkaufWert(),
            umsetzungStrategie.History.get(Rowc).isErfolg()));
    }
}
```

Textfeld mit wichtigen Kennzahlen

Die wichtigsten Kennzahlen werden in einer VBox dargestellt. Diese VBox wurde auch mit einer Border versehen, um sie deutlicher erkenntlich zu machen.

```
public static VBox showValues() {
    calcHitratio();
    VBox inhalte = new VBox();
    Label Hitratio = new Label("Hitratio : \t\t\t" + getHitratio());
    Label ProfitFactor = new Label("ProfitFactor : \t\t" + getProfitFactorORPayoutRatio(1));
    Label PayoutRatio = new Label("PayoutRatio : \t\t" + getProfitFactorORPayoutRatio(2));
    Label Kaufzeit = new Label("Kaufzeit : \t" + s1);
    Label Verkaufszeit = new Label("Verkaufszeit : \t" + s2);
    inhalte.getChildren().addAll(Hitratio, ProfitFactor, PayoutRatio, Kaufzeit, Verkaufszeit);
    inhalte.setStyle("-fx-border-style: solid inside;" +
        "-fx-border-width: 2;" +
        "-fx-border-insets: 5;" +
        "-fx-border-radius: 5;" +
        "-fx-border-color: black;");
    return inhalte;
}
```

Profit Factor und PayoutRatio

Es werden ProfitFactor und PayoutRatio berechnet:

- ProfitFactor = summe gewinner / summe verlierer
- Payout Ratio = AVG(Summe Gewinner) / AVG(summe Verlierer)

```
private static double getProfitFactorORPayoutRatio(int option) {
    double SummeDerGewinner = 0;
    double SummeDerVerlierer = 0;
    int AnzahlGewinner = 0;
    int AnzahlVerlierer = 0;
    for (int i = 0; i < GewinnBeiJeweilgemTrade.size(); i++) {
        if ((-1)*GewinnBeiJeweilgemTrade.get(i) > 0) {
            AnzahlGewinner++;
            SummeDerGewinner = SummeDerGewinner + GewinnBeiJeweilgemTrade.get(i);
        } if ((-1)*GewinnBeiJeweilgemTrade.get(i) < 0) {
            AnzahlVerlierer++;
            SummeDerVerlierer = (SummeDerVerlierer + GewinnBeiJeweilgemTrade.get(i)*(-1));
        }
    }
    if (option == 1) {
        System.out.println(SummeDerGewinner / SummeDerVerlierer);
        System.out.println(AnzahlGewinner);
        System.out.println(AnzahlVerlierer);
        return SummeDerGewinner / SummeDerVerlierer;
    } else {
        return (SummeDerGewinner/AnzahlGewinner) / (SummeDerVerlierer/AnzahlVerlierer);
    }
}

public static void calcProfitFactor(int stückzahl, double buyValue, double sellValue) {
    // TODO Auto-generated method stub
    double wertDesKaufes = stückzahl * buyValue;
    double wertDesVerkaufes = stückzahl * sellValue;
    double Gewinn = wertDesKaufes - wertDesVerkaufes;
    GewinnBeiJeweilgemTrade.add(Gewinn);
}
```

Hitratio

Es wird außerdem die Hitratio berechnet, sie gibt an wie hoch die Chance auf einen Hit (positiven Trade) ist.

```
public static void calcHitratio () {
    for (int i = 0; i < umsertzungStrategie.History.size(); i++) {
        if (umsertzungStrategie.History.get(i).isErfolg() == true) {
            erfolg = erfolg + 1.0;
        }
        else {
            nichterfolg = nichterfolg + 1.0;
        }
    }
}

private static double getHitratio() {
    return erfolg/(nichterfolg+erfolg)*100;
}
```

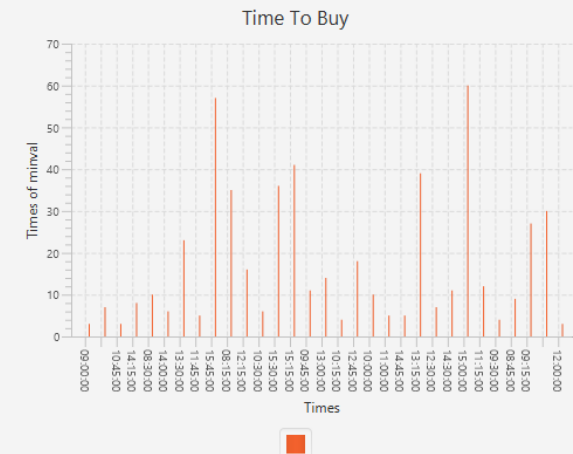
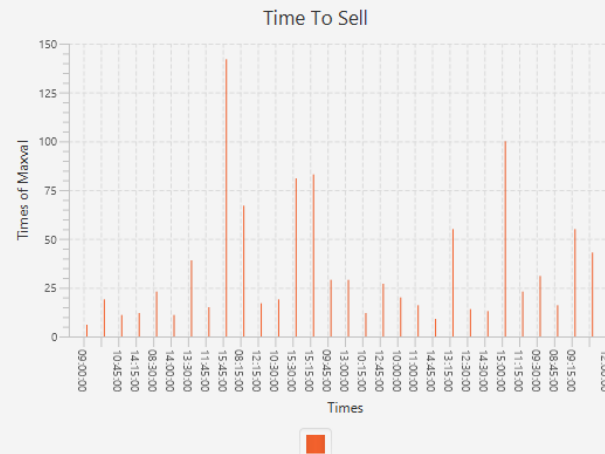
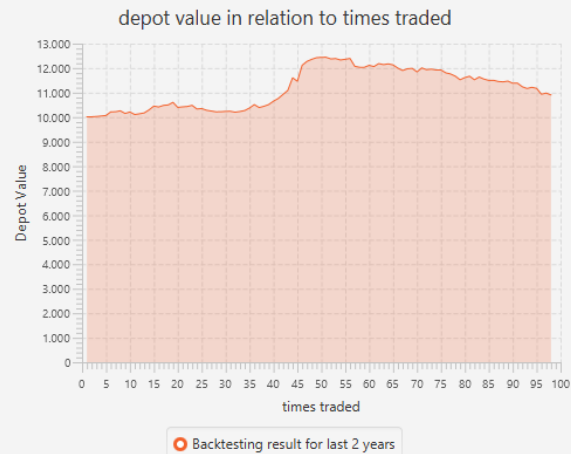
Zurück Button

Bei Interaktion mit dem Zurück Button wird die Szene wieder auf Szene 1 gesetzt und der Backtest zurückgesetzt.

```
private Object goBack() {
    st.setScene(sce2);
    areaChart.getData().clear();
    barchartMin.getData().clear();
    barchartMax.getData().clear();
    tabelle.getChildren().remove(true);
    kennzahlen.getChildren().clear();
    layout1.getChildren().clear();
    return null;
}
```

Beispiel TLT (Do kauf Fr Verkauf)

Untersuchung der ausgewählten Aktie



Hitratio : 59.59595959595959
ProfitFactor : 1.253842287120612
PayoutRatio : 0.850062567539398
Kaufzeit : 15:00:00
Verkaufszeit : 15:45:00

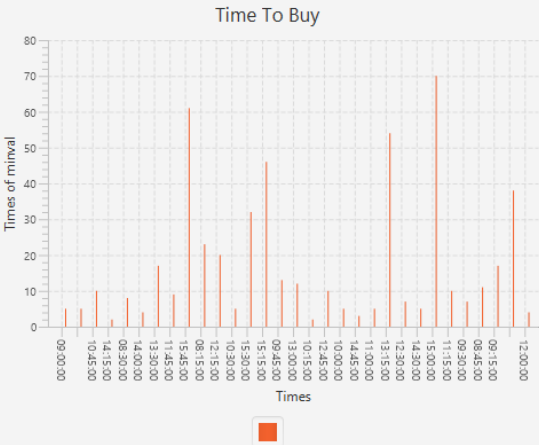
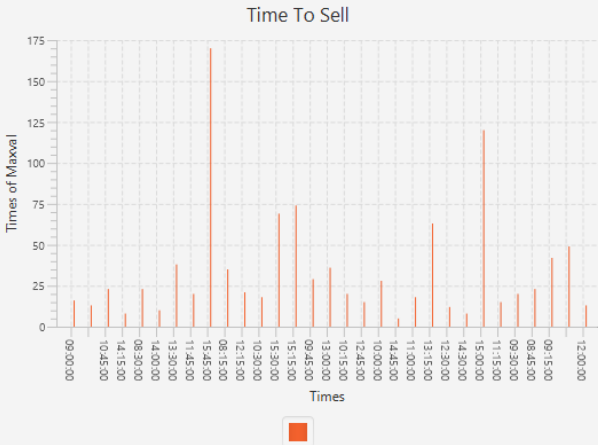
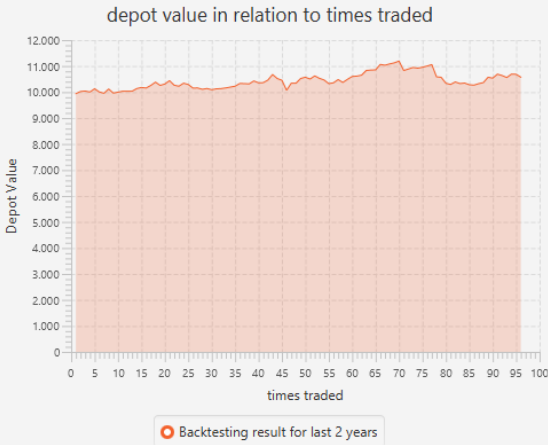
Trading History

Datum	Positionen	Kaufpreis	Verkaufpreis	Umsatz	
2019-04-25	83	119.02050018310547	119.59982299804688	positiver Trade	
2019-05-02	83	119.40032196044922	119.41434478759766	positiver Trade	
2019-05-09	82	120.98177337646484	121.1317367553711	positiver Trade	
2019-05-16	82	121.5560073852539	121.79450988769531	positiver Trade	
2019-05-23	81	123.11036682128906	123.3716049194336	positiver Trade	
2019-05-30	80	124.84217834472656	126.6902847290039	positiver Trade	
2019-06-06	80	127.42627716064453	127.50074005126953	positiver Trade	
2019-06-13	80	126.90933227539062	127.37470245361328	positiver Trade	
2019-06-20	79	129.15859985351562	127.80128479003906	negativer Trade	
2019-06-27	79	128.13101196289062	128.87744140625	positiver Trade	
2019-07-11	80	127.41157531738281	126.1390151977539	negativer Trade	
2019-07-18	79	127.66414642333984	128.02357482910156	positiver Trade	
2019-07-25	79	127.24352264404297	127.71272277832031	positiver Trade	
2019-08-01	77	130.385498046875	131.99147033691406	positiver Trade	
2019-08-08	76	135.2034149169922	137.22792053222656	positiver Trade	
2019-08-15	73	142.13343811035156	141.62730407714844	negativer Trade	

Go Back to choose new Parameters

Beispiel GLD (Fr kauf Mo Verkauf)

Untersuchung der ausgewählten Aktie



Hitratio : 56.70103092783505
ProfitFactor : 1.148007950718027
PayoutRatio : 0.8766606169119479
Kaufzeit : 15:00:00
Verkaufszeit : 15:45:00

Trading History				
Datum	Positionen	Kaufpreis	Verkaufpreis	Umsatz
2019-05-05	82	120.86000061035156	120.80500030517578	negativer Trade
2019-05-12	81	121.44999694824219	122.48999786376953	positiver Trade
2019-05-19	83	120.37000274658203	120.5699969482422	positiver Trade
2019-05-27	82	121.12000274658203	120.66999816894531	negativer Trade
2019-06-02	81	122.87999725341797	124.42279815673828	positiver Trade
2019-06-09	79	126.87000274658203	125.3199969482422	negativer Trade
2019-06-16	78	127.11000061035156	126.45500183105469	negativer Trade
2019-06-23	75	131.3300018310547	133.52040100097656	positiver Trade
2019-06-30	75	133.14999389648438	131.05499267578125	negativer Trade
2019-07-07	75	131.22999572753906	131.6999969482422	positiver Trade
2019-07-14	75	132.66000366210938	133.16000366210938	positiver Trade
2019-07-21	74	134.5399932861328	134.50999450683594	negativer Trade
2019-07-28	74	133.83999633789062	133.8699951171875	positiver Trade
2019-08-04	73	136.3000030517578	137.67999267578125	positiver Trade
2019-08-11	71	141.3699951171875	141.97000122070312	positiver Trade
2019-08-18	71	141.8000030517578	141.49000549316406	negativer Trade

Go Back to choose new Parameters