DOKUMENTATION PYTHON-BACKTESTING-SYSTEM

Durchführung in Python:

Einfache Variante ist http://code.google.com/p/winpython/
Stand-alone Variante (auch in 64bit) ohne Installation und inklusive der wichtigsten Pakete aus python(x,y).
Entwickler ist ebenfalls der von python(x,y) - er arbeitet jetzt nur noch an winpython wenn ich das richtig lese.
Als Editor kann der mitgelieferte Editor Spyder benutzt werden. Design ist an Matlab angelehnt.
--> Zumindest die 64bit-Variante von winpython arbeitet schneller als python(x,y).

Dateien:

(1) ticker.txt
(2) 1_GetData.py
(3) 2_RestructData.py
(4) 3_GetSignals.py
(5) 4_Backtesting.py
(6) Utils.py

--> Dateien müssen sich in identischem Ordner befinden

Programmablauf:

- Daten von YahooFinance laden anhand der Tickerliste
- Fehlerbereinigung der Kursdaten
- tägliche Signale finden (Scoring aller Aktien)
- Backtesting der Signale
- Kennwerte und Depotverlauf berechnen und ausgeben

--> Programme müssen nacheinander ausgeführt (öffnen mit Spyder und starten mit F5) werden. Kein Backtesting ohne Signale. Keine Signale ohne Kursdaten. Wenn alles erstellt ist, kann das Backtesting oder die Signalgenerierung wiederholt werden um andere Parameter oder Einstiegssignale zu testen.

Allgemeiner Hinweis für die nun erklärten Programme:
bTestmode = true oder false
In jedem Programm implementiert. Hiermit werden nur ein paar Schleifendurchgänge durchgeführt. Also nur einige Daten heruntergeladen, auf Signale geprüft oder nur einige Trades im Backtesting behandelt. Anzahl der Durchläufe finden sich im Programmcode.

Programmdoku:

(1) ticker.txt:
Tickerliste mit Texteditor öffnen und anpassen. Jeweils eine Zeile für eine Aktie (Ticker nach yahoo-finance)

(2) 1_GetData.py :
Datei in Python-Editor (Spyder) öffnen.
Datendownload: date1 (Start) und date2 (Ende) der Studie im Programmcode angeben (Zeile 24 und 25).
Dateiname der Tickerliste überprüfen.
Nach Programmstart (mit F5 oder unter dem Reiter "Run" wird für jede Aktie täglich [Date, Open, Close, High, Low, Volume] geladen und im selben Ordner gespeichert. Eine Datei pro Aktie (z.B. "RAW_mData_ADS.DE.npy" für Adidas).

--> Info über Programmablauf im Terminalfenster unten rechts. Abbruch der Anwendung über das orange Ausrufezeichen im Terminalfenster (kill the current process).

(3) 2_RestructData.py:
Fehlerbereinigung:
Open und Close hin und wieder außerhalb von high/low
high/low wird an Open/Close herangezogen
wenn Volumen an einem Tag =NULL, wird Vortageskerze wiederholt
Danach werden zusätzliche Kennwerte wie ATR2, ATR10, lokale Hochs und Tiefs, SMA20 und SMA100 erstellt, an die Matrix gehängt und gespeichert (z.B. "mData_ADS.DE.npy").
Rohdaten bleiben erhalten, können also bei Änderung an Datei (3) nicht neu geladen werden!

(4) 3_GetSignals.py:
Zunächst einen Datumsvektor erzeugen, so dass bekannt ist, für welche Tage Kursdaten vorhanden sind.
Scoring pro Aktie erstellen: Jeden Tag untersuchen auf verschiedene Einstiegsfaktoren und Scoring für extra Bedingungen.

Mindestvoraussetzungen Long am Tag [k] (Schleifendurchlauf):
```
# ATR2 niedriger als ATR10
if vDataATR2[k] < vDataATR[k]:
# bullishe Kerze
if vDataClose[k] > vDataOpen[k]:
# günstigerer Einstieg
if vDataHigh[k] < vDataHigh[k-1]:
# Range < 0,8 ATR10
if (vDataHigh[k] - vDataLow[k]) < 0.8*vDataATR[k]:
# Close in oberer Hälfte
if vDataClose[k] >= (vDataHigh[k] - (vDataHigh[k] - vDataLow[k])/2):
```

--> Das waren die Mindestvoraussetzungen. Wenn diese an einem Tag erfüllt sind, geht es in weitere Bedingungen.

Jetzt werden für zusätzliche Bedingungen Bonus-Punkte verteilt:
```
# mindestens 3 Tage Korrektur
if vDataClose[k] < vDataHigh[k-1] and vDataClose[k] < vDataHigh[k-2] and vDataClose[k] < vDataHigh[k-3]:
    vScoreL[k] = vScoreL[k] +1
    # aktuelle Vola kleiner als am Vortag
```

```
    if abs(vDataATR[k]) < abs(vDataATR[k-1]):
        vScoreL[k] = vScoreL[k] +1
    # naher Trigger
    if (vDataHigh[k] - vDataClose[k]) < ((0.3*vDataHigh[k]) - vDataLow[k]):
        vScoreL[k] = vScoreL[k] +1
    # fallende Lows, Korrektur im Gange
    if vDataLow[k] < vDataLow[k-1] < vDataLow[k-2]:
        vScoreL[k] = vScoreL[k] +1
    # besonders kleine Kerze
    if vDataHigh[k] - vDataLow[k] < 0.6*vDataATR[k]:
        vScoreL[k] = vScoreL[k] +1
    # Close am Tageshoch
    if vDataClose[k] == vDataHigh[k]:
        vScoreL[k] = vScoreL[k] +1
    # Abzug für geringes Volumen
    if vDataVolume[k] * vDataClose[k] < 2000000:
        vScoreL[k] = vScoreL[k] -1
    # Abzug für geringes Volumen
    if vDataVolume[k] * vDataClose[k] < 1000000:
        vScoreL[k] = vScoreL[k] -1
    # kein neues Tief
    if vLMax2[k] > vLMax15[k]:
        vScoreL[k] = vScoreL[k] +1
    # neues Tief (Punktabzug)
    if vLMax2[k] <= vLMax15[k]:
        vScoreS[k] = vScoreS[k] -1
    # Punkte für Korrekturausmaß
    if vHMax15[k] - vDataHigh[k] > 0.4 * vDataATR[k]:
        vScoreL[k] = vScoreL[k] +1
    if vHMax15[k] - vDataHigh[k] > 0.6 * vDataATR[k]:
        vScoreL[k] = vScoreL[k] +1
    if vHMax15[k] - vDataHigh[k] > 0.8 * vDataATR[k]:
        vScoreL[k] = vScoreL[k] +1
    if vDataSMA100[k] < vDataSMA20[k]:
        vScoreL[k] = vScoreL[k] +1
    # steigender SMA20
    if vDataSMA20[k] >= vDataSMA20[k-1]:
        vScoreL[k] = vScoreL[k] +1
```

.
Umgekeht dann für den Short-Fall. Das hier ist eine große Baustelle. Long- und Short-Bedingungen müssen nicht zwangsläufig genau entgegengesetzt sein.
Dies wird für alle Aktien durchgeführt. Am Ende wird nochmals geschaut, welche Aktie an einem Tag den höchsten Wert erreicht hat. Sofern die Mindestbedingungen erfüllt sind wird pro Tag jeweils ein Long- und Shortsignal erstellt und für die Verarbeitung im Backtesting gespeichert. Es wird der Wert mit der maximalen Punktzahl ausgewählt, bei Punktgleichheit wird gewürfelt! Deshalb können für eine Strategiedefinition mehrere Backtestings unterschiedliche Ergebnisse entstehen. Ein Würfeln z.B. mit unterschiedlicher Gewichtung der Einstiegsparameter verhindert werden. [wie gesagt: Baustelle, Spielwiese]

Gespeichert werden: "mSignals_Long.npy", "mSignalDates_Long.npy", "mSignalPunkte_Long.npy" (+ 3x für den Short-Fall)

(5) 4_Backtesting.py:
Hier wird das Backtesting durchgeführt. Zunächst die Erklärung der Variablen zu Beginn des Programmcodes als Stellglieder:

```
bTestmode: true oder false - im Testmodus werden nur die ersten 500 Trades betrachtet

Stellglieder:
Was soll ausgewertet werden? Long und/oder Shorttrades?
bLong = "true"
bShort = "true"

Chart zu jedem Trade speichern (dauert 0.5-1 Sekunde extra pro Trade!)
bSaveTrade = "false"

Chart zu jedem Trade anzeigen (Chart-Fenster muss immer weggeklickt werden. Abschalten für
schnellen Durchlauf)
bShowTrade = "false"

Charts zur Zusammenfassung zeigen. (GuV, Depotvalue)
bShowFig = "true"

Charts zur Zusammenfassung speichern. (GuV, Depotvalue)
bSaveFig = "false"

Clean-Mode: ohne Spread, Gebühren, Slippage
bCleanMode = "false"

Stellglieder die relevant für die Performance sind. Feintuning der Strategie:
Differenzbetrag über High zum Einstieg in EUR (2 Cent):
iDiffEntry = 0.02

Spread (Prozentsatz dezimal) 0,1%:
iSpread = 0.001

Ziel CRV:
cCRV = 2

StopLoss (x*ATR):
cSL = 1.0

TakeProfit:
cTP = cSL * cCRV

Startkapital:
iDepotValue = 10000

Risiko pro Trade (Prozentsatz dezimal):
iRisk = 0.01

Slippage (Prozentsatz dezimal):
iSlippage = 0.001

Am Ende des fünften Tages Position schließen:
iTimeStop = 5

Fixe Gebühr in EUR pro Halfturn:
iFixGeb = 0

var. Gebühr dezimal ETX: 0.001(0.00085 inkl. Rabatt) [ETX-Capital & BrokerDeal]:
iVarGeb = 0.00085
```

Minimum-Score für Einstieg (muss durchgetestet werden):
iMinScore = 7

Die Variablendefinition beinhalten also die Strategiedefinitionen und können sinnvoll angepasst werden für ein Backtesting.
Im Anschluss werden nun alle Kursdaten der zu untersuchenden Aktien geladen und in ein Dictionary abgelegt. Die Signal-Dateien werden ebenfalls geladen und in eine Reihenfolge (Long+Short) gebracht.
Dann wird jedes Signal auf Ausführung überpüft. Die Eröffnung mit einem Gap wird abgedeckt - Stoploss und TakeProfit werden nach Ausführung angepasst. Danach weitere Betrachtung bis zum Timestop.

Gewinn-/Verlust und Depotwert werden als Vektoren geführt, nach Abschluss kann also eine Entwicklung betrachtet werden. Ein Gewinn oder Verlust wird an dem Tag dem Depotwert zugeschrieben, an dem der Trade abgeschlossen wurde (zwischenzeitliche Buchgewinne sind später also nicht sichtbar).

Im Anschluss ein wenig Statistik:
- min/max/mean Gewinn/Verlust/Rentabilität in R
- Trefferquoten
- Profitfaktor long/short/gesamt
- ausgeführte/gelöschte Trades

--> Ausgabe im Terminalfenster (bei Spyder unten rechts)

wenn Graphen zu den Trades angezeigt (bShowTrade = "true") oder gespeichert (bSaveTrade = "true") werden:
- Kennzahlen zum Trade in Titelzeile
- SMA20 in blau
- SMA100 in grün
- Open: weißer Punkt
- StopLoss: roter Punkt
- TakeProfit: grüner Punkt
- Close: schwarzer Punkt

Graphen der Zusammenfassung (bShowFig = "true" oder bSaveFig = "false"):
1. Histogramm der Rentabilität in R. [Im Idealfall die höheren Häufigkeiten auf der rechten (Gewinn-)Seite]
2. Histogramm für die Tradedauer (gemessen in ganzen Tagen)
3. Rentabilität der Trades. Ein Plus (+) pro Trade. [Ausreißer sind zu vermeiden]
4. Gewinn und Verlust der Trades. Ein Plus (+) pro Trade. [Gesamtentwicklung und Ausreißer betachten]
5. Entwicklung des Depots.

Hinweis: Das Jahr 2001 wird nicht betrachtet, da hier die Kursdaten kompliziert waren. Es scheint so als wären dort an einigen Stellen Aktiensplits nicht korrekt umgesetzt. Deshalb habe ich das Jahr in Zeile 294 vom Programmcode übersprungen.

(6) Utils.py:

Hier sind Funktionen ausgelagert für die Berechnung von Moving-Average und speziellen Vektor-Operationen.