암호화폐 트레이딩 교육

Day 3 (1/2)

백테스팅

Kangwuk Heo
calvin.heo@gmail.com

Contents

Overview

암호화폐 트레이딩 교육 과정에서, 실습을 통해서, 트레이딩이 무엇인지, 트레이딩을 어떤 방식으로 구성하고 진행하는지를 배울 수 있습니다.

백테스팅(Backtesting)

Section 1 은 백테스팅을 위한 데이터 준비 및 변동성 돌파 전략 및 상승전략 기준으로 테스트를 진행하는 단계가 포함되어 있습니다.

Section 1: 백테스팅

백테스팅(Backtesting)은 과거 데이터를 사용해서 투자 전략이 어느 정도의 수익률이 나는지를 확인하는 과정입니다.

과거 데이터를 기반으로 트레이딩 전략의 실행 가능성을 확인하고, 정립한 투자 전략이 어떻게 수행되고, 어떤 결과를 보이는지를 테스트를 통해 방향성 및 전략의 위험성, 잠재적인 수익성을 분석할 수 있습니다.

__1. 가상화폐의 과거 데이터(일봉)를 가상화폐 거래소로부터 가져오는지 실행해 봅니다

```
import pybithumb

df = pybithumb.get_ohlcv("BTC")
print(df.tail(10))
```

```
2021-11-07 00:00:00  74026000.0  75771000.0  ...  74914000.0  2544.977262
2021-11-08 00:00:00  74901000.0  80124000.0  ...  79217000.0  3826.749156
2021-11-09 00:00:00  79253000.0  82477000.0  ...  80611000.0  4027.476441
2021-11-10 00:00:00  80641000.0  82445000.0  ...  81806000.0  4036.842562
2021-11-11 03:00:00  81807000.0  82198000.0  ...  81738000.0   384.623030
```

__2. 가상화폐일 과거 데이터 (일봉 차트) 를 엑셀파일로 저장되는지 확인해 봅니다.

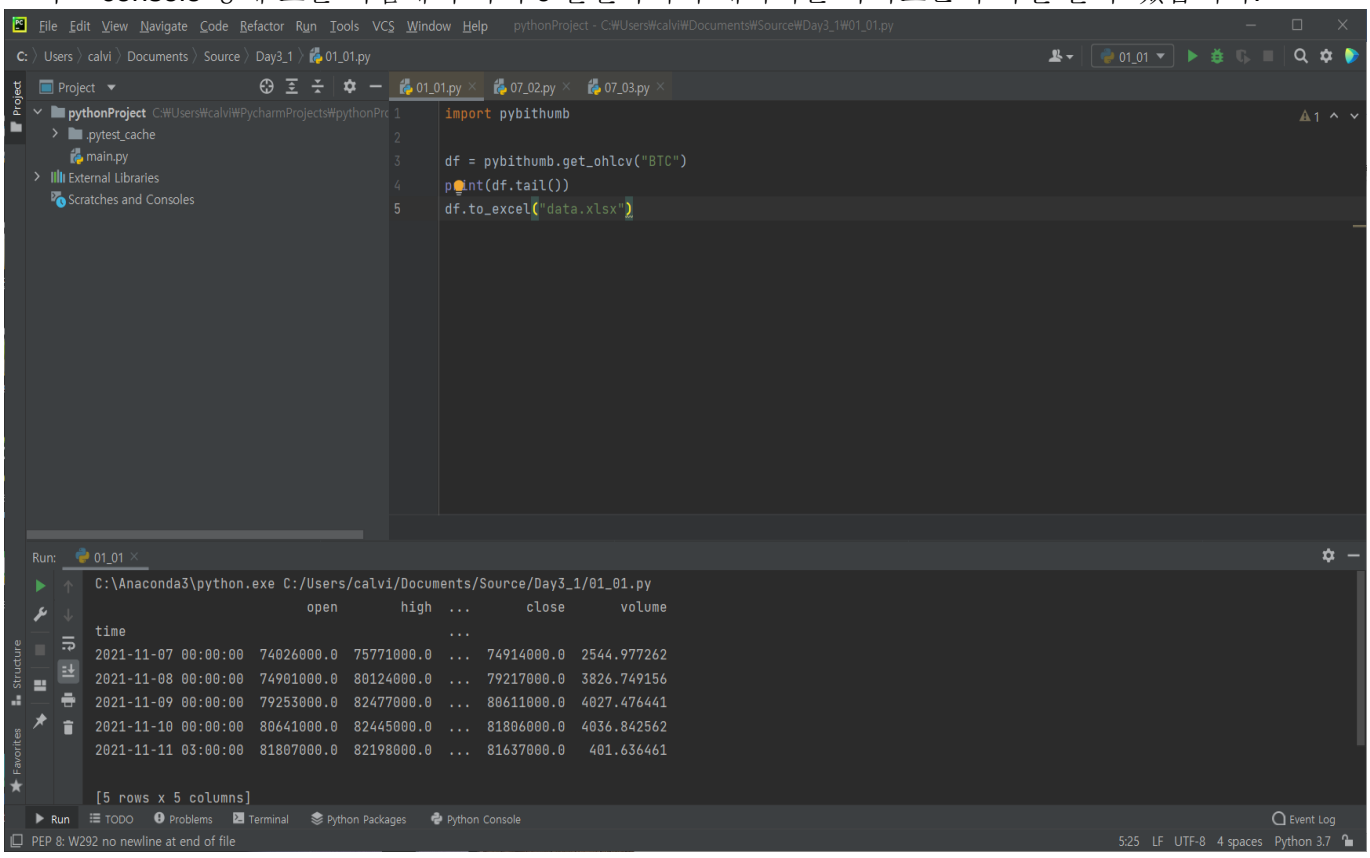
```
import pybithumb

df = pybithumb.get_ohlcv("BTC")
print(df.tail(10))
df.to_excel("data.xlsx")
```

지정한 파일명으로 엑셀파일이 저장되어, 파이썬 코드가 있는 동일한 디렉토리에 엑셀파일이 생성된 것을 확인할 수 있습니다.

엑셀 파일을 open 해 보면, 가상화폐의 과거 데이터가 시작시점부터 오늘시점까지 전체 데이터가 저장된 것을 확인할 수 있습니다.

time	open	high	low	close	volume
2013-12-27 00:00:00	737000	755000	737000	755000	3.78
2013-12-28 00:00:00	750000	750000	750000	750000	12
2013-12-29 00:00:00	750000	750000	728000	739000	19.058
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303
2014-01-04 00:00:00	831000	846000	819000	846000	107.757
2014-01-05 00:00:00	846000	925000	845000	925000	140.7098
2014-01-06 00:00:00	928000	1050000	920000	1010000	222.7226
2014-01-07 00:00:00	1001000	1026000	900000	935000	169.0501
2014-01-08 00:00:00	935000	960000	825000	897000	194.5596
2014-01-09 00:00:00	880000	909000	860000	865000	140.5152
2014-01-10 00:00:00	875000	905000	862000	890000	126.6037
2014-01-11 00:00:00	889000	950000	880000	932000	144.6352
2014-01-12 00:00:00	935000	952000	904000	910000	129.8415
2014-01-13 00:00:00	920000	925000	872000	881000	105.8332
2014-01-14 00:00:00	877000	905000	872000	874000	119.6229
2014-01-15 00:00:00	889000	918000	875000	905000	138.0606
2014-01-16 00:00:00	895000	940000	892000	910000	216.1478

그리고 console 창에 오늘 시점에서 과거 9 일전까지의 데이터를 가져오는지 확인 할 수 있습니다.



__3. 기본 데이터를 기반으로 새로운 조건 항목을 추가하여, 항목이 생성되는지를 확인합니다.

```
import pybithumb

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df.to_excel("data1.xlsx")
```

새로운 항목인 range 는 가상화폐의 고가(high)와 저가(low)를 뺀 나머지에 0.5 를 곱한 값으로 처리합니다.

time	open	high	low	close	volume	range
2013-12-27 00:00:00	737000	755000	737000	755000	3.78	9000
2013-12-28 00:00:00	750000	750000	750000	750000	12	0
2013-12-29 00:00:00	750000	750000	728000	739000	19.058	11000
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973	16000
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035	18500
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033	15000
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398	7500
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303	33000
2014-01-04 00:00:00	831000	846000	819000	846000	107.757	13500

__4. 기본 데이터를 기반으로 새로운 항목인 (목표가)을 추가해 봅니다.

```
import pybithumb

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)
df.to_excel("data2.xlsx")
```

목표가는 가상화폐의 당일 시가에 Range 를 더한 값으로, 각 거래일 기준으로 전날의 레인지를 사용하는 방식을 사용합니다.

목표가 range 기준으로 한컬럼 아래에서 시작된 것을 확인할 수 있습니다.

time	open	high	low	close	volume	range	target
2013-12-27 00:00:00	737000	755000	737000	755000	3.78	9000	
2013-12-28 00:00:00	750000	750000	750000	750000	12	0	759000
2013-12-29 00:00:00	750000	750000	728000	739000	19.058	11000	750000
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973	16000	751000
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035	18500	784000
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033	15000	786500
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398	7500	791000
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303	33000	785500

__5. 열 항목들에 데이터를 비교하기 위해 기본적으로 제공하는 함수들을 활용하여 생성해야 합니다.

첫번째로 목표가와 고가를 비교하여 수익률(ror)을 계산해야 합니다. 수익률 계산을 위해서는 엑셀파일에 있는 각각의 행에 존재하는 비교 값 기준으로 행별로 계산되어야 합니다.

where 함수는 numpy 모듈에 포함되어 있으며, 행 단위로 조건문을 적용할 수 있습니다.
(조건, 조건이 참일때의 값, 조건이 거짓일 때의 값)

where 함수가 어떤 형식으로 동작되는지를 확인해 봅니다.

```
import numpy as np
from pandas import DataFrame

data = {'test1': [200, 300, 400],
        'test2': [90, 210, 300]}

df = DataFrame(data)
df['low'] = np.where(df['test1'] < df['test2'], 'test1', 'test2')
df.to_excel("test.xlsx")
```

Data 배열 순서대로 엑셀파일에 데이터가 포함되면서, where 조건문 기반, low

	test1	test2	low
0	200	90	test2
1	300	210	test2
2	400	300	test2

수익률을 계산하여 엑셀파일에 새로운 열로 추가되도록 실행해 봅니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)

df['ror'] = np.where(df['high'] > df['target'],
                     df['close'] / df['target'],
                     1)

df.to_excel("data3.xlsx")
```

정상적으로 실행되고 나면, 엑셀파일에 수익률(ror)이 추가 된 것을 확인할 수 있습니다.

time	open	high	low	close	volume	range	target	ror
2013-12-27 00:00:00	737000	755000	737000	755000	3.78	9000		1
2013-12-28 00:00:00	750000	750000	750000	750000	12	0	759000	1
2013-12-29 00:00:00	750000	750000	728000	739000	19.058	11000	750000	1
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973	16000	751000	1.022636
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035	18500	784000	0.979592
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033	15000	786500	0.987921
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398	7500	791000	1
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303	33000	785500	1.061744

__6. 수익률 기준으로 기간 수익률(HPR : Holding Period Return) 계산하여, 엑셀파일에 추가합니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)

df['ror'] = np.where(df['high'] > df['target'],
                     df['close'] / df['target'],
                     1)

ror = df['ror'].cumprod()[-2]
print(ror)
```

Cumpod 는 series 객체에서 모든 값을 곱해주는 역할을 수행하는 메소드입니다.

Cumpod 메소드를 기반으로 수익률(ror) 곱하여 전체 수익률을 확인 할 수 있습니다.

70.39413327250776

가상화폐의 데이터를 기간별 (년도) 기준으로 가져와서, 해당년도 데이터를 기준으로 수익률을 계산할 수 있습니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")
df = df['2019']

df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)

df['ror'] = np.where(df['high'] > df['target'],
                     df['close'] / df['target'],
                     1)

ror = df['ror'].cumprod()[-2]
print(ror)
```

1.0689868284522164

__7. 거래수수료를 반영하여, 실제 수수료를 계산해 봅니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)

fee = 0.0015
df['ror'] = np.where(df['high'] > df['target'],
                     df['close'] / df['target'] - fee,
                     1)

ror = df['ror'].cumprod()[-2]
print(ror)
```

12.690917847442396

가상화폐 거래소에서 매도/매수 주문시, 거래량 부족으로 가격대보다 조금 더 비싸게 매수되거나, 조금 더 싸게 매도 될수 있는 상황에서, 발생하는 비용을 슬리피지라고 합니다.

__8. 거래수수료를 반영하여, 높은 기간 수익률을 계산해 봅니다.

```
import pybithumb
import numpy as np


def get_ror(k=0.5):
    df = pybithumb.get_ohlcv("BTC")
    df['range'] = (df['high'] - df['low']) * k
    df['target'] = df['open'] + df['range'].shift(1)

    fee = 0.0015
    df['ror'] = np.where(df['high'] > df['target'],
                         df['close'] / df['target'] - fee,
                         1)

    ror = df['ror'].cumprod()[-2]
    return ror


for k in np.arange(0.1, 1.0, 0.1):
    ror = get_ror(k)
    print("%.1f %f" % (k, ror))
```

0.1 0.395360
0.2 1.096279
0.3 2.353348
0.4 8.223662
0.5 12.690918
0.6 23.652168
0.7 25.662492
0.8 21.031787
0.9 12.747428

__9. 최대누적손실률(MDD : Maximum Draw Down) 계산해 봅니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)

fee = 0.0015
df['ror'] = np.where(df['high'] > df['target'],
                     df['close'] / df['target'] - fee,
                     1)

df['hpr'] = df['ror'].cumprod()
df.to_excel("data4.xlsx")
```

최대누적 손실률은 투자 기간중에 포트폴리오의 전 고점에서 저점까지의 최대 누적 손실입니다.

MDD 를 계산하기 위해서는 거래일별로 기간 수익률 데이터를 생성해야 합니다.

거래일마다 기간 수익률을 계산하여, hpr 항목으로 생성합니다.

time	open	high	low	close	volume	range	target	ror	hpr
2013-12-27 00:00:00	737000	755000	737000	755000	3.78	9000		1	1
2013-12-28 00:00:00	750000	750000	750000	750000	12	0	759000	1	1
2013-12-29 00:00:00	750000	750000	728000	739000	19.058	11000	750000	1	1
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973	16000	751000	1.019436	1.019436
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035	18500	784000	0.976392	0.995369
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033	15000	786500	0.984721	0.980161
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398	7500	791000	1	0.980161
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303	33000	785500	1.058544	1.037544
2014-01-04 00:00:00	831000	846000	819000	846000	107.757	13500	864000	1	1.037544

__10. 각 거래일별 기간 수익률 데이터를 기반으로 MDD 값을 계산해 봅니다.

MDD 는 아래와 같은 계산 규칙을 가지고 있습니다.

MDD = (max – low) / max * 100

MDD 를 계산하기 전, 거래일별 낙폭을 구하고, 낙폭 중 최대값 기준으로 MDD 를 처리해 보도록 합니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)

fee = 0.0015
df['ror'] = np.where(df['high'] > df['target'],
                     df['close'] / df['target'] - fee,
                     1)

df['hpr'] = df['ror'].cumprod()
df['dd'] = (df['hpr'].cummax() - df['hpr']) / df['hpr'].cummax() * 100
print("MDD(%): ", df['dd'].max())
df.to_excel("data5.xlsx")
```

낙폭 값(dd) 기준 전고점을 추출하는 함수는 cummax 메소드를 통해서 쉽게 계산할 수 있습니다.

time	open	high	low	close	volume	range	target	ror	hpr	dd
2013-12-27 00:00:00	737000	755000	737000	755000	3.78	9000		1	1	0
2013-12-28 00:00:00	750000	750000	750000	750000	12	0	759000	1	1	0
2013-12-29 00:00:00	750000	750000	728000	739000	19.058	11000	750000	1	1	0
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973	16000	751000	1.021136	1.021136	0
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035	18500	784000	0.978092	0.998765	2.190816
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033	15000	786500	0.986421	0.985203	3.518951
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398	7500	791000	1	0.985203	3.518951
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303	33000	785500	1.060244	1.044556	0
2014-01-04 00:00:00	831000	846000	819000	846000	107.757	13500	864000	1	1.044556	0
2014-01-05 00:00:00	846000	925000	845000	925000	140.7098	40000	859500	1.074707	1.122592	0
2014-01-06 00:00:00	928000	1050000	920000	1010000	222.7226	65000	968000	1.041888	1.169615	0

MDD(%): 58.67817758257955

__11. 최대누적손실률(MDD : Maximum Draw Down) 및 누적수일률을 계산해 봅니다.

```
import pybithumb
import numpy as np

df = pybithumb.get_ohlcv("BTC")

df['ma5'] = df['close'].rolling(window=5).mean().shift(1)
df['range'] = (df['high'] - df['low']) * 0.5
df['target'] = df['open'] + df['range'].shift(1)
df['bull'] = df['open'] > df['ma5']

fee = 0.0015
df['ror'] = np.where((df['high'] > df['target']) & df['bull'],
                     df['close'] / df['target'] - fee,
                     1)

df['hpr'] = df['ror'].cumprod()
df['dd'] = (df['hpr'].cummax() - df['hpr']) / df['hpr'].cummax() * 100
print("MDD: ", df['dd'].max())
print("HPR: ", df['hpr'][-2])
df.to_excel("data6.xlsx")
```

이동평균값을 이용하여, 상승장인지, 하락장이인지에 따라 매수결정 여부를 판단할 수 있는 조건 결과를 생성합니다.

MDD: 31.285437455485717
HPR: 51.05449149344027

time	open	high	low	close	volume	ma5	range	target	bull	ror	hpr	dd
2013-12-27 00:00:00	737000	755000	737000	755000	3.78		9000		FALSE	1	1	0
2013-12-28 00:00:00	750000	750000	750000	750000	12		0	759000	FALSE	1	1	0
2013-12-29 00:00:00	750000	750000	728000	739000	19.058		11000	750000	FALSE	1	1	0
2013-12-30 00:00:00	740000	772000	740000	768000	9.488973		16000	751000	FALSE	1	1	0
2013-12-31 00:00:00	768000	800000	763000	768000	18.65035		18500	784000	FALSE	1	1	0
2014-01-01 00:00:00	768000	795000	765000	777000	65.38033	756000	15000	786500	TRUE	0.986421	0.986421	1.357883
2014-01-02 00:00:00	776000	788000	773000	778000	83.20398	760400	7500	791000	TRUE	1	0.986421	1.357883
2014-01-03 00:00:00	778000	840000	774000	834000	124.3303	766000	33000	785500	TRUE	1.060244	1.045847	0
2014-01-04 00:00:00	831000	846000	819000	846000	107.757	785000	13500	864000	TRUE	1	1.045847	0
2014-01-05 00:00:00	846000	925000	845000	925000	140.7098	800600	40000	859500	TRUE	1.074707	1.123979	0
2014-01-06 00:00:00	928000	1050000	920000	1010000	222.7226	832000	65000	968000	TRUE	1.041888	1.171061	0

__12. 수익률이 높은 가상화폐를 무엇인지, 가상화폐 거래소를 통해 계산해 봅니다.

```
import pybithumb
import numpy as np


def get_hpr(ticker):
    df = pybithumb.get_ohlcv(ticker)


    df['ma5'] = df['close'].rolling(window=5).mean().shift(1)
    df['range'] = (df['high'] - df['low']) * 0.5
    df['target'] = df['open'] + df['range'].shift(1)
    df['bull'] = df['open'] > df['ma5']

    fee = 0.0015
    df['ror'] = np.where((df['high'] > df['target']) & df['bull'],
                         df['close'] / df['target'] - fee,
                         1)

    df['hpr'] = df['ror'].cumprod()
    df['dd'] = (df['hpr'].cummax() - df['hpr']) / df['hpr'].cummax() * 100
    return df['hpr'][-2]


tickers = pybithumb.get_tickers()

hprs = []
for ticker in tickers:
    hpr = get_hpr(ticker)
    hprs.append((ticker, hpr))

sorted_hprs = sorted(hprs, key=lambda x:x[1])
print(sorted_hprs[-10:])
```

기간 수익률은 오름차순으로 정렬하여, 총 10 개의 코인을 추출해 봅니다.

[('SOC', 10.143010266333778),
('WEMIX', 10.397083456935825),
('BCH', 15.20085424620074),
('TMTG', 22.878061122418874),
('ADA', 23.73890361902178),
('XRP', 34.38664877099747),
('LINK', 36.55826546410901),
('BTC', 51.05449149344027),
('ETH', 77.60063608474826),
('MTL', 79.4265279974704)]

연습문제

BTC 가 아닌, 타 가상화폐(예 : XRP, ETC) 를 기준으로 1 번항목부터 11 번까지 순서대로 테스트를 진행해 봅니다.