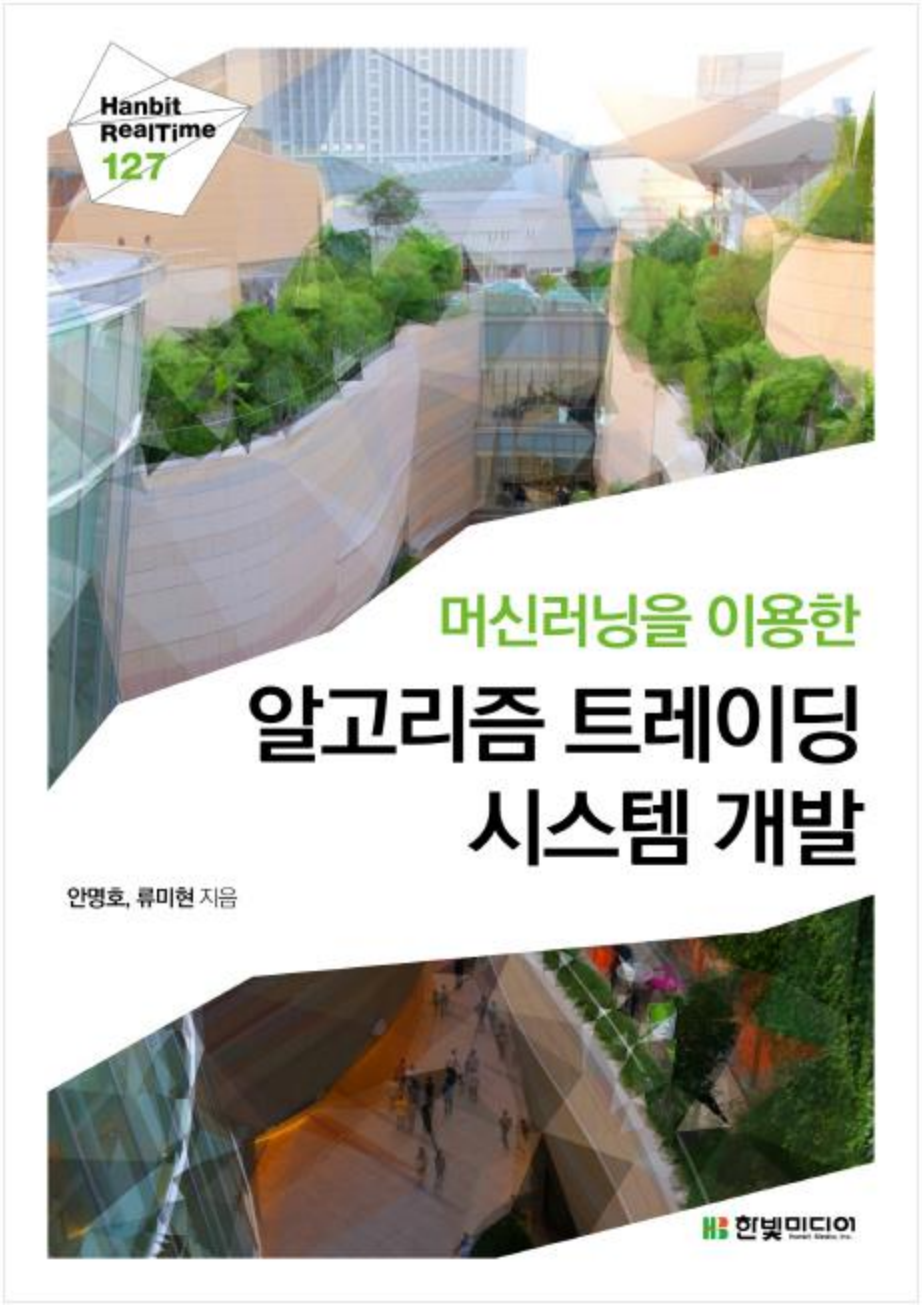
Finance data analysis in Python

2017-06-14

박현종



평점 ★★★½☆ 7.0
리뷰 8건
저자 안명호, 류미현
출판사 한빛미디어
출판일 2016.05.24

도서
16,200원 ~~18,000원~~ -10%

가격정보

예제소스 다운로드

Contents

1. 금융 데이터 소개
2. 금융 데이터 분석을 위한 파이썬 라이브러리들
3. 금융 데이터 수집
4. 예측 모델1 (평균회귀모델)
5. 예측 모델2 (머신러닝)
6. 정리

증권 분석

증권 분석은 크게 3가지로 나뉩니다

- 1. **기본 분석** Fundamental analysis
  - 기본요인(fundamentals)인 내재가치(intrinsic value)를 분석
- 2. **기술 분석** Technical analysis
  - 과거 가격과 거래량의 변화를 기초로 가격의 움직임을 예측
- 3. **정서 분석** Sentiment analysis
  - 투자자들의 시장에 대한 욕심이나 두려움을 분석

시장가치(주가)가 내재가치에 수렴한다는 전제

- 내재가치
  - 양적 분석: 재무제표, 주가와 손익, 산업자료 등 수치 자료
  - 질적 분석: 경기, 산업동향, 노사문제, CEO의 능력 등

- 기본 분석의 한계
  - 재무제표의 데이터는 시의성이 떨어진다 (년간 4회)
  - 시장가치와 내재가치의 괴리가 지나치게 오래 지속될 수 있다

기술 분석 Technical analysis

과거데이터를 근거로, 추세 예측.

차트를 들여다 보고 있다면, 기술 분석을 하고 있다고 보면 된다.

- 기술 분석의 가정
  - 가격에는 모든 정보가 반영되어 있다.
  - 가격은 일정한 추세로 움직인다.
  - 가격의 움직임은 반복된다.

- 기본 기술 분석 비판(한계)
  - 예측 가능성에 대해서는 논란 (그럼에도 불구하고 "추세"를 파악하는 도구로 사용)
  - 수치와 차트를 사용하기 때문에 객관적인 것 처럼 보이지만 해석이 주관적.

정서 분석 Sentiment analysis, Opinion mining

(자연어 처리, 텍스트 분석 등을 통해) 주관적인 정보를 추출

- 정서 분석 이란?
  - 대상에 대한 (긍정적/부정적) 표현을 분석하여 의사결정에 활용
  - 투자자들의 두려움이나 욕심을 수치화 활용

- 정서 분석 관련 연구
  - 텍스트 마이닝
  - 뉴스 마이닝
  - 어휘 트렌드 분석
  - 소셜 정서 분석

• 데이터 마이닝은 대용량의 데이터 속에서 유용한 정보를 발견하는 과정

증권 분석 데이터(1/2)

- 종목 코드 (심볼 목록)
  - ^KS11 : KOSPI Composite Index (거래소 지수)
  - ^KQ11 : KOSDAQ Composite Index (코스닥 지수)
  - 종목코드에 .KS 가 붙는다. 예를 들어, 삼성전자의 심볼은 005930.KS
- 종목 기본 정보, 상태 기본분석
- 재무정보 (주요 투자지표) 기본분석
  - PER: 주가 / 주당순익(EPS)
  - EPS: 순이익 / 주식수
  - PBR: 주가 / 주당 순
  - BPS: 순자산/ 주식수

증권 분석 데이터(2/2)

- 주가 데이터 (틱,초,분,일,주,월) - 초,분,**일** 기술분석
  - 일자 별 주가: 전체, 과거 10년치 이상
  - 기관외국인동향: 일자 별 매수,매도
- 뉴스, SNS 텍스트 등 비정형 데이터 정서분석
  - 공시 전체, 뉴스(과거 6개월)

데이터 제공 사이트

로그인도 필요 없는 TOP 5

- 1. KRX 한국거래소 http://krx.co.kr
  - 증권 시장 전체 데이터, 증권 통계, 표준코드 시스템
- 2. 전자공시시스템(금융감독원) DART http://dart.fss.or.kr
  - 기업 공시, 재무제표
- 3. 포탈 증권 (네이버, Daum) http://stock.naver.com
  - 종목 정보 상세, 뉴스, 투자 정보
- 4. 야후, 구글 파이낸스 http://finance.yahoo.com
  - 세계/미국 증시, 뉴스, 종목 정보
- 5. 연방준비은행 경제 데이터 http://research.stlouisfed.org/fred2
  - 거시 분석, 해외 시황 (포괄적인 경제 데이터)

모든 금융 데이터, 정말 다 중요한가?

트레이딩 스타일에 따라 다르다

Trading Style	매매 간격	보유 기간	비고
Position Trading	장기간	수 개월 ~ 수 년	-
Swing Trading	짧은 기간	몇 일 ~ 몇 주	-
Day Trading	수 시간	하루는 넘기지 않음	오버나잇 포지션 없음
Scalp Trading	초 단타	수초~수분 단위로 수십 번 거래	오버나잇 포지션 없음

- 오버나잇 포지션은 외환시장 또는 금융거래의 Position으로 1영업일을 경과하여 보유하는 경우를 말한다.

출처: https://www.slideshare.net/plusjune/py-con-2014-47762297

1. 금융 데이터 소개
2. **금융 데이터 분석을 위한 파이썬 라이브러리들**
3. 금융 데이터 수집
4. 예측 모델1 (평균회귀모델)
5. 예측 모델2 (머신러닝)
6. 정리

왜 파이썬인가?

- 시스템 트레이딩의 3 요소가 한 개의 언어로 커버

데이터 확보

- **Beautiful-Soup, Request**
  - 데이터 수집
- **SQLite, MySQL**
  - DB

전략(예측모델)개발 및 검증

- Numpy, Pandas
  - 관계 형 데이터베이스 형식의 숫자를 다루는 강력한 라이브러리
- Statsmodel
  - 통계
- Scikit-learn
  - 머신 러닝
- Matplotlib
  - 그래픽

이행 (주문/계좌관리)

- 증권사 API
  - Win32 com object 활용
- Flask
  - 간단하고 빠른 웹 개발

출처:pycon 2016

- 생산성

금융 데이터 분석을 위한 Python 라이브러리들

- 데이터 분석
  - Pandas: 금융 데이터 라이브러리
  - NumPy: 과학 계산 라이브러리
  - Matplotlib: 그래프 라이브러리
  - Beautiful Soup: 데이터 수집용 라이브러리
  - Scikit-learn: 머신러닝 라이브러리
  - Statsmodels: 통계 라이브러리

pandas

- panel data analysis 다차원 구조화된 데이터 (계량경제학)
- 데이터 분석 라이브러리 (데이터 분석, 클리닝, 모델링 등)
- 시작(2009년)은 **금융 데이터 분석을 위해 설계**
- 시계열 데이터 기능 통합
- R 혹은 Matlab 사용자들이 접근하기 용이

주가 데이터 수집 (pandas)

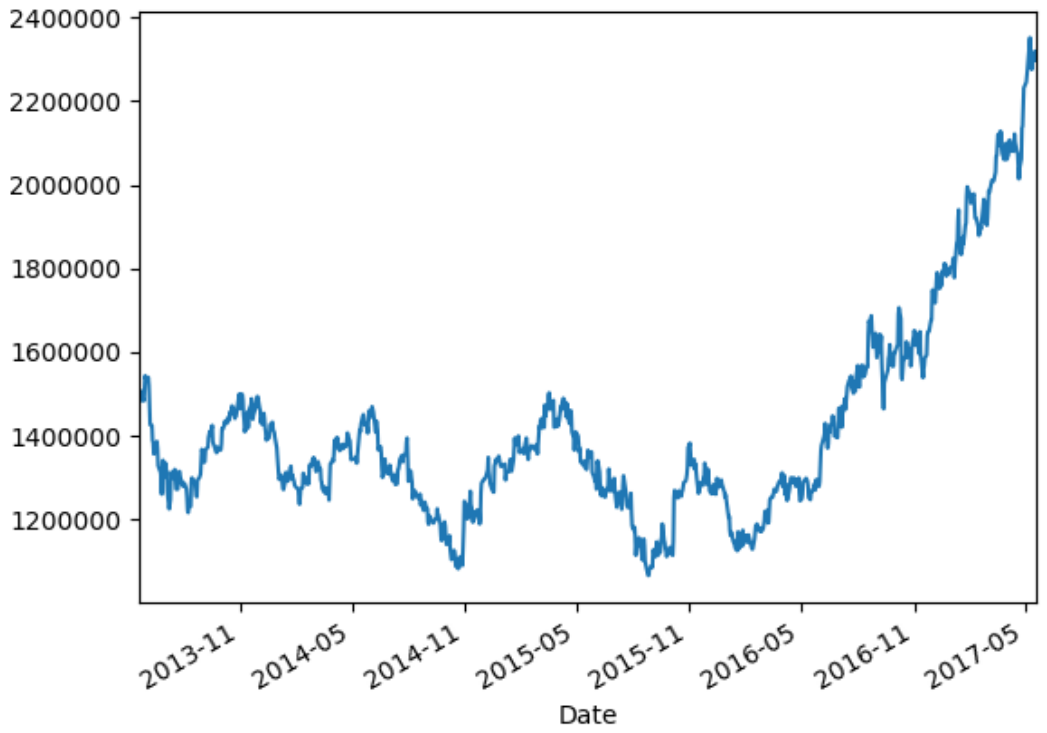
```
import pandas_datareader.data as web

def download_stock_data(file_name, company_code, start, end):
    df = web.DataReader("%s.KS" % (company_code), "yahoo", start, end)
    df.to_pickle(file_name)

download_stock_data('삼성전자', '005930', 2013101,20170601)
```

삼성전자 종가 (pandas)

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_pickle('삼성전자')
print(df)
plt.plot(df['Close'])
plt.show()
```

Date	Open	High	Low	Close	Volume
2013-05-20	1502000.0	1512000.0	1496000.0	1497000.0	242797
2013-05-21	1510000.0	1510000.0	1485000.0	1492000.0	218091
2013-05-22	1504000.0	1510000.0	1491000.0	1509000.0	192913
2013-05-23	1492000.0	1502000.0	1484000.0	1484000.0	232965

…



1. 금융 데이터 소개

2. 금융 데이터 분석을 위한 파이썬 라이브러리들

3. 금융 데이터 수집

4. 예측 모델1 (평균회귀모델)

5. 예측 모델2 (머신러닝)

6. 정리

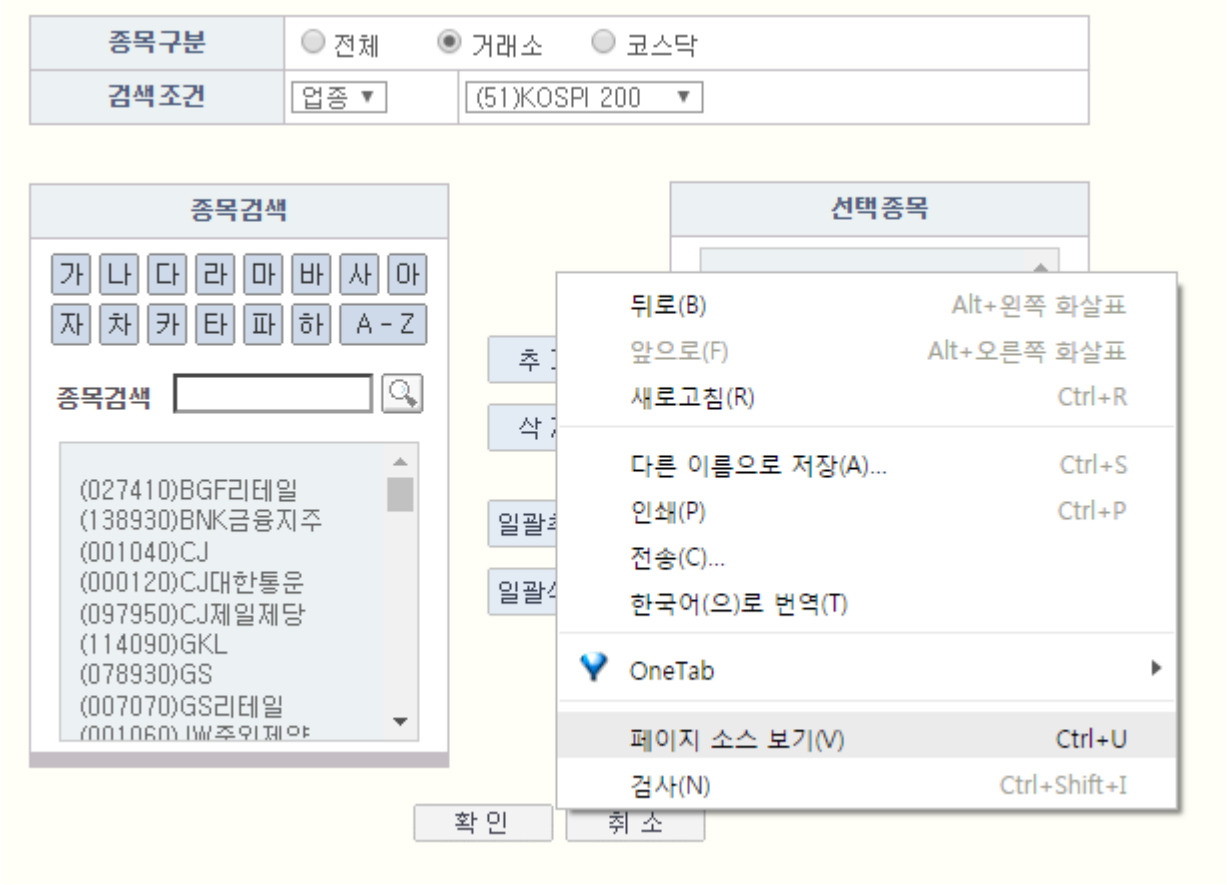
KOSPI200 종목 데이터 저장

데이터 제공 사이트 (koscom, google)

1. **(Requests) HTML 문서 받기** (kospi200가 포함된 페이지)
2. **(BeautifulSoup) HTML 문서 파싱** (kospi200 종목 코드)
3. **(Pandas) 각 종목의 주가 데이터 수집** (kospi200 코드 별 주가)

종목 코드 수집

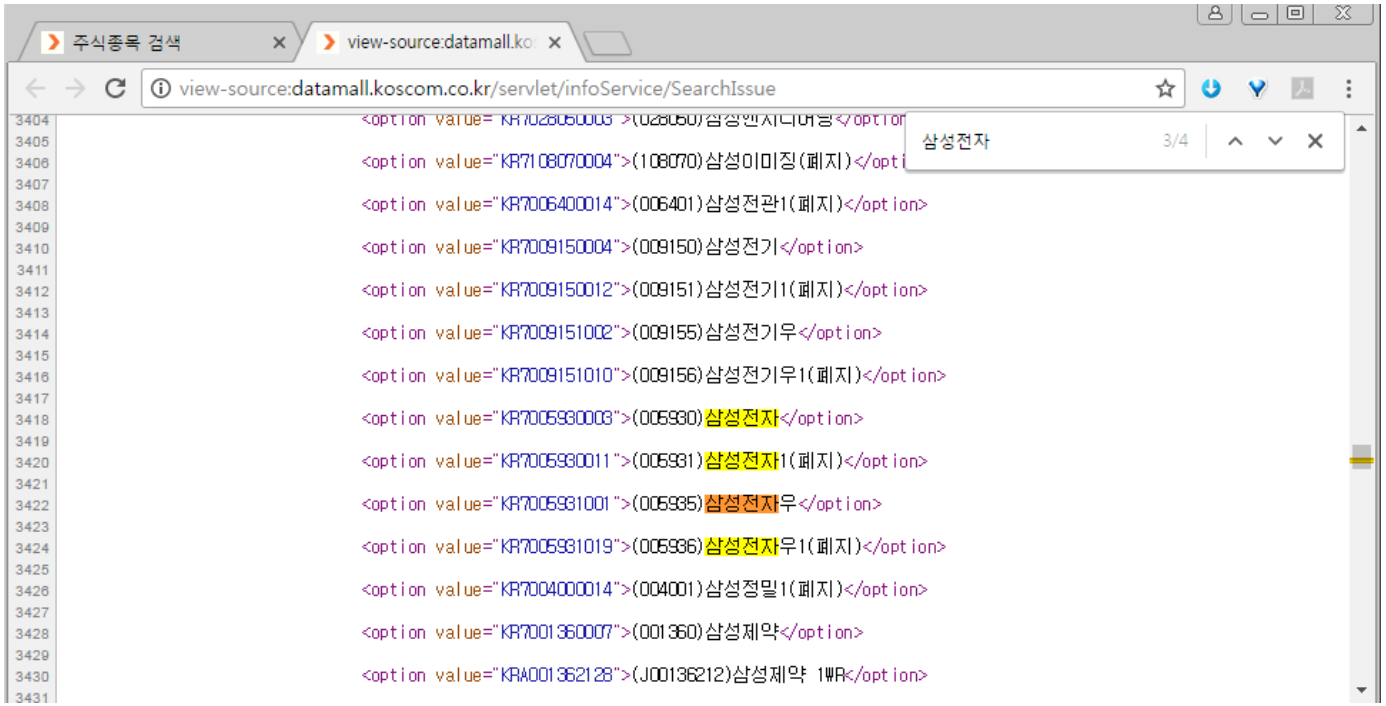
코스콤 (koscom) http://datamall.koscom.co.kr/servlet/infoService/SearchIssue



종목 코드 수집

Request으로 데이터 수집

```
import requests
def downloadCode(self, market_type,whereCode):
    url = 'http://datamall.koscom.co.kr/servlet/infoService/SearchIssue'
    html = requests.post(url, data={'flag': 'SEARCH','marketBit': market_type})
    return html.content
```



종목 코드 수집

Beautiful-Soup 으로 데이터 파싱

```
def parseCodeHTML(self, html):
    soup = BeautifulSoup(html, "lxml")
    options = soup.findAll('option')
    …
    for a_option in options:
        …
        code = a_option.text[1:7]
        company = a_option.text[8:]
        full_code = a_option.get('value')
        …
        codes.add(code, full_code, company)
  return codes
```

```
…
<option value="KR7005930003">(005930)삼성전자</option>
<option value="KR7005930003">(005930)삼성전자</option>
<option value="KR7005930003">(005930)삼성전자</option>
…
```

주가 데이터 수집

pandas 으로 수집

```
def download_stock_data(self, file_name, company_code, start, end):
    try:
        df = web.DataReader("KRX:%s" % (company_code), "google", start, end)
        save_stock_data(df, file_name)
    except Exception as ex:
        print('except [%s] %s' % (company_code, ex))
        return None
    return df
```

Save File:005930_삼성전자.data (pandas.DataFrame)

....

h□Y□R□□□G□□□B)□0□□.□K□□/□W□k"□V□□>□ E□□X□gBE□%.□□|□N□2>□□□whn□5T□?□^□[7□?□ê□□|Ez□
□□5□n□□□□□uU□□□^□~=□)□□□□□□□ne□8□□□□c□"#□%□□hu□a□ ; □□□□ □3□ N□□&!;a□"□□a"□□□b□IY

....

데이터 수집 결과

kospi200 주가 데이터

000030_우리은행.data	002240_고려제강.data	005180_빙그레.data	009290_광동제약.data	020000_한섬.data
047050_포스코대우.data	103140_풍산.data	000050_경방.data	002270_롯데푸드.data	005300_롯데칠성.data
001800_오리온.data	104700_한국철강.data			
….				
005090_삼광글라스.data	009240_한샘.data	019680_대교.data	047040_대우건설.data	097950_CJ제일제당.data

	Open	High	Low	Close	Volume
count	2.460000e+02	2.460000e+02	2.460000e+02	2.460000e+02	2.460000e+02
mean	1.745577e+06	1.765902e+06	1.732305e+06	1.751106e+06	2.405352e+05
std	2.583176e+05	2.599001e+05	2.573962e+05	2.581271e+05	1.162698e+05
min	1.280000e+06	1.297000e+06	1.268000e+06	1.280000e+06	8.364400e+04
25%	1.554250e+06	1.575000e+06	1.545000e+06	1.559500e+06	1.699208e+05
50%	1.663000e+06	1.685500e+06	1.638000e+06	1.676000e+06	2.156975e+05
75%	1.950750e+06	1.977250e+06	1.946250e+06	1.963500e+06	2.726378e+05
max	2.333000e+06	2.361000e+06	2.305000e+06	2.351000e+06	1.198180e+06

1. 금융 데이터 소개

2. 금융 데이터 분석을 위한 파이썬 라이브러리들

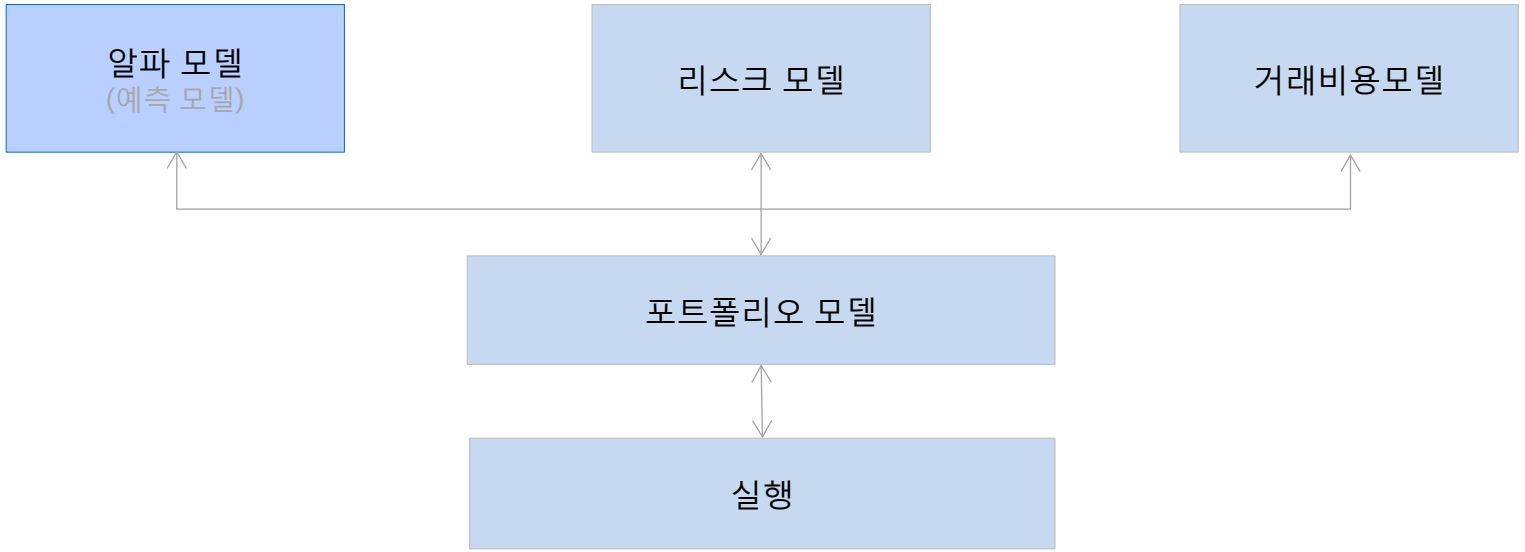
3. 금융 데이터 수집

4. 예측 모델1 (평균회귀모델)

5. 예측 모델2 (머신러닝)

6. 정리

일반적인 알고리즘 트레이딩 시스템 구성



알파 모델(예측모델) 주가나 주가 방향 등을 예측하기 위한 모델

리스크 모델 거래했을 때 예측이 틀렸다면 어느 정보의 손실을 보는지 등 위험도를 측정하는 모델

거래비용 모델 실제로 거래하면 그에 따른 수수료나 세금 등의 비용을 계산하는 모델

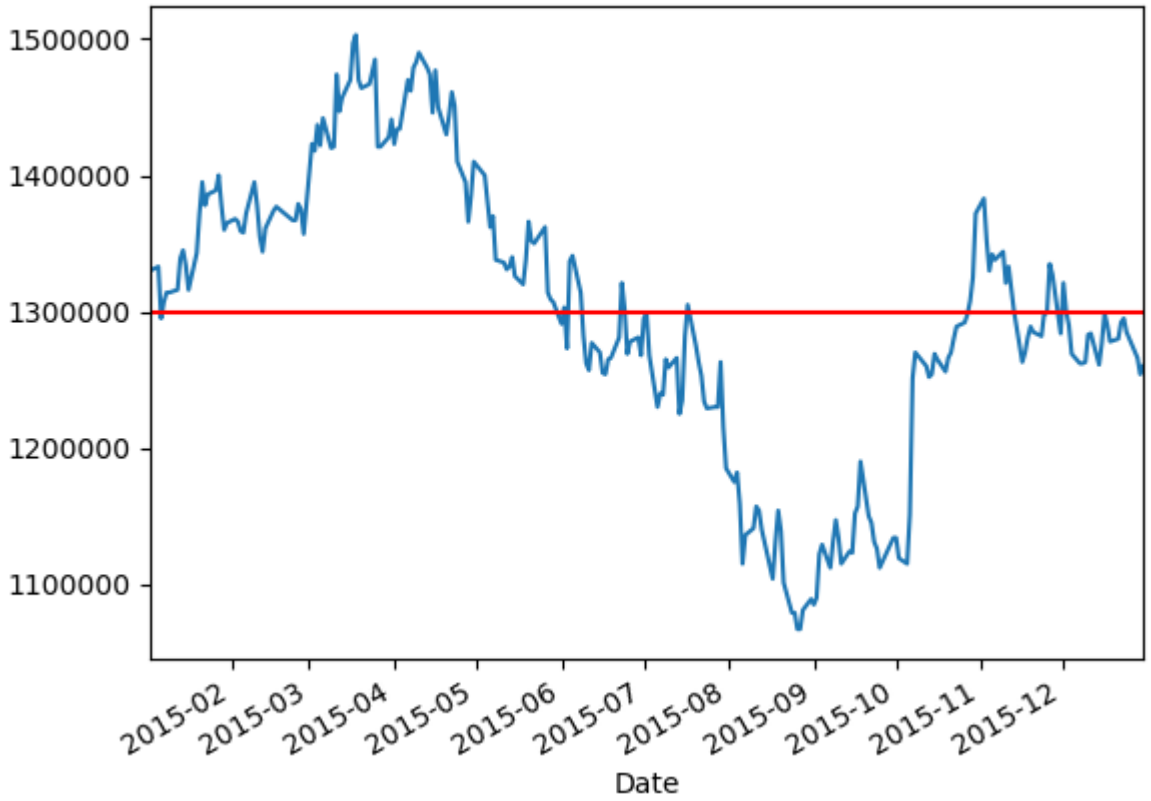
포트폴리오 모델 위 3가지 모델의 결과로 최종 거래 여부 결정 및 거래의 규모를 산정하는 모델

실행 포트폴리오 모델의 결정에 따라 실제로 거래

평균회귀 모델 사전 지식

시계열 데이터(time-series data)란?

- **시간이라는 독립변수에 의해 변화하는 데이터**
  - 데이터의 순서가 있고, 그 값이 지속적으로 변화한다.
  - 주가 데이터, 온도 데이터, 상품판매량, 환율 데이터 등



평균회귀 모델 사전 지식

시계열 데이터의 특성

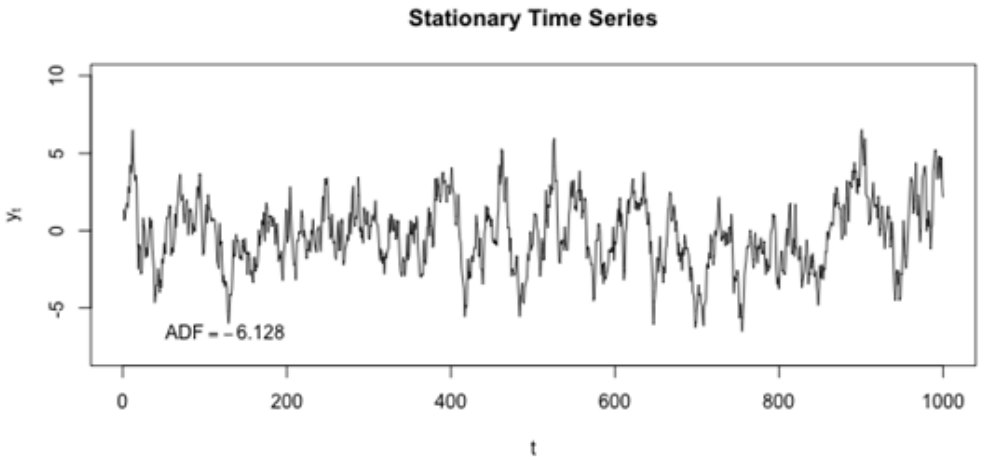
Trend: 측정값이 시간의 흐름에 따라 증가나 감소 또는 반복 등의 일정한 패턴이나 경향

Seasonality: 일, 월, 년, 계절 등 일정 시간에 따라 지속해서 반복되는 패턴이 있는가?

…

정상성: 데이터가 증가하거나 감소하지만, 평균값인 0을 기준으로 움직이는 패턴

과거의 데이터와 미래의 데이터가 유사한 모습을 보이는 성질, 이런 데이터의 분포를 정상성



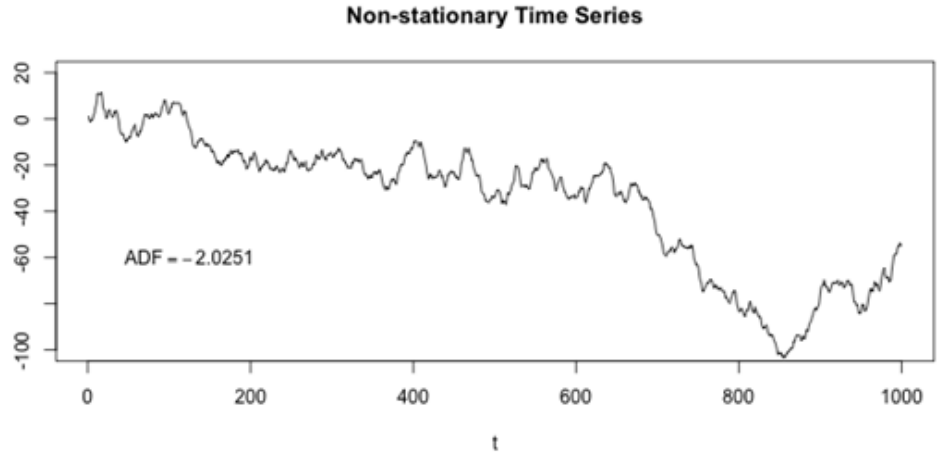


사진 출처: https://en.wikipedia.org/wiki/Stationary_process

IDEA

평균회귀모델은 주식가격이 장기적으로 평균으로 수렴한다는 것을 가정한다.

“ 주식 데이터는 시계열 데이터이고,

시계열 데이터의 특성 중 정상성이 있는 종목들만 찾아서

많이 떨어진 시점에서 주식을 사고 오르면 팔면 되겠네!!”

“그럼 정상성을 판별하기 위해 많이 사용하는 방법들?”

평균회귀 테스트

많이 사용하는 판단 방법

- **ADF**(augmented Dickey-fuller) 테스트
- **허스트**(Hurst Exponent) 지수
- **평균회귀의 Half-life**

ADF(augmented Dickey-fuller) 테스트

“어떤 시계열 데이터가 랜덤워크를 따른다는 가설을 세우고,

이 가설이 맞으면 랜덤워크, 아니면 평균회귀라는 결론을 얻게 된다”

- 랜덤워크는 다음 행보가 이전 행보에 영향을 받지 않는 독립적인 사건

```
import statsmodels.tsa.stattools as ts

df = load_stock_data('005930_삼성전자.data')
adf_result = ts.adfuller(df["Close"])
pprint.pprint(adf_result)
```

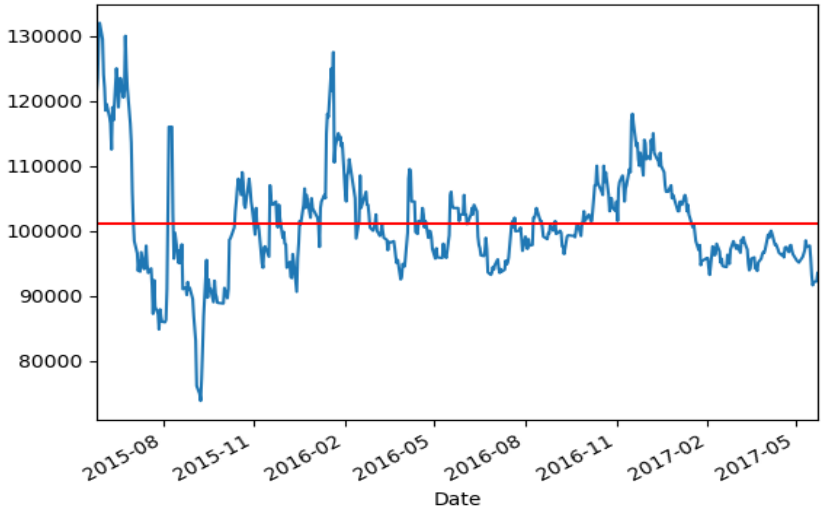
```
(-2.9483436516372885,                  검정통계량(Test statistic)
 0.04000821269323322,                  p-value
 3,
 739,                                  테이터 수
 {'1%': -3.439229783394421,            1% 기각값(Critical Value)
  '10%': -2.5688568756191392,          10% 기각값(Critical Value)
  '5%': -2.8654589481476198},          5% 기각값(Critical Value)
 16362.693071135693)
```

검정 통계량 값이 1%, 5%, 10% 값 중 어느 하나보다도 작으면 평균 회귀

ADF(augmented Dickey-fuller) 테스트

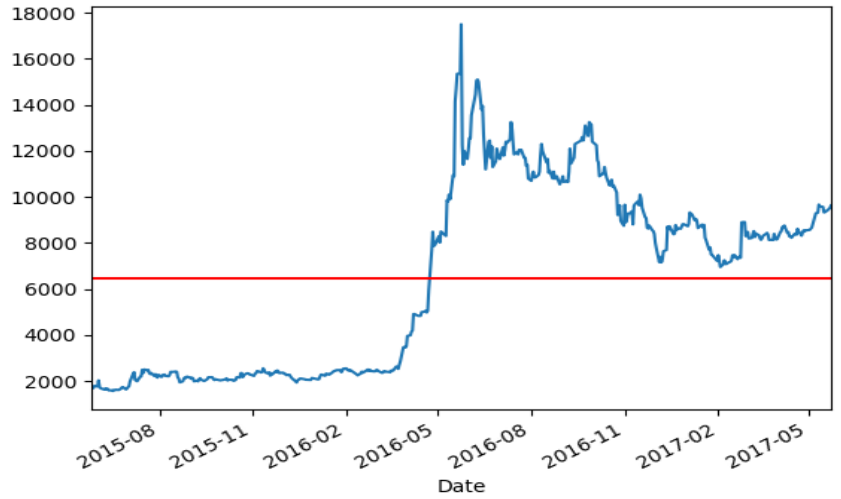
- **적용 가능** (145990_삼양사)

```
(-4.1641663614307367,
 0.0007572222682829038 1,
 7,
 485,
 {'1%': -3.443905150512834 2,
  '10%': -2.569953900520777 9,
  '5%': -2.867517732199813 1},
 8832.343866041879 7)
```



- **적용 불가능** (003520_영진약품)

```
(-1.4511459207313071,
 0.5575256529740336 7,
 17,
 475,
 {'1%': -3.444192086326286 3,
  '10%': -2.570021186703601 1,
  '5%': -2.867643981361714 7},
 7052.661212513412 5)
```



허스트(Hurst Exponent) 지수

"주가(시간에 따른 주식값의 변화)의 확산속도를 수학적 계산을 통해 알 수 있고,

그 값을 GBM (기하적 브라운 운동) 속도와 비교하면 랜덤워크인지 판별 가능

즉, GBM보다 천천히 값이 퍼져나가면 정상과정이다

```
Import numpy as np
def get_hurst_exponent(self,df, lags_count=100):
    lags = range(2, lags_count)
    ts = np.log(df)
    tau = [np.sqrt(np.std(np.subtract(ts[lag:], ts[:-lag]))) for lag in lags]
    poly = np.polyfit(np.log(lags), np.log(tau), 1)
    return poly[0] * 2.0def hurst_exponent(self):

def hust_exponent(self)
    hurst = self.get_hurst_exponent(self.df['Close'])
    print("Hurst Exponent : %s=%s" % (self.code, hurst))
    return hurst
```

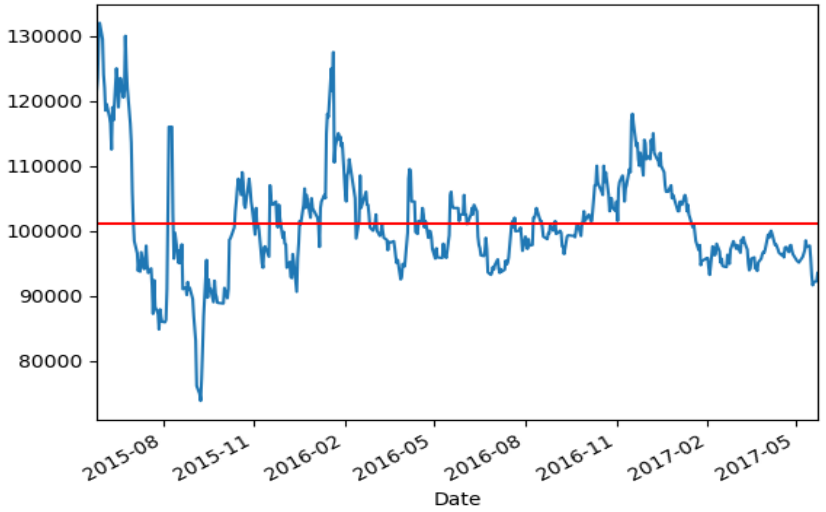
Hurst Exponent: 0.239489239801

H < 0.5면 평균회귀, H > 0.5면 추세 성향, 값이 0에 가까울수록 평균회귀 성향이 강하다

허스트(Hurst Exponent) 지수

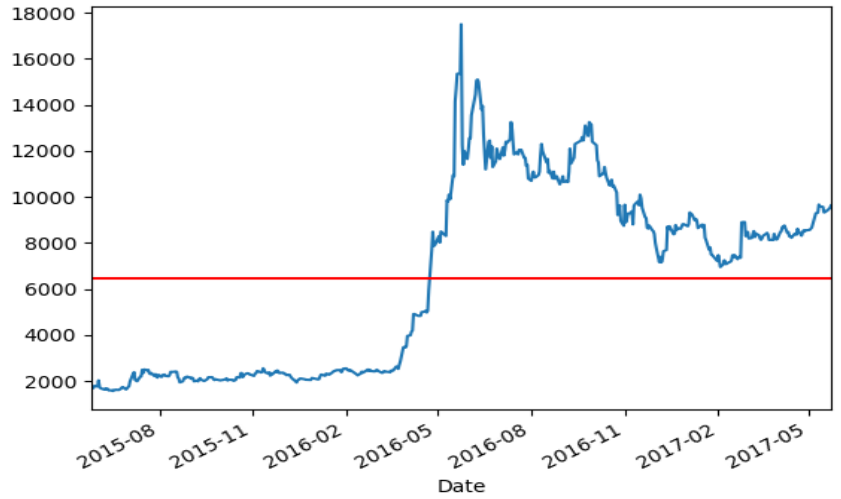
- **적용 가능** (145990_삼양사)

Hurst Exponent: 0.239489239801



- **적용 불가능** (003520_영진약품)

Hurst Exponent: 0.648439318562



평균회귀의 Half-life

Half-life값은 평균으로 회귀하는데 걸리는 시간

값이 크다는 것은 장기간 지속하는 경향이 있다는 것을 의미할 수도 있고,
반대로 작다는 것은 그만큼 주식값의 변동이 잦다는 것으로 해석

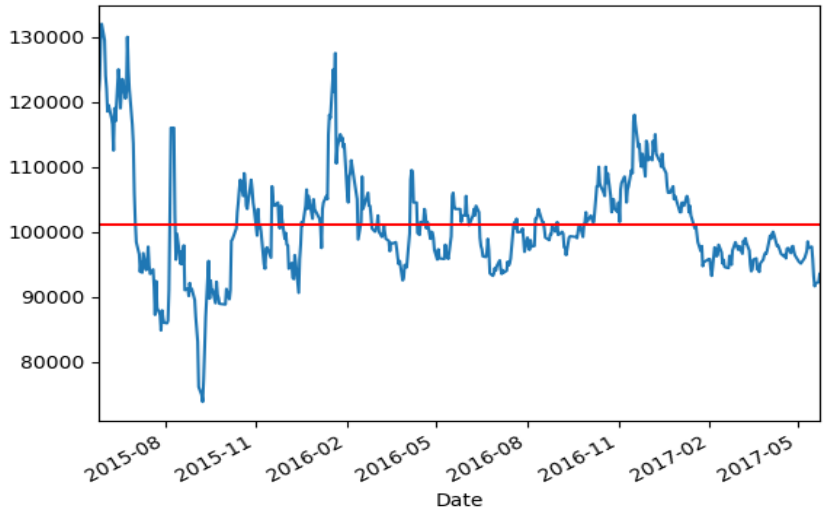
```
import pandas as pd

def get_half_life(df):
    price = pd.Series(df)
    lagged_price = price.shift(1).fillna(method="bfill")
    delta = price - lagged_price
    beta = np.polyfit(lagged_price, delta, 1)[0]
    return (-1 * np.log(2) / beta)def half_life(df):

def half_life(df):
    half_life = get_half_life(df['Close'])
    print("Half_life: %s" % (half_life))
```

```
Half_life: 11.4972629258
```

평균회귀의 Half-life

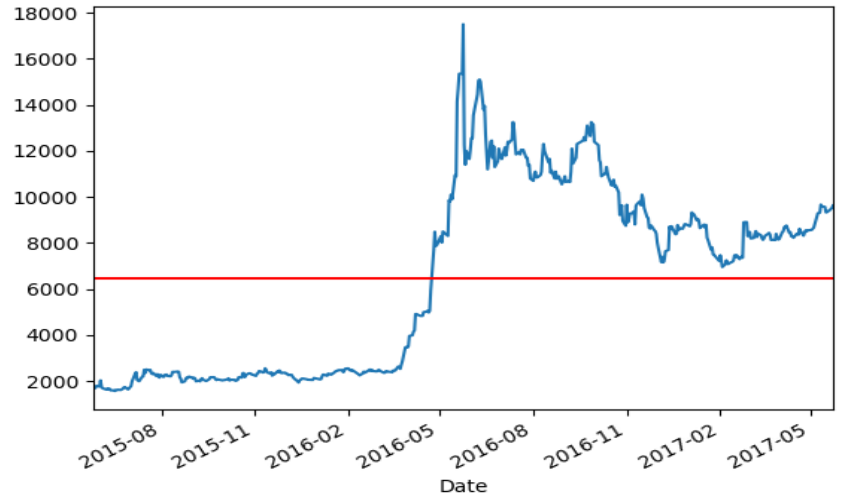
- **적용 가능** (145990_삼양사)

Half_life: 11.4972629258



- **적용 불가능** (003520_영진약품)

Half_life: 111.772452834



평균회귀 모델의 구현

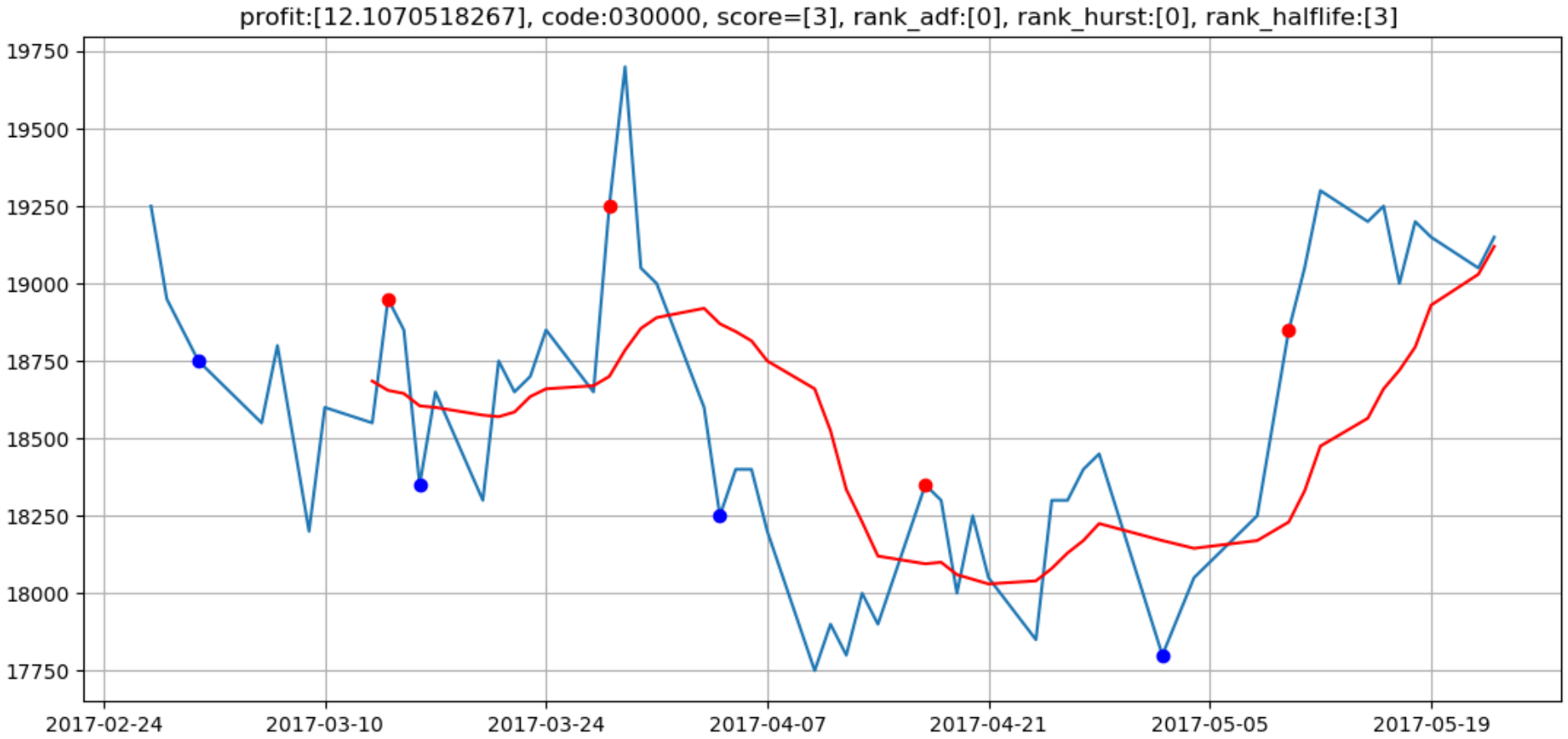
- **평균회귀 모델의 기본 개념**
  - 평균회귀 특성이 있는 주식을 찾고
  - 주가가 평균보다 낮으면 주식을 매입
  - 반대로 평균보다 높으면 주식을 매도
- **3가지 결정 요소**
  1. 이동 평균 정의: 5일, 10일, 30일 … 1년?
  2. 매도, 매수 기준: 평균보다 높아진 순간? 평균과 주가의 차이가 2배?
  3. 데이터 선택: Close?, Open?, Adj Close? …

평균회귀 모델의 알고리즘

- **평균 정의** 10일 이동평균
- **매도, 매수 기준** 10일 이동평균과 주가의 차이가 표준편차보다 크면 주문
  - |주가 – 이동평균| > 표준편차일 때
  - 주가 –이동평균 > 0, 매도
  - 주가 –이동평균 < 0, 매수
- **데이터 선택** 종가 사용

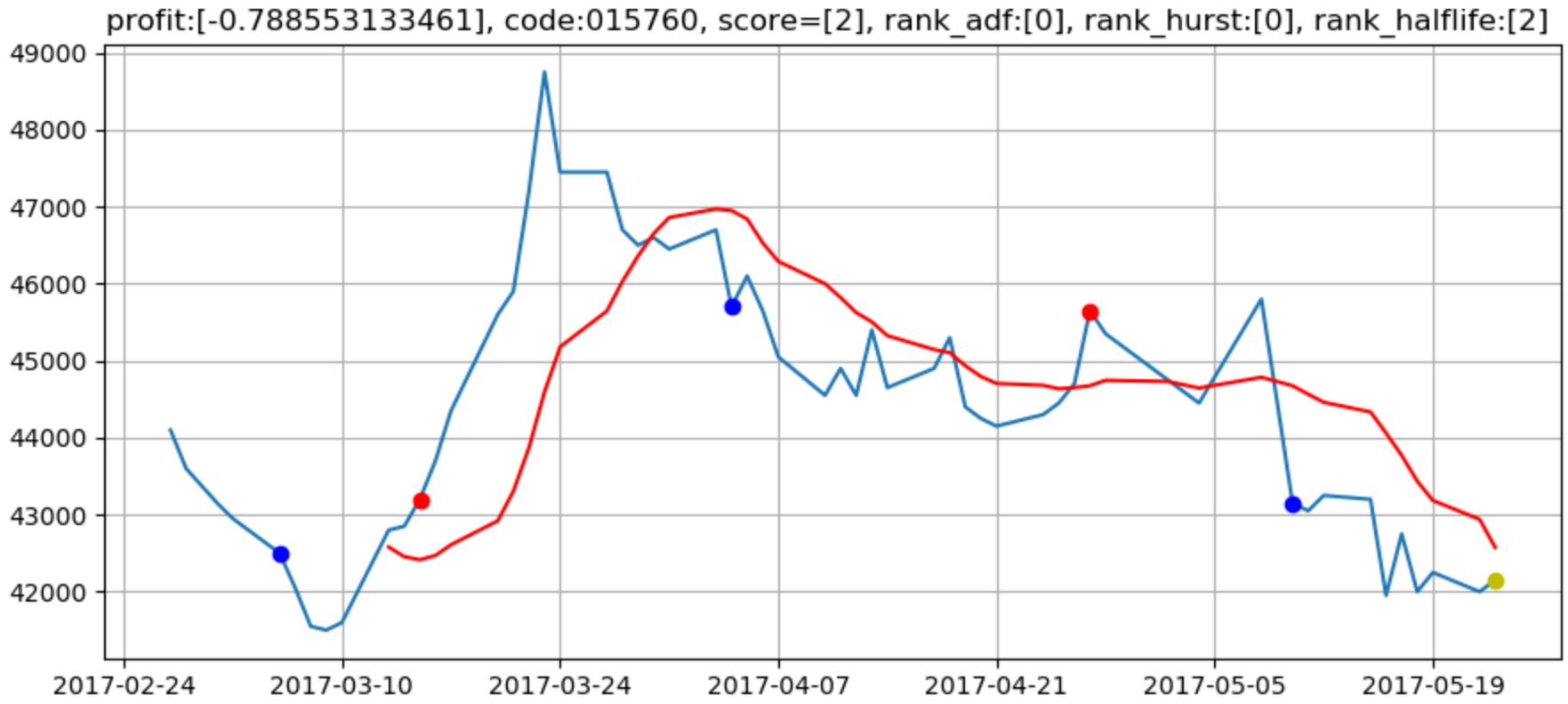
```
diff = pd.rolling_mean(df['Close'], window) - df['Close']
if np.abs(diff) > pd.rolling_std(df['Close'], window):
   if diff[len(df) - 1] > 0:
      sell_stock()
   else:
      buy_stock()
```

평균회귀 모델의 결과



빨간점: 매도
파란점: 매수

평균회귀 모델의 결과



빨간점: 매도
파란점: 매수

평균회귀 모델의 결과

Kospi200 중 8개 종목에서 매수.매도

기간: 2017-05-10 ~ 2017-05-25

수익률: 4.3

[011780] total: 3.05444887118, profit: 3.05444887118, [2017-05-10 ~ 2017-05-22]
[078930] total: 8.67625466335, profit: 5.62180579216, [2017-05-10 ~ 2017-05-12]
[023530] total: 11.0200046633, profit: 2.34375, [2017-05-11 ~ 2017-05-12]
[036570] total: 17.7151613585, profit: 6.69515669516, [2017-05-12 ~ 2017-05-17]
[000210] total: 20.5794203871, profit: 2.86425902864, [2017-05-17 ~ 2017-05-19]
[004020] total: 24.0784995768, profit: 3.49907918969, [2017-05-17 ~ 2017-05-19]
[010130] total: 25.4969392931, profit: 1.41843971631, [2017-05-17 ~ 2017-05-22]
[004000] total: 34.5636059598, profit: 9.06666666667, [2017-05-18 ~ 2017-05-19]
TOTAL:4.32045074498

평균회귀 모델의 결과

같은 기간 동안 Kospi200을 사고 들고 있었다면??

기간: 2017-05-10 ~ 2017-05-25

수익률: 1.8

[000100][0/200] total: -2.07468879668, profit: -2.07468879668,
[020150][1/200] total: 14.6792379049, profit: 16.7539267016,
[185750][2/200] total: 11.8337094496, profit: -2.84552845528,

…
[000150][197/200] total: 368.792029035, profit: 12.9032258065,
[004170][198/200] total: 376.945506253, profit: 8.15347721823,
[008490][199/200] total: 367.937673355, profit: -9.00783289817,
TOTAL:1.8396883667

평균회귀 모델의 결과

Kospi200이 떨어지는 기간 동안 측정

기간: 2015-07-01 ~ 2016-01-01

Kospi200 수익률: -7.4

평균회귀 수익률: -0.2

Kospi200이 오르는 기간 동안 측정

기간: 2016-05-25 ~ 2017-05-25

Kospi200 수익률: 4.5

평균회귀 수익률: 6.7

사진 출처: naver

1. 금융 데이터 소개

2. 금융 데이터 분석을 위한 파이썬 라이브러리들

3. 금융 데이터 수집

4. 예측 모델1 (평균회귀모델)

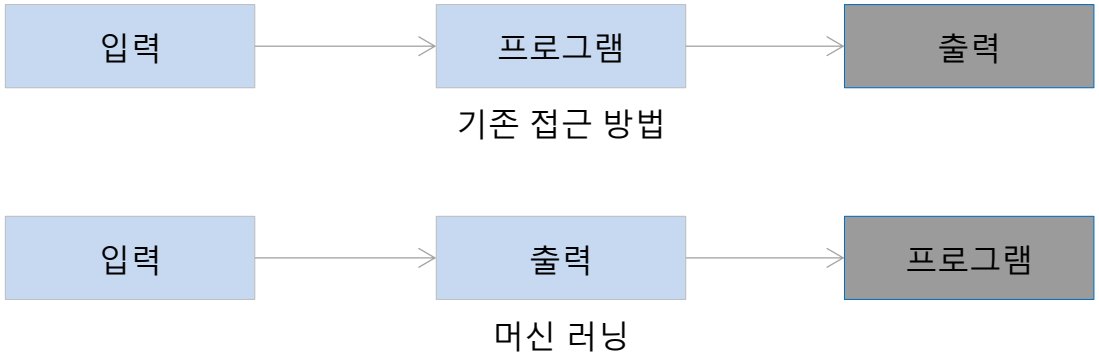
5. 예측 모델2 (머신러닝)

6. 정리

사전 지식(머신 러닝 모델)

*"머신 러닝은 인공지능의 **패턴인식과 계산학습 이론**에서 발전한 컴퓨터 과학의 한 분야이다*
*머신 러닝은 **데이터**로부터 **학습**하고, **예측**할 수 있는 알고리즘을 연구한다.*
이러한 알고리즘은 정적으로 주어진 프로그램이 아닌
*입력된 데이터로부터 **모델**을 만들어 예측이나 결정을 내린다."*

출처: https://en.wikipedia.org/wiki/Machine_learning



사전 지식(머신 러닝 모델)

- “최신 머신 러닝기술을 금융투자에 적용하면 높은 수익률을 만들 수 있을까?”
  - 대답은 “가능하다. 단 적절히 잘 사용한다면” (머신 러닝이 무작정 좋은 결과를 만들어 내지는 않는다)

- **머신 러닝의 핵심은 데이터이다.** **(안명호 딥넘버스대표)**

  “ 머신 러닝의 알고리즘은 입력 값 간의 관계를 나름의 방법을 통해 유추하고, 모델을 완성하기 때문에 출력변수에 영향을 미치는 입력변수가 많을 수록 (반드시 많다고 좋은 것은 아니다) 예측력이 높아진다.”

출처: 안명호의 인공지능과 미래금융

사전 지식(머신 러닝 모델)

- 미래 예측의 전제 '정상성' ("**과거**에 발생했던 일은 **미래**에도 발생한다"이다.)
  - 만약 과거에 보였던 특성이 미래에 동일 혹은 유사하게 나타나지 않는다면 과거의 데이터를 **학습**하는 것은 아무런 의미가 없다.

- **금융 데이터는 정상성을 가지고 있을까요?**
  - 그 동안 수많은 사람이 연구하고 논의한 결과 다수 설은 바로 "정상성이 없다"이다.

- **그럼에도 불구하고**
  1. 정상성이 있는 종목만 학습시킨다면,
  2. 정상성이 없는 데이터라도 정상성이 있도록 데이터를 변환한다면,

출처: 안명호의 인공지능과 미래금융

사전 지식(머신 러닝 모델)

- **가격이냐 방향이냐**(무엇을 예측할 것인가?)
  - 가격(오늘 1만원인 주가가 내일은 얼마가 될지?)의 예측이면 **회귀**
  - 방향(내일 주가가 올라갔지 내릴지)으로 구분한다면 **분류**
    - 출력 데이터: 1(상승), 0(보합), -1(하락)
    - 입력 데이터: 주가, 거래량, 지수 …

- **대표적인 분류 모델**
  - **로지스틱 회귀**, **SVM**(서포트 벡터 머신), **랜덤 포레스트**, 의사결정 트리, SGD 분류기 …

출처: 안명호의 인공지능과 미래금융

IDEA

머신 러닝의 분류 모델들을 사용하면 내일 주가를 예측율을 알 수 있다.

“우선 얼마나 적중률을 보이는지 확인해 보고서

평균회귀 모델과 같이 사용할 방법을 고민해보자!!

“우선 쉽게 사용할 수 있는 머신 러닝 알고리즘에 뭐가 있지?”

머신러닝 적중률 테스트

많이 사용하는 판단 방법(Scikit-learn)

- **로지스틱 회귀**(Logistic Regression)
- **SVM**(Support Vector Machine)
- **랜덤 포레스트**(Ramdom Forest)

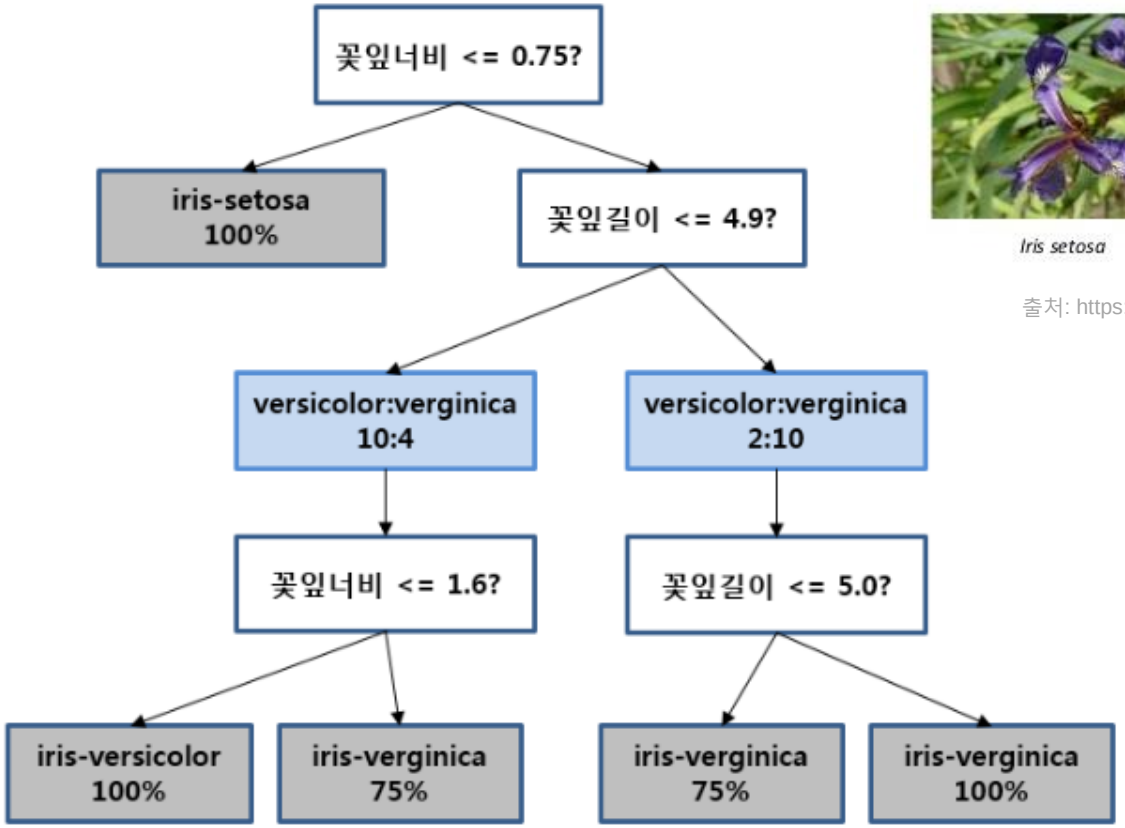
미래 주가 방향의 적중률을 판단

- **출력** 방향(상승, 하락, 보합)
- **입력** 출력과 관련 있을거 같은 데이터

의사 결정 트리(Decision tree learning)

- **기본 아이디어**

  "스무고개와 비슷한 원리로, 일련의 질문에 근거하여 주어진 데이터를 분류한다."

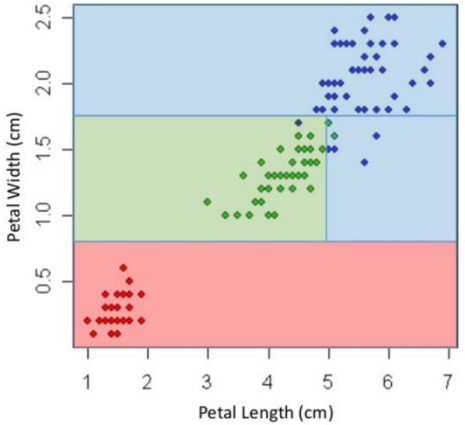


Iris setosa

Iris versicolor

Iris virginica

출처: https://www.slideshare.net/DavidGerster1/gerster-hands-on-machine-learning-v2



출처: http://blog.naver.com/samsjang/220976772778

랜덤 포레스트 (Random Forest)

- **기본 아이디어**

  “여러 개의 의사 결정 트리를 사용하는 앙상블 학습기법입니다.

  서로 다른 의사 결정 트리를 이용해 예측하고, 각각의 예측 값을 통합해 최종적인 예측을 낸다”

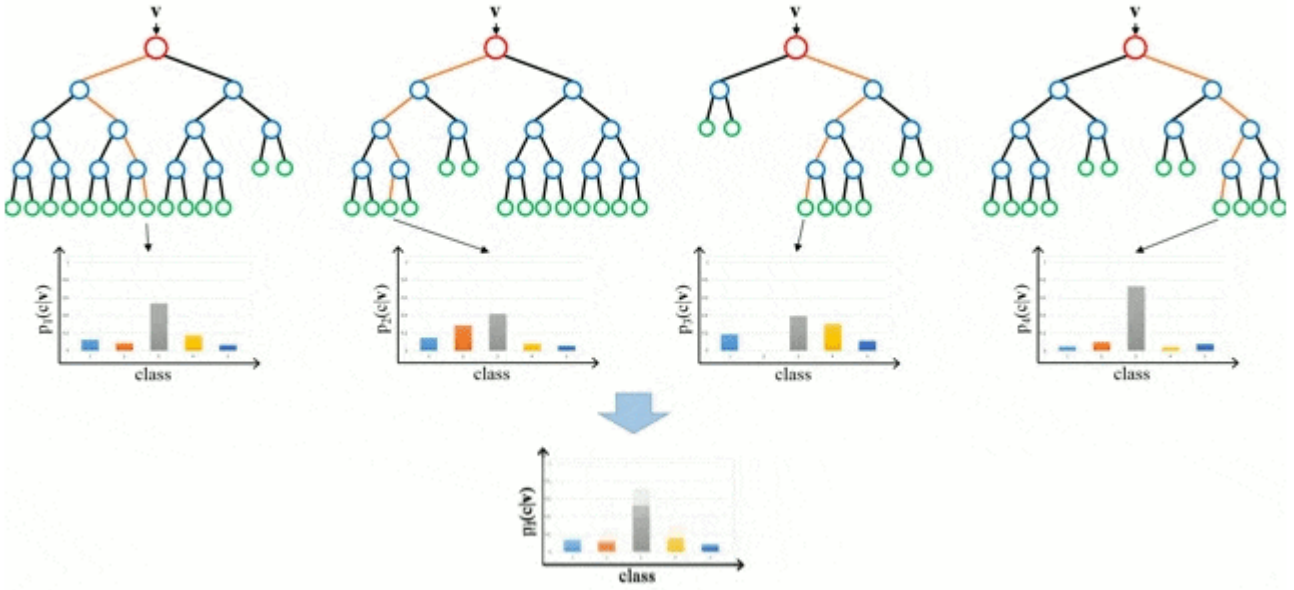


사진 출처: http://blog.naver.com/samsjang/220979751089

SVM(Support Vector Machine)

- **기본 아이디어**

  “주어진 데이터를 분류하기 위한 **최적경계선(support vector)**을 찾고,
  분류할 데이터가 경계선의 어느 쪽에 위치하느냐에 따라 분류한 것이다.”

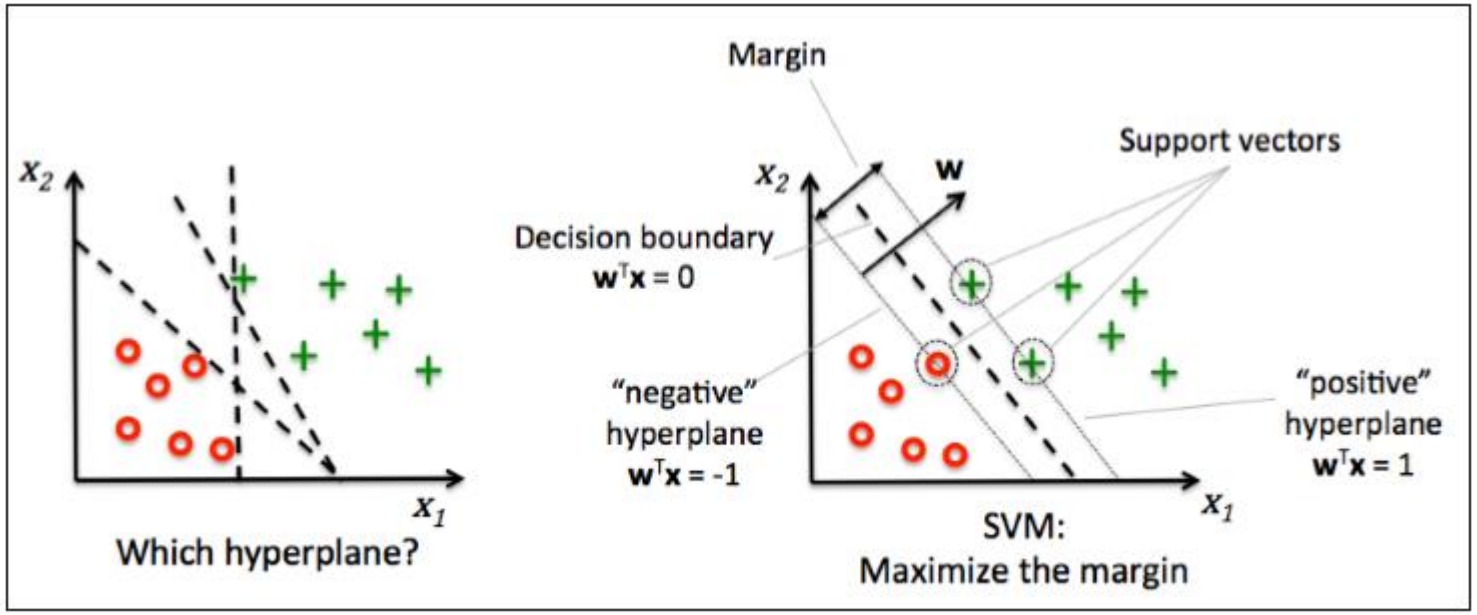


사진 출처: http://m.blog.naver.com/samsjang/220969601609

머신 러닝 모델의 구현

- **Scikit-learn 라이브러리를 이용해 다음 날 주가방향을 예측할 수 있는 예측변수를 구현한다.**

- **머신 러닝 모델의 기본 개념**
  - 정상성이 있는 종목만 학습시킨다.
  - 학습과 테스트에 사용할 데이터 셋을 일정 비율로 나눈다 (8:2)
  - 각각의 머신 러닝을 학습 시키고 테스트의 결과(적중률)를 구한다.
  - 3가지 예측 모델의 결과를 조합한다.

- **데이터 셋**
  - 출력 데이터: 1(상승), 0(보합), -1(하락)
  - 입력 데이터: 주가, 거래량, 지수 … (출력 데이터와 연관이 있다 싶은 데이터)
  - 비율: 학습용(8), 테스트용(2)

데이터 셋 (입출력)

```
def make_dataset(self, df):
  df_lag['Close'] = df['Close']
  df_lag["Close_Lag"] = df['Close'].shift(-1)
  df_lag["Close_Lag_Change"] = df_lag["Close_Lag"].pct_change()*100.0
  return df_lag
```

	Close	Close_Lag1	Close_Lag1_Change	Close_Lag1_Direction
Date				
2016-11-28	5250.0	5250.0	NaN	NaN
2016-11-29	5250.0	5170.0	-1.523810	-1.0
2016-11-30	5170.0	5190.0	0.386847	1.0
...	...	...	...	
2017-05-19	7930.0	8070.0	1.765448	1.0
2017-05-22	8070.0	8120.0	0.619579	1.0
2017-05-23	8120.0	NaN	NaN	NaN

데이터 셋 나누기(학습, 테스트)

```
def split_dataset(self, df, input_column_array, output_column, spllit_date=0.8):
    input_data = df[input_column_array]
    output_data = df[output_column]
    X_train = input_data[input_data.index < split_date]
    Y_train = output_data[output_data.index < split_date]
    X_test = input_data[input_data.index >= split_date]
    Y_test = output_data[output_data.index >= split_date]
    return X_train, X_test, Y_train, Y_test
```

	입력 데이터(X)	출력 데이터(Y)
학습 (train)	2016-11-29 5250.0 2016-11-30 5170.0 ... ... 2017-04-13 6780.0 2017-04-14 6820.0	2016-11-29 -1.0 2016-11-30 1.0 ... 2017-04-13 1.0 2017-04-14 1.0
테스트 (test)	2017-04-17 6880.0 2017-04-18 6910.0 ... ... 2017-05-22 8070.0 2017-05-23 8120.0	2017-04-17 1.0 2017-04-18 1.0 ... 2017-05-22 1.0 2017-05-23 NaN

머신 러닝 학습 전체 코드

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

def do_logistic_regression(x_train, y_train):
    classifier = LogisticRegression()
    classifier.fit(x_train, y_train)
    return classifier

def do_random_forest(x_train, y_train):
    classifier = RandomForestClassifier()
    classifier.fit(x_train, y_train)
    return classifier

def do_svm(x_train, y_train):
     classifier = SVC()
     classifier.fit(x_train, y_train)
     return classifier
```

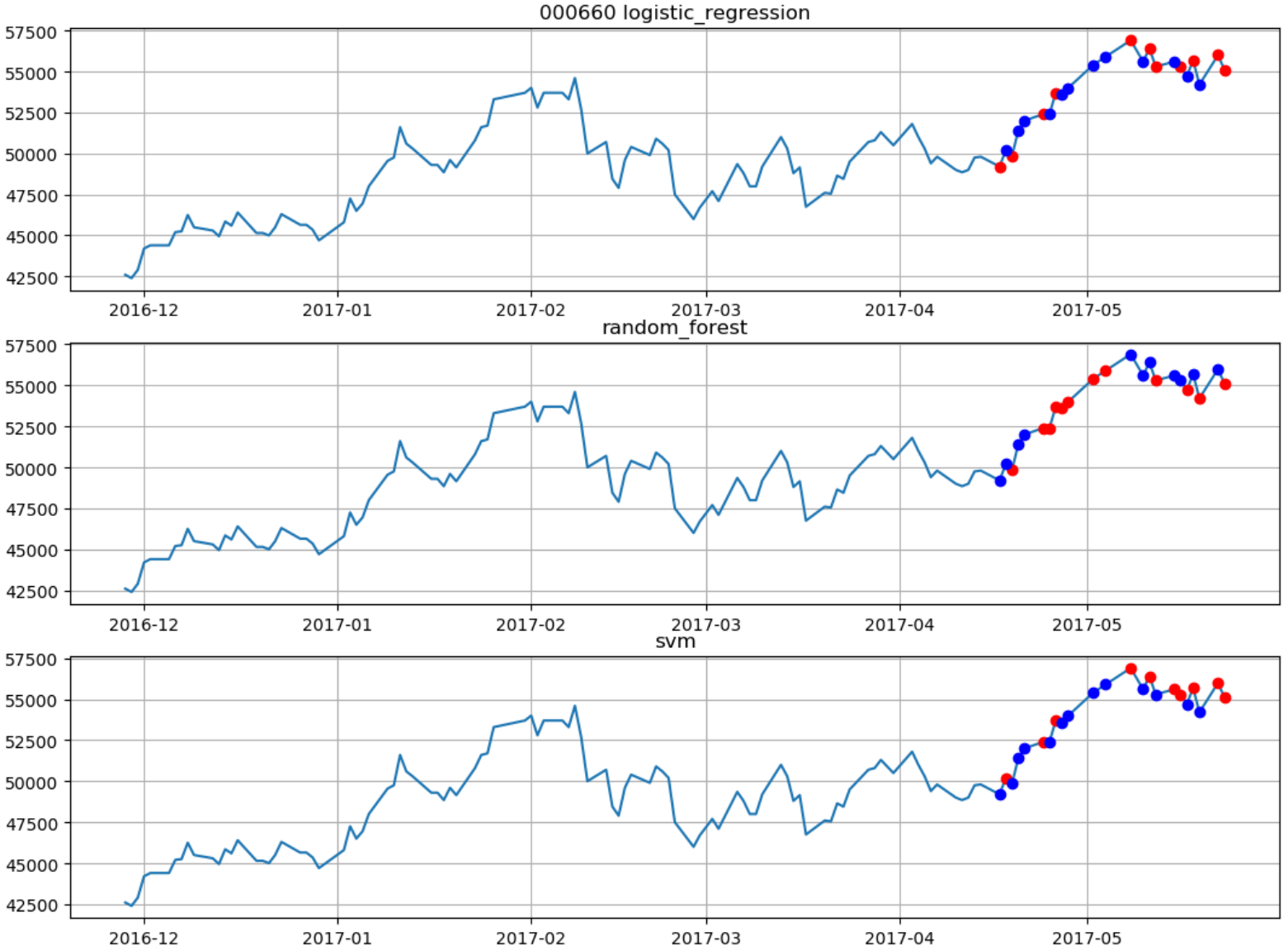
머신 러닝 테스트 실행

```
def test_predictor(classifier, x_test, y_test):
    pred = classifier.predict(x_test)
    hit_count = 0
    for index in range(len(y_test)):
        if (pred[index]) == (y_test[index]):
            hit_count = hit_count + 1

    hit_ratio = hit_count / total_count
    score = classifier.score(x_test, y_test)
    print ("hit_count=%s, total=%s, hit_ratio = %s" % (hit_count,total_count,hit_ratio))
    return hit_ratio, score
```

hit_count=12, total=23, hit_ratio = 0.5217391304347826 [2017-04-17 ~ 2017-05-23]
hit_count=11, total=23, hit_ratio = 0.4782608695652174 [2017-04-17 ~ 2017-05-23]
hit_count=13, total=23, hit_ratio = 0.5652173913043478 [2017-04-17 ~ 2017-05-23]

000660 : Hit Ratio - Logistic Regreesion=**0.52**, RandomForest=**0.48**, SVM=**0.57**

머신 러닝 모델의 결과


000660 logistic_regression
random_forest
svm
57500
55000
52500
50000
47500
45000
42500
2016-12
2017-01
2017-02
2017-03
2017-04
2017-05

머신 러닝 모델의 결과

- 대상: Kospi200
- input: **['Close', 'Volume']**
- 기간:
  - 8: 2 (비율)
  - Training (2016-11-30 ~ 2017-04-27)
  - Test (2017-04-28 ~ 2017-05-25)

적중률: Logistic Regreesion=0.44, RandomForest=0.43, SVM=0.45

```
…
199/200 code:024110 ==================================
024110 2016-11-30 09:56:46.022576 ~ 2017-05-30 09:56:46.022576
hit_count=9, total=22, hit_ratio = 0.4090909090909091 [2017-04-18 ~ 2017-05-23]
hit_count=6, total=22, hit_ratio = 0.2727272727272727 [2017-04-18 ~ 2017-05-23]
hit_count=6, total=22, hit_ratio = 0.2727272727272727 [2017-04-18 ~ 2017-05-23]
024110 : Hit Ratio - Logistic Regreesion=0.41, RandomForest=0.27, SVM=0.27, [2016-11-30 ~ 2017-05-30]
Total Hit Ratio - Logistic Regreesion=0.44, RandomForest=0.43, SVM=0.45, [2016-11-30 ~ 2017-05-30]
```

머신 러닝 모델의 결과

- 대상: Kospi200 (평균회귀 점수가 6점 이상(max: 9))
- input: **['Close', 'Volume']**
- 기간:
  - 8: 2 (비율)
  - Training (2016-11-30 ~ 2017-04-27)
  - Test (2017-04-28 ~ 2017-05-25)

적중률: Logistic Regreesion=0.46(0.02상승), RandomForest=0.45(0.02상승), SVM=0.46(0.02상승)

```
...
27/28 code:000210 ==================================
000210 2016-11-30 10:49:26.480220 ~ 2017-05-30 10:49:26.480220
hit_count=10, total=22, hit_ratio = 0.45454545454545453 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
hit_count=13, total=22, hit_ratio = 0.5909090909090909 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
hit_count=9, total=22, hit_ratio = 0.4090909090909091 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
000210 : Hit Ratio - Logistic Regreesion=0.45, RandomForest=0.59, SVM=0.41, [2016-11-30 ~ 2017-05-30]
Total Hit Ratio - Logistic Regreesion=0.46, RandomForest=0.45, SVM=0.46, [2016-11-30 ~ 2017-05-30]
```

머신 러닝 모델의 결과

- 대상: Kospi200
- input: **['Open', 'High', 'Low', 'Close', 'Volume', 'kospi', 'kospi_volume', 'SMA', 'BBANDS_upper', 'BBANDS_middle', 'BBANDS_lower', "MOM", "STOCH_slowk", "STOCH_slowd" , "MACD_macd", "MACD_signal", "MACD_hist"]**
- 기간:
  - 8: 2 (비율)
  - Training (2016-11-30 ~ 2017-04-27)
  - Test (2017-04-28 ~ 2017-05-25)

적중률: Logistic Regreesion=0.48(0.02상승), RandomForest=0.60(0.15상승), SVM=0.62(0.16상승)

```
…
198/200 code:064960 ==================================
064960 2016-11-30 10:53:41.990620 ~ 2017-05-30 10:53:41.990620
hit_count=12, total=22, hit_ratio = 0.5454545454545454 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
hit_count=15, total=22, hit_ratio = 0.6818181818181818 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
hit_count=15, total=22, hit_ratio = 0.6818181818181818 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
064960 : Hit Ratio - Logistic Regreesion=0.55, RandomForest=0.68, SVM=0.68, [2016-11-30 ~ 2017-05-30]
Total Hit Ratio - Logistic Regreesion=0.48, RandomForest=0.60, SVM=0.62, [2016-11-30 ~ 2017-05-30]
```

머신 러닝 모델의 결과

- 대상: Kospi200 (평균회귀 점수가 6점 이상(max: 9))
- input: **['Open', 'High', 'Low', 'Close', 'Volume', 'kospi', 'kospi_volume', 'SMA', 'BBANDS_upper', 'BBANDS_middle', 'BBANDS_lower', "MOM", "STOCH_slowk", "STOCH_slowd" , "MACD_macd", "MACD_signal", "MACD_hist"]**
- 기간:
  - 8: 2 (비율)
  - Training (2016-11-30 ~ 2017-04-27)
  - Test (2017-04-28 ~ 2017-05-25)

적중률: Logistic Regreesion=0.51(0.03상승), RandomForest=0.63(0.03상승), SVM=0.62(-)

```
...
27/28 code:034220 ==================================
034220 2016-11-30 10:56:15.084086 ~ 2017-05-30 10:56:15.084086
hit_count=13, total=22, hit_ratio = 0.5909090909090909 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
hit_count=17, total=22, hit_ratio = 0.7727272727272727 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
hit_count=12, total=22, hit_ratio = 0.5454545454545454 [2017-04-18 00:00:00 ~ 2017-05-23 00:00:00]
034220 : Hit Ratio - Logistic Regreesion=0.59, RandomForest=0.77, SVM=0.55, [2016-11-30 ~ 2017-05-30]
Total Hit Ratio - Logistic Regreesion=0.51, RandomForest=0.63, SVM=0.62, [2016-11-30 ~ 2017-05-30]
```

머신 러닝 모델의 결과

- 파라 미터 최적화
  - Input data, 평균회귀

	Input: 2, 200개 종목	Input: 17개, 28개 종목
평균회귀 미 적용	Regreesion=0.44, RandomForest=0.43, SVM=0.45	Regreesion=0.48, RandomForest=0.60, SVM=0.62
평균회귀 적용	Regreesion=0.46, RandomForest=0.45, SVM=0.46	Regreesion=0.51 (0.7 상승), RandomForest=0.63 (1.9 상승), SVM=0.62 (1.7 상승)

머신 러닝 모델의 결과

- **종목: Kospi 200**
- **매수 조건**
  - AND, 평균회귀 모델에서 매수하라고 추천한 종목
  - AND, 머신 러닝 모델이 다음 날 상승한다고 추천한 종목 (내일 뭘 살지?)
  - AND, 하락했다가 상승하는 시점에 시장가로 매수 (언제 살지?)
- **매도 조건**
  - 매수 가격에서 0.8% 상승 (1% 하락 시점에 추가 매수)

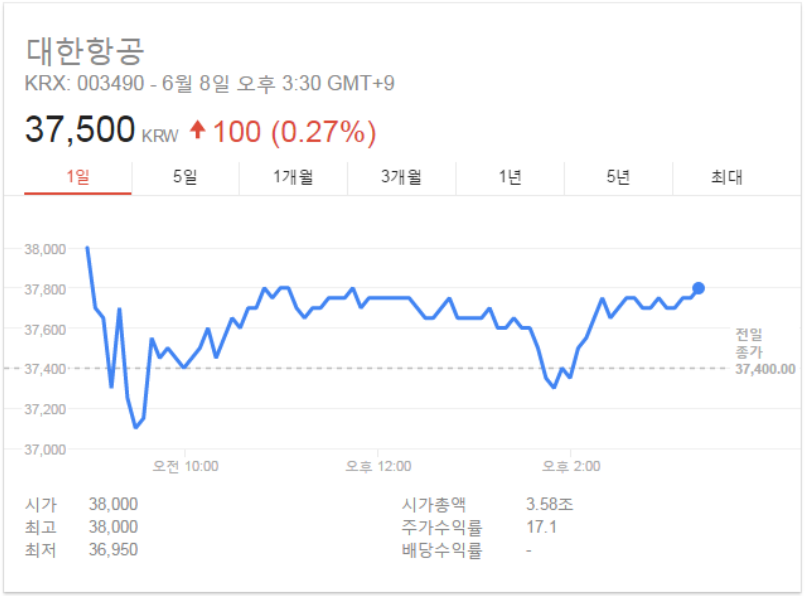
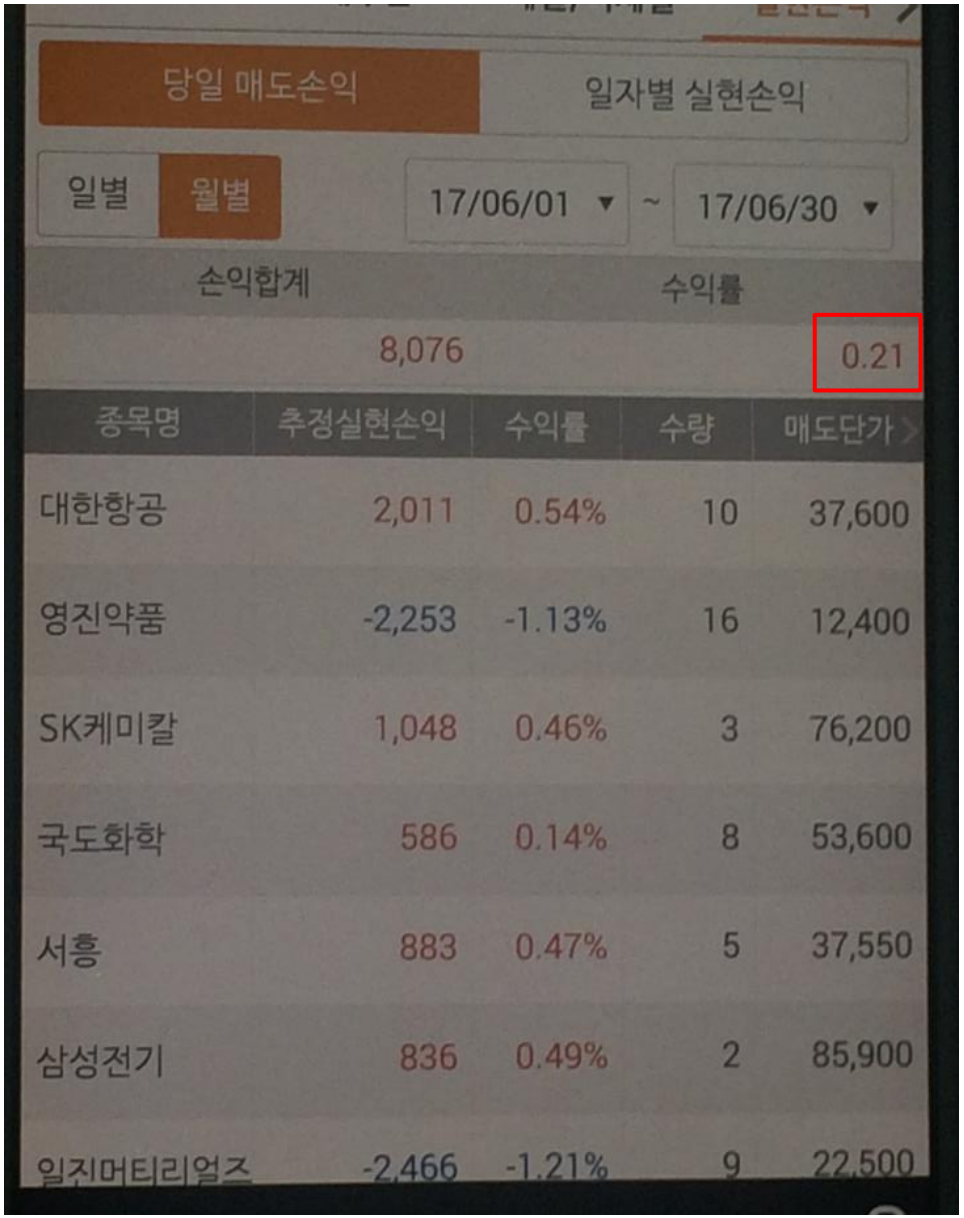


사진 출처: google

머신 러닝 모델의 결과

- 대상 Kospi200
- 기간 2017-06-07~
- 수익률: 0.21



1. 금융 데이터 소개

2. 금융 데이터 분석을 위한 파이썬 라이브러리들

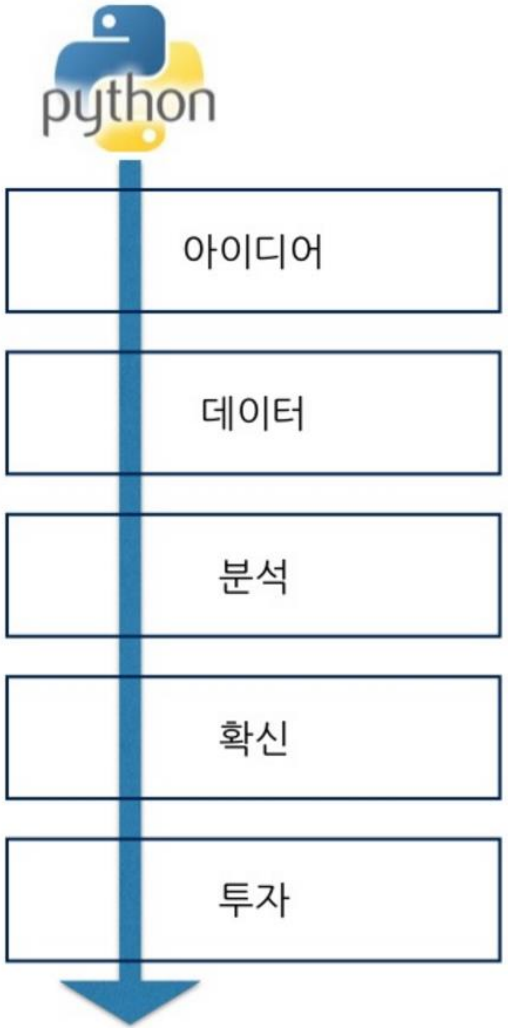
3. 금융 데이터 수집

4. 예측 모델1 (평균회귀모델)

5. 예측 모델2 (머신러닝)

6. 정리

정리



- 1인 트레이딩 시스템 개발은 **파이썬**으로…
  - 훌륭한 라이브러리들
  - 압도적 생산성
- **과적합 (Overfitting) 문제**
  - 과거의 성과가 미래의 수익률을 절대로 보장하지 않습니다.
  - 자신의 아이디어를 증명하는 수준으로 사용하세요
- **노이즈 + 노이즈 = 노이즈**

참조: https://www.slideshare.net/ssuser130257/pycon2016-64946828

“수치로 확인하지 않고서는 한 발짝도 움직이지 마라”

서울대 문병로 교수