돈 좀 벌자

기계학습 7조

202001197 김희은 201701472 박승리
201901803 변정민 201602362 이선우

Contents

1 주제 선정

2 데이터 및 모델 소개

3 분석 및 결론 도출

4 시사점 및 한계, 개선 방안

5 QnA

1%의 수익률로
999일 동안 투자 시 수익률
"원금의 약 20750배"

→ " 100만원 투자 " 〉〉〉 " 207억 5천만원 수익 "

→ 머신러닝을 활용하여 주가를 예측하고 부자가 되자!

주가 데이터의 특징

- ## 시계열 데이터

  과거 데이터가 현재 데이터에 영향을 줄 수 있음
  (Autoregressive 한 선형관계)
  >> 예측시 기존 데이터간 **선형관계 추정이 어려움**

- ## 외부 충격

  데이터 추세와 독립된 외부요인의 영향이 존재한다.
  (White Noise, Economic Shock)
  >> 다양한 외부 충격에 대한 **주가의 반응 추정이 어려움**

- ## 경제 이론들간의 연계성

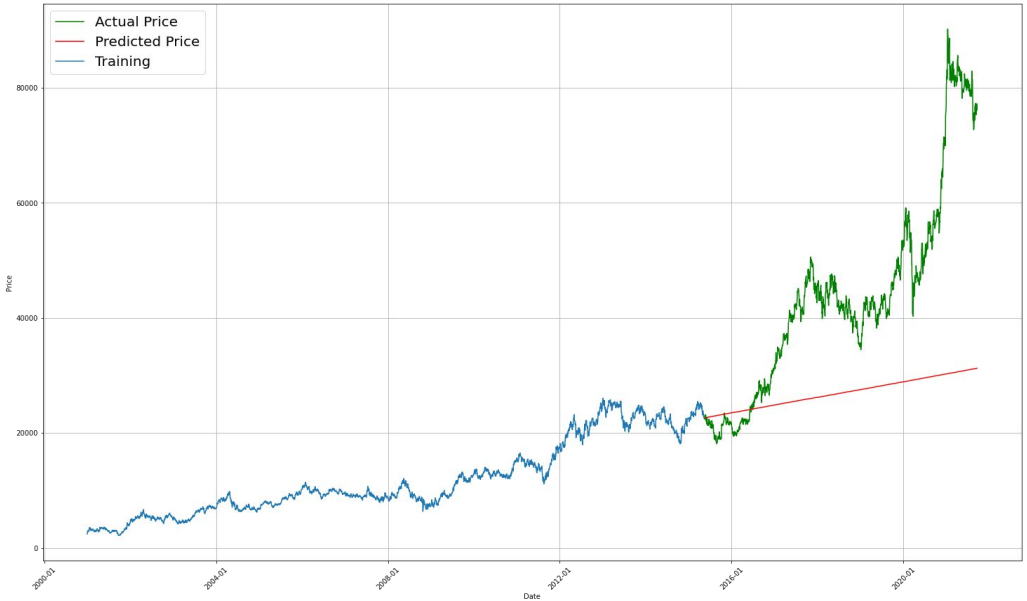
  APT, CAPM의 재무이론, 이자율과 관련된 거시경제이론 등
  여러 경제이론과 연계 됨
  >> 주가는 **여러 경제주체간 상호작용의 결과물**

ex) 로지스틱 모델

모델 후보

KNN, Logistic, ARIMA, XGBoost 모델

ex) ARIMA 모델



회귀 모델

>> **추세**를 예측

>> White Noise 영향 감소

>> 외부요인이 많은 주식에 부적합 할 수 있다

>> Autoregressive 효과로 정확한 예측이 어려움

회귀 모델 배제

모델 선정

주가 데이터의 특징을 고려

주요 특징

선형관계 추정의 한계 / 외부 요인에 대한 반응 추정의 한계 / 가격이 지니는 경제적 상호작용 분석의 한계

모델 선정 기준

1. 회귀 모형이 아닐 것 (∵ 다중 공선성 문제, 오차항간 자기상관성 존재)
2. 외부 요인에 대한 불확실한 반응 추정 가능 (∴ 비효율적 시장가설을 통한 추가수익 가능성)
3. 금일 데이터를 활용해, 다음날 예측이 용이한 모델일 것
4. 기술적 투자 전략을 반영할 수 있을 것

최종모델 선정

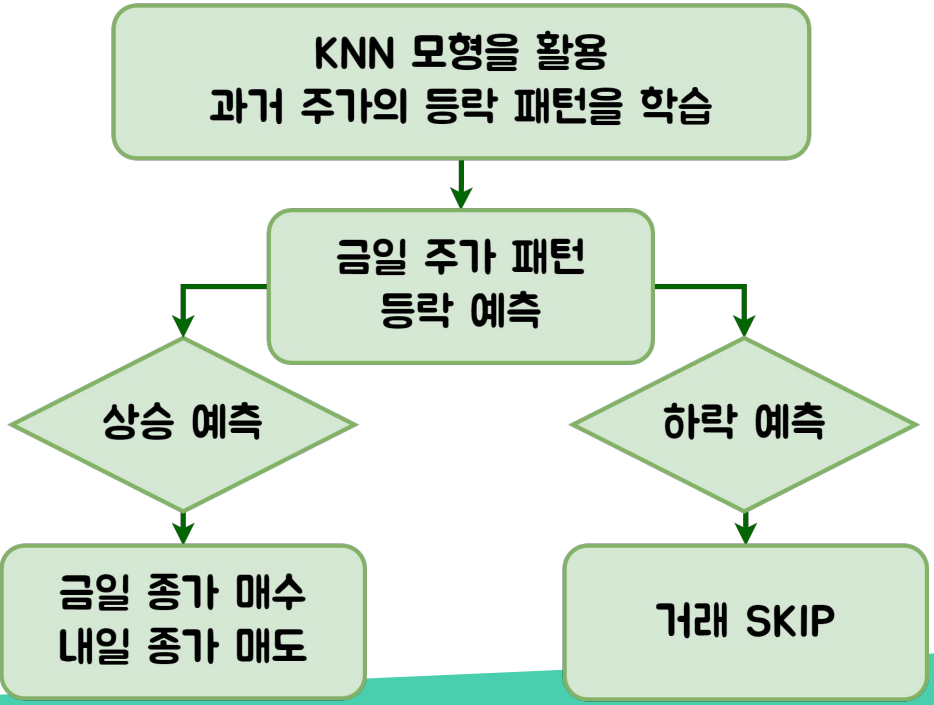
>> KNN 모형

모델 선정

KNN 모델을 활용한 투자전략 알고리즘

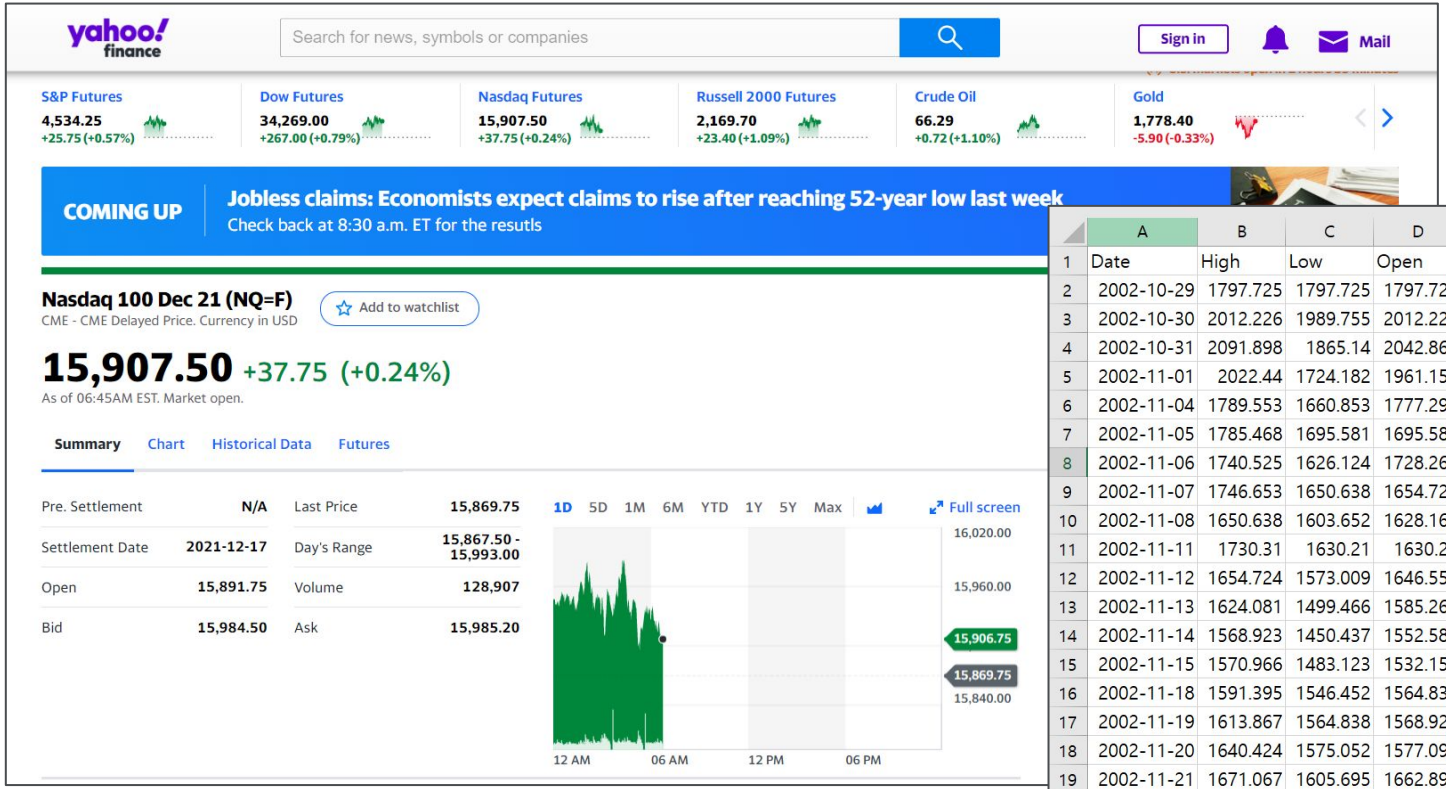
KNN 모델의 장점

1. 모델을 가정하지 않음
   >> 주가 가격의 선형적 가설 불필요
2. 다양한 패턴들을 학습시킬 수 있음
   >> 주가의 과거 외부요인 반응들 반영 가능
3. 정규분포를 가정하지 않음
   >> 주가의 움직임을 보다 현실적으로 반영



데이터 출처

Yahoo Finance API



	A	B	C	D	E	F	G
1	Date	High	Low	Open	Close	Volume	Adj Close
2	2002-10-29	1797.725	1797.725	1797.725	1797.725	501745	1695.226
3	2002-10-30	2012.226	1989.755	2012.226	2012.226	20892185	1897.497
4	2002-10-31	2091.898	1865.14	2042.869	1912.125	32327083	1803.104
5	2002-11-01	2022.44	1724.182	1961.154	1793.639	18373668	1691.373
6	2002-11-04	1789.553	1660.853	1777.296	1703.753	16939411	1606.611
7	2002-11-05	1785.468	1695.581	1695.581	1701.71	15353406	1604.685
8	2002-11-06	1740.525	1626.124	1728.267	1664.938	12487339	1570.01
9	2002-11-07	1746.653	1650.638	1654.724	1666.981	11682099	1571.936
10	2002-11-08	1650.638	1603.652	1628.167	1634.295	6740519	1541.114
11	2002-11-11	1730.31	1630.21	1630.21	1675.153	10783852	1579.642
12	2002-11-12	1654.724	1573.009	1646.552	1585.266	7176181	1494.881
13	2002-11-13	1624.081	1499.466	1585.266	1503.552	10570917	1417.825
14	2002-11-14	1568.923	1450.437	1552.581	1495.38	8566383	1410.12
15	2002-11-15	1570.966	1483.123	1532.152	1570.966	10548889	1481.396
16	2002-11-18	1591.395	1546.452	1564.838	1548.495	6510451	1460.206
17	2002-11-19	1613.867	1564.838	1568.923	1577.095	9129317	1487.175
18	2002-11-20	1640.424	1575.052	1577.095	1640.424	10766720	1546.893
19	2002-11-21	1671.067	1605.695	1662.896	1613.867	6620590	1521.85
20	2002-11-22	1728.267	1626.124	1650.638	1648.595	17414233	1554.599

데이터 종목

NAVER

네이버

CELLTRION

셀트리온

kakao

카카오

HYUNDAI

현대자동차

엔씨소프트

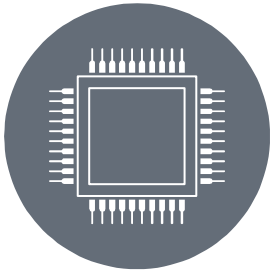
SAMSUNG

삼성전자

독립변수 소개

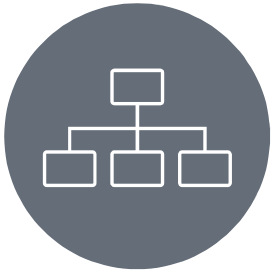
주가의 내재적 데이터

하루치 가격 등락 변화율
시작가와 종가의 차액 변화율
거래량 변화율



주가의 기술적 데이터

Moving Average 비율
RSI 지표
(골든 크로스, 과매수/과매도)



주가의 외부적 데이터

코스피, 나스닥, S&P500,
다우존스, 환율

데이터 불러오기

```
'''데이터 불러오기'''   >>> "Naver 종목 분석"

asset_name = 'naver' #종목 이름

tmp1 = pd.read_csv(f"{path_dir}/Data/{asset_name}.csv", index_col='Date', parse_dates=['Date']) #자산지수

tmp2 = pd.read_csv(f'{path_dir}/Data/kospi.csv", index_col='Date', parse_dates=['Date']) #코스피지수

tmp3 = pd.read_csv(f'{path_dir}/Data/nasdaq.csv', index_col='Date', parse_dates=['Date']) #나스닥 지수

tmp4 = pd.read_csv(f'{path_dir}/Data/s&p500.csv', index_col='Date', parse_dates=['Date']) #S&P500 지수

tmp5 = pd.read_csv(f'{path_dir}/Data/dowjones.csv', index_col='Date', parse_dates=['Date']) #DowJones 지수

tmp6 = pd.read_csv(f'{path_dir}/Data/exch.csv', index_col='Date', parse_dates=['Date']) #원달러 환율 지수

data = [tmp1, tmp2, tmp3, tmp4, tmp5, tmp6]
```

시간 데이터 전처리

```
'''시간 데이터 전처리'''

def process_time (df):

    '''시작일'''
    start_date_list = []
    end_date_list = []
    for i in df:
        start_date_list.append(i.index[0])
        end_date_list.append(i.index[len(i)-1])
    start_date = max(start_date_list)
    end_date = min(end_date_list)
    date_index = pd.date_range(start_date, end_date)
```

시간 데이터 전처리

```
'''시간대 맞춰서 데이터 가져오기'''
tmp = pd.DataFrame(index=date_index)
for i in df:
    if i is tmp1:
        name = "asset"
    elif i is tmp2:
        name = 'kospi'
    elif i is tmp3:
        name = 'nasdaq'
    elif i is tmp4:
        name = 'snp500'
    elif i is tmp5:
        name = 'dowjones'
    elif i is tmp6:
        name = 'ex_rate'
    for j in i.columns:
        tmp[f'{name}_{j}'] = i[j]
#tmp 저장 완료
```

시간 데이터 전처리

```
'''외부데이터 결측치 처리'''
    for row_index, value in tmp.iterrows():
        day = datetime.weekday(row_index)
        if day == (5 or 6):
            tmp.drop(row_index, axis=0, inplace=True)

    '''자산데이터 결측치 처리'''
    tmp_col = tmp.columns
    col_list = []
    for i in range(len(tmp.columns)-1):
        if tmp_col[i][0:5] == "asset":
            col_list.append(tmp_col[i])
```

```
    for index, row in tmp[col_list].iterrows():
        if row.isnull().sum() != 0:
            tmp = tmp.drop(index = index)

'''외부데이터 빈 값, 앞방향 채우기
        (휴장으로 인한 데이터 불변상황 가정)'''

    tmp = tmp.fillna(method='ffill')

    return tmp
```

변수 데이터 전처리

```
'''다음날 등락여부''
  for i in df.index:
    if df.loc[i, 'asset_Close_pct' ] > 0:
      df.loc[i,'signal'] = 1
    else:
      df.loc[i,'signal'] = 0
  df['signal'] = df['signal'].shift(-1)  #다음날 승률을 예측 하려는 것이니 한칸 당겨주기
```

전일 종가 < 금일 종가 == 1
전일 종가 > 금일 종가 == 0

>> 금일 데이터 패턴들로 다음날 등락 여부 학습

변수 데이터 전처리

```
'''내재 데이터(가격)'''
df['asset_high-low_pct'] = df['asset_High'] - df['asset_Low'].pct_change()
df['asset_close-open_pct'] = df['asset_Close'] - df['asset_Open'].pct_change()
df['asset_delta_vol_pct'] = df['asset_Volume'].pct_change()

'''기술 데이터(기술적 지표)'''
df['MA5_20_ratio'] = talib.SMA(np.asarray(df['asset_Close']), timeperiod=5)/talib.SMA(np.asarray(df['asset_Close']), timeperiod=20)
df['MA5_20_ratio_pct'] = df['MA5_20_ratio'].pct_change()
df['MA20_60_ratio'] = talib.SMA(np.asarray(df['asset_Close']), timeperiod=20)/talib.SMA(np.asarray(df['asset_Close']), timeperiod=60)
df['MA20_60_ratio_pct'] = df['MA5_20_ratio'].pct_change()
df['RSI14'] = talib.RSI(np.asarray(df['asset_Close']), 14) / 100
```

내재적 데이터

- 금일 주가 최고점 최저점 변화율
- 금일 주가 종가 시초가 변화율
- 금일 주가 거래량 변화율

기술적 지표

- 장단기 골든, 데드 크로스: 수렴발산비율 변화율
- 상대강도지표(RSI): 과매수 과매도 (%)

변수 데이터 전처리

```
'''외부 데이터(시장 지표)'''
df['kospi_rtn'] = df['kospi_Close'].pct_change()
df['nasdaq_rtn'] = df['nasdaq_Close'].pct_change(-1)  #미국과 한국의 시차가 있으니 하루 당겨주기 (한국시장이 미국시장보다 먼저열림)
df['snp500_rtn'] = df['snp500_Close'].pct_change(-1)  #미국과 한국의 시차가 있으니 하루 당겨주기 (한국시장이 미국시장보다 먼저열림)
df['dowjones_rtn'] = df['dowjones_Close'].pct_change(-1)  #미국과 한국의 시차가 있으니 하루 당겨주기 (한국시장이 미국시장보다 먼저열림)
df['ex_rate_ch'] = df['ex_rate_Close'].pct_change()
```

시장 외부 데이터

- **국내 시장: 코스피지수 변화율**
- **국외 시장: 나스닥, S&P500, 다우존스 변화율 (금융동조화 현상 가정)**
- **외환 시장: 원달러 환율 변화율**

데이터 전처리 결과

```
'''불러온 Raw data 처리 (1)'''
df = process_timedata(data)
df
```

	asset_High	asset_Low	asset_ Open	asset_ Close	asset_ Volume	asset_ Adj Close	kospi_High	kospi_Low	kospi_ Open	kospi_ Close	kospi_ Volume	kospi_ Adj Close
2003-12-01	6247.094	5707.776	5707.776	6189.894	8806242	5836.971	811.49	790.8	796.36	807.39	585000	807.39
2003-12-02	6336.98	6071.407	6275.694	6157.208	7112545	5806.148	813.95	806.3	813.49	807.78	532000	807.78
2003-12-03	6169.465	5965.178	6128.607	6169.465	4547525	5817.706	816.02	801.08	807.17	808.34	440600	808.34
2003-12-04	6308.38	6128.607	6210.322	6189.894	4459414	5836.971	809.68	800.31	805.1	805.13	493900	805.13
2003-12-05	6328.809	6108.179	6128.607	6153.122	4439834	5802.294	802.86	789.41	802.71	789.41	426500	789.41
...	...	...	...	...	...	...	...	...	...	...	...	...
2021-11-30	392500	381000	391000	381000	1103481	381000	2942.93	2822.73	2932.71	2839.01	982400	2839.01
2021-12-01	392500	382000	385000	390000	548840	390000	2905.74	2837.03	2860.12	2899.72	563100	2899.72
2021-12-02	399000	382000	382500	398500	613006	398500	2945.27	2874.64	2874.64	2945.27	533700	2945.27
2021-12-03	402000	394500	397500	402000	501099	402000	2975.44	2927.55	2935.93	2968.33	486100	2968.33
2021-12-06	400500	389500	400500	392000	456773	392000	2983.5	2932.49	2954.82	2973.25	479400	2973.25

4485 rows × 36 columns

데이터 전처리 결과

```
'''불러온 Raw data 처리 (1)'''
df = process_timedata(data)
df
```

	nasdaq_ High	nasdaq_ Low	nasdaq_ Open	nasdaq_ Close	nasdaq_ Volume	nasdaq_ Adj Close	snp500_ High	snp500_ Low	snp500_ Open	snp500_ Close	snp500_ Volume	snp500_ Adj Close
2003-12-01	1989.82	1968.54	1972.97	1989.82	1.84E+09	1989.82	1070.47	1058.2	1058.2	1070.12	1.38E+09	1070.12
2003-12-02	1996.08	1978.23	1986.8	1980.07	1.80E+09	1980.07	1071.22	1065.22	1070.12	1066.62	1.38E+09	1066.62
2003-12-03	2000.92	1960.13	1989.14	1960.25	2.24E+09	1960.25	1074.3	1064.63	1066.62	1064.73	1.44E+09	1064.73
2003-12-04	1971.25	1942.67	1966.92	1968.8	2.11E+09	1968.8	1070.37	1063.15	1064.73	1069.72	1.46E+09	1069.72
2003-12-05	1960.39	1935.58	1949.26	1937.82	1.67E+09	1937.82	1069.72	1060.09	1069.72	1061.5	1.27E+09	1061.5
...	...	...	...	...	...	...	...	...	...	...	...	...
2021-11-30	15828.2	15451.39	15716.5	15537.69	6.61E+09	15537.69	4646.02	4560	4640.25	4567	4.95E+09	4567
2021-12-01	15816.82	15243.93	15752.27	15254.05	6.27E+09	15254.05	4652.94	4510.27	4602.82	4513.04	4.08E+09	4513.04
2021-12-02	15444.54	15150.12	15181.82	15381.32	5.39E+09	15381.32	4595.46	4504.73	4504.73	4577.1	3.77E+09	4577.1
2021-12-03	15470.36	14931.06	15428.71	15085.47	5.86E+09	15085.47	4608.03	4495.12	4589.49	4538.43	3.97E+09	4538.43
2021-12-06	15281.99	14931.61	15117.63	15225.15	5.10E+09	15225.15	4612.6	4540.51	4548.37	4591.67	3.31E+09	4591.67

4485 rows × 36 columns

데이터 전처리 결과

```
'''불러온 Raw data 처리 (1)'''
df = process_timedata(data)
df
```

	dowjones_ High	dowjones_ Low	dowjones_ Open	dowjones_ Close	dowjones_ Volume	dowjones_ Adj Close	ex_rate_ High	ex_rate_ Low	ex_rate_ Open	ex_rate_ Close	ex_rate_ Volume	ex_rate_ Adj Close
2003-12-01	9902.23	9785.35	9785.35	9899.05	2.28E+08	9899.05	1203.5	1195	1197.3	1198.4	0	1198.4
2003-12-02	9900.45	9837.27	9899.64	9853.64	2.58E+08	9853.64	1198.5	1186.2	1198.1	1195	0	1195
2003-12-03	9942.01	9851.42	9851.94	9873.42	2.23E+08	9873.42	1198.4	1192.5	1195	1192.5	0	1192.5
2003-12-04	9933.86	9865.78	9874.83	9930.82	2.68E+08	9930.82	1197.9	1174.8	1192.6	1190.1	0	1190.1
2003-12-05	9923.42	9846.31	9923.27	9862.68	2.02E+08	9862.68	1193.8	1178.6	1189.4	1184.7	0	1184.7
...	...	...	...	...	...	...	...	...	...	...	...	...
2021-11-30	35056.99	34424.44	35056.99	34483.72	6.79E+08	34483.72	1190.96	1181.71	1189.96	1190.74	0	1190.74
2021-12-01	35004.64	34006.98	34678.94	34022.04	4.96E+08	34022.04	1184.45	1159.1	1182.86	1182.86	0	1182.86
2021-12-02	34759.65	34076.25	34076.25	34639.79	4.67E+08	34639.79	1178.5	1172.36	1176.2	1177.11	0	1177.11
2021-12-03	34801.31	34264.57	34692.78	34580.08	4.4E+08	34580.08	1185.1	1175.44	1175.88	1176.77	0	1176.77
2021-12-06	35356.75	34633.43	34633.43	35227.03	4.17E+08	35227.03	1185.14	1178.74	1184.56	1183.14	0	1183.14

4485 rows × 36 columns

데이터 전처리 결과

```
'''불러온 Raw data 처리 (2)'''
df, X_factors, Y_factors, XY_factors = process_factors(df)
df[XY_factors]
```

	Signal	asset_high_low_pct	asset_close_open_pct	asset_delta_vol_pct	MA5_20_ratio_pct	MA20_60_ratio_pct	RSI14	kospi_rtn	nasdaq_rtn	snp500_rtn	dowjones_rtn	ex_rate_pct
2004-02-24	0.0	-0.09836	-0.13953	0.726826	-0.00681	0.006673	0.544212	-0.01474	-0.00867	-0.00401	-0.00333	0.000593
2004-02-25	0.0	0.090911	-0.32432	-0.06995	-0.01403	0.005414	0.494922	0.002637	-0.00472	-0.00108	0.00203	-0.01805
2004-02-26	1.0	0.866666	2.2	-0.4941	-0.01581	0.004642	0.470226	-0.00232	0.001355	-2.6E-05	-0.00036	0.006041
2004-02-27	0.0	-0.64286	-1.3	-0.46456	-0.01558	0.003567	0.503274	0.02146	-0.0136	-0.00954	-0.00882	0.006004
2004-03-01	0.0	-1	-1	-1	-0.01243	0.00301	0.503274	0	0.008899	0.005979	0.008182	0.003496
...		...	...	...	...	...	...	...	...	...	...	...
2021-11-26	0.0	0.363636	0.5	0.465805	-0.00509	-0.0009	0.395216	-0.01471	-0.01845	-0.01303	-0.00673	0.00197
2021-11-29	0.0	0.2	-1	-0.03437	-0.00912	-0.00101	0.378616	-0.00924	0.015777	0.019328	0.018914	0.003688
2021-12-01	1.0	-0.08696	-1.5	-0.50263	-0.00102	0.000719	0.426955	0.021384	-0.00827	-0.014	-0.01783	-0.00662
2021-12-02	1.0	0.619048	2.2	0.116912	0.002389	0.000204	0.488781	0.015708	0.019612	0.008521	0.001727	-0.00486
2021-12-03	0.0	-0.55882	-0.71875	-0.18255	0.00749	0.001035	0.512122	0.00783	-0.00917	-0.0116	-0.01837	-0.00029

4166 rows × 12 columns

KNN 모델링

```
'''데이터 불러오기'''
X = df[X_factors]
Y = df[Y_factors[0]] # 등락 Signal

'''테스트 데이터 나누기 (70%)'''
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, shuffle=False) #섞지않고

'''변수 정규화'''
# 데이터 전처리 과정을 통해, % 스케일로 바꾸어, 변수 정규화를 따로 진행하지 않음.
```

학습용 데이터, 테스트 및 검증 데이터 비중 70% : 30%

독립변수 백분위 %로 정규화 진행 X

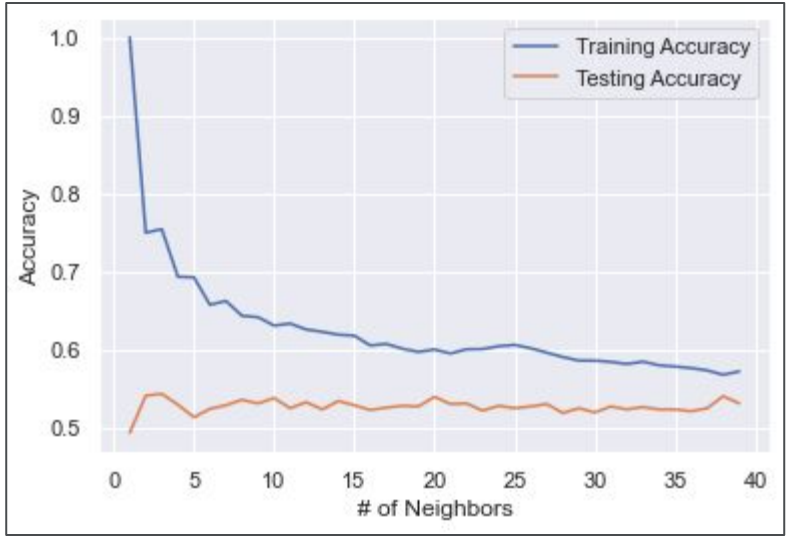
KNN 모델링

```
'''Best K 찾기'''

training_accuracy = []
test_accuracy = []
k_settings = range(1, 40)

for k in k_settings:
    ploan_knn = neighbors.KNeighborsClassifier(n_neighbors=k)
    ploan_knn.fit(X_train, Y_train)
    training_accuracy.append(ploan_knn.score(X_train, Y_train))
    test_accuracy.append(ploan_knn.score(X_test, Y_test))

'''K에 따른 훈련, 예측 정확도 그리기'''
plt.plot(k_settings, training_accuracy, label="Training Accuracy")
plt.plot(k_settings, test_accuracy, label="Testing Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("# of Neighbors")
plt.legend()
plt.show()
```



KNN 모델링

```
'''K에 따른 일치비율'''

dict = {}
for i in range(1,40):
    best_ploan_knn = neighbors.KNeighborsClassifier(n_neighbors=i)
    best_ploan_knn.fit(X_train, Y_train)
    best_ploan_knn.predict(X_test)

    n_test = len(Y_test)
    Y_predict = best_ploan_knn.predict(X_test)
    dict[f'k = {i}'] = round(sum(Y_test == Y_predict)/n_test*100,2)

print(dict)
max_dict = max(dict, key= lambda x: dict[x])
min_dict = min(dict, key= lambda x: dict[x])
print(f'최대 예측 일치 비율: {max_dict}: {dict.get(max_dict)}')
print(f'최소 예측 일치 비율: {min_dict}: {dict.get(min_dict)}')
```

'k = 1': 50.48, 'k = 2': 49.2, 'k = 3': 52.72, 'k = 4': 53.36, 'k = 5': 52.56, 'k = 6': 52.32, 'k = 7': 53.44, 'k = 8': 53.36, 'k = 9': 52.56, 'k = 10': 52.64, 'k = 11': 52.72, 'k = 12': 52.4, 'k = 13': 52.88, 'k = 14': 51.76, 'k = 15': 51.12, 'k = 16': 51.52, 'k = 17': 50.96, 'k = 18': 51.36, 'k = 19': 50.4, 'k = 20': 51.76, 'k = 21': 50.56, 'k = 22': 51.76, 'k = 23': 50.24, 'k = 24': 49.84, 'k = 25': 50.32, 'k = 26': 50.56, 'k = 27': 51.44, 'k = 28': 50.4, 'k = 29': 50.24, 'k = 30': 50.56, 'k = 31': 51.2, 'k = 32': 50.56, 'k = 33': 51.12, 'k = 34': 52.0, 'k = 35': 50.4, 'k = 36': 51.44, 'k = 37': 50.16, 'k = 38': 50.64, 'k = 39': 50.08

최대 예측 일치 비율: k = 7: 53.44
최소 예측 일치 비율: k = 2: 49.2

KNN 모델링

```
'''최적의 K값'''
best_k_temp = test_accuracy==max(test_accuracy)
best_k = list(compress(k_settings, best_k_temp))[0]
print('Most Accurate k:', best_k)

'''k에서의 일치 비율'''
best_ploan_knn = neighbors.KNeighborsClassifier(n_neighbors=best_k)
best_ploan_knn.fit(X_train, Y_train)
best_ploan_knn.predict(X_test)
```

```
n_test = len(Y_test)
Y_predict = best_ploan_knn.predict(X_test)

print('테스트1 데이터 개수:', n_test)
print('예측과 일치한 데이터 개수:', sum(Y_test == Y_predict))
print('일치 비율:',round(sum(Y_test == Y_predict)/n_test*100,2),'%')
```

Most Accurate k: 7
테스트 데이터 개수: 1250
예측과 일치한 데이터 개수: 668
일치 비율: 53.44 %

KNN 모델링

```
'''모델 성능 지표'''


accuracy = accuracy_score(Y_test, Y_predict)
precision = precision_score(Y_test, Y_predict)
recall = recall_score(Y_test, Y_predict)
f1 = f1_score(Y_test, Y_predict)
print('Accuracy : {0:.3f}'.format(accuracy))
print('Precision : {0:.3f}'.format(precision))
print('Recall : {0:.3f}'.format(recall))
print('F1 : {0:.3f}'.format(f1))
```

Accuracy : 0.534
Precision : 0.506
Recall : 0.482
F1 : 0.494

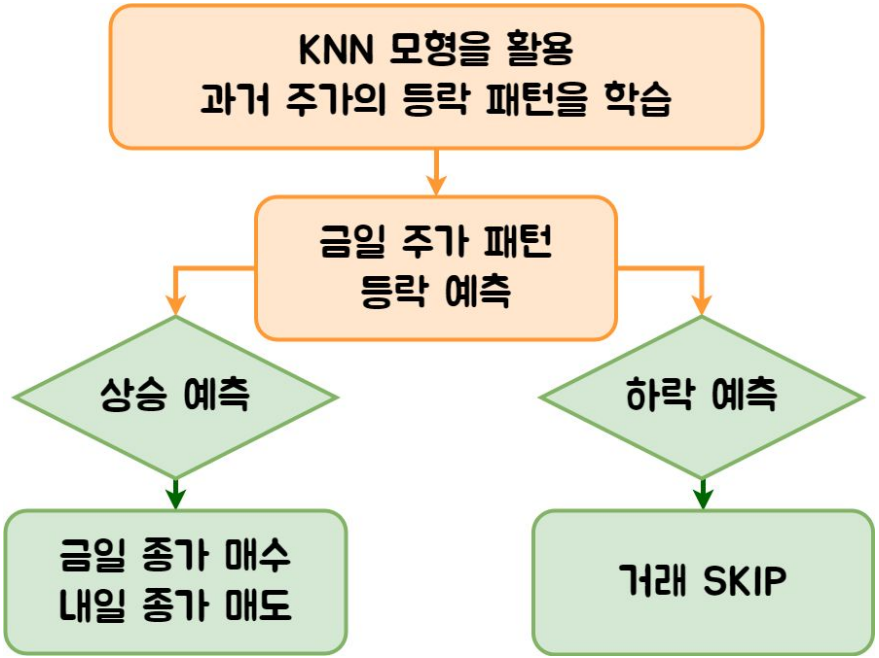
모델을 활용한 투자 전략 (1)

```
best_ploan_knn                                          =
neighbors.KNeighborsClassifier(n_neighbors=best_k)
best_ploan_knn.fit(X_train, Y_train)
train_pct = 0.7
split = int(train_pct*len(df))

'''KNN예측 등락 시그널'''
df['st_signal'] = best_ploan_knn.predict(X)

'''초기 투자금'''
initial_fund = df[split:]['asset_Open'][0]
#초기투자금은 종목 1주 사는 것으로 가정

'''비교전략: Buy and Hold Strategy(매수 후 비교)'''
'''시장 수익률 데이터(시장 종가)'''
df['mkt_rtn'] = df[split:]['asset_Close'] # - df[split:]['asset_Open'][0]
cum_mkt_rtn = df[split:]['mkt_rtn']
```



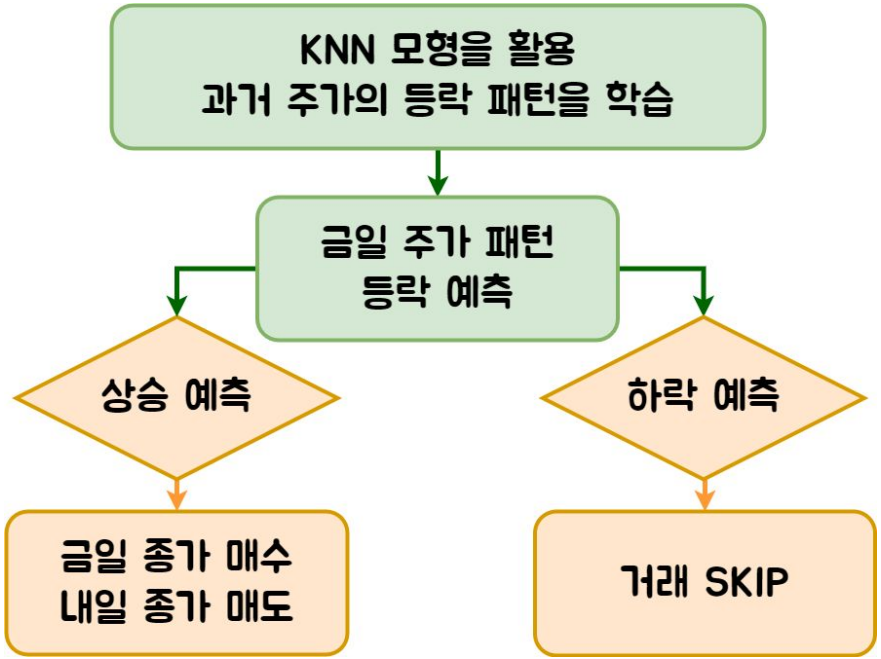
모델을 활용한 투자 전략 (2)

```
'''KNN전략 수익률 데이터(일일 매수매도)'''
df['st_signal'] = df['st_signal'].shift(1) #계산에 활용
df['rtn_pct'] = df['asset_Close'].pct_change()



for i in df[split:].index:
    if df.loc[i, 'st_signal'] == 0:
        df.loc[i, 'st_rtn'] = initial_fund
        #오르지 않음을 예측시 매수매도가 이뤄지지 않으니 초기투자금 그대로
    else:
        df.loc[i, 'st_rtn'] = ( df.loc[i, 'rtn_pct'] + 1 ) * initial_fund
        initial_fund = df.loc[i, 'st_rtn']
        #오른다고 예측시 매수매도 이후, 당일 상승분을 가져감
df['st_real_rtn'] = df[split:]['st_rtn'] # - df[split:]['asset_Open'][0]
cum_st_rtn = df[split:]['st_real_rtn']


print(cum_mkt_rtn)
print(cum_st_rtn)
```



모델을 활용한 투자 전략

```
'''두 전략의 수익률 비교 그래프'''

plt.figure(figsize=(10,5))
plt.gca().yaxis.set_major_formatter(mticker.FormatStrFormatter('%d'))
plt.plot(cum_mkt_rtn, color='r', label='asset_rtn')
plt.plot(cum_st_rtn, color='b', label='st_rtn(Daily_Rtn_pct)')
plt.legend()
plt.savefig(f'{path_dir}/Data_pic/Return Compare on KNN modeling_{asset_name}.png')
plt.show()
print(f"{asset_name}'s Buy and Hold strategy return: {round(net_mkt_rtn,2)}%")
print(f"{asset_name}'s KNN strategy return: {round(net_st_rtn,2)}%")

#샤프비율로 비교
( (자산의 기대수익률 - 기준지표의 기대수익률) / 자산 수익률의 표준편차 : 기준지표 대비 초과수익비율)
std = cum_st_rtn.std()
sharpe = (cum_st_rtn - cum_mkt_rtn)/std
sharpe = sharpe.mean()
print(f"{asset_name}'s Sharpe Ratio: {sharpe}")
```

$$S_a = \frac{E[R_a - R_b]}{\sigma_a}$$

샤프비율을 이용
>> 자산의 위험 대비
KNN전략의 초과수익률 측정

분석결과 (1) - Naver

KNN전략
KOSPI 전체종목 승률: 73.019801%
승리: 590 비김:0 패배:218

Naver's Buy and Hold strategy return: 175.34%
Naver's KNN strategy trading return: 379.43%
Naver's Sharpe Ratio: 0.6821210619625723

분석결과 (2) - 셀트리온

셀트리온's Buy and Hold strategy return: 133.39%
셀트리온's KNN strategy trading return: 579.35%
셀트리온's Sharpe Ratio: 0.5437546217847284

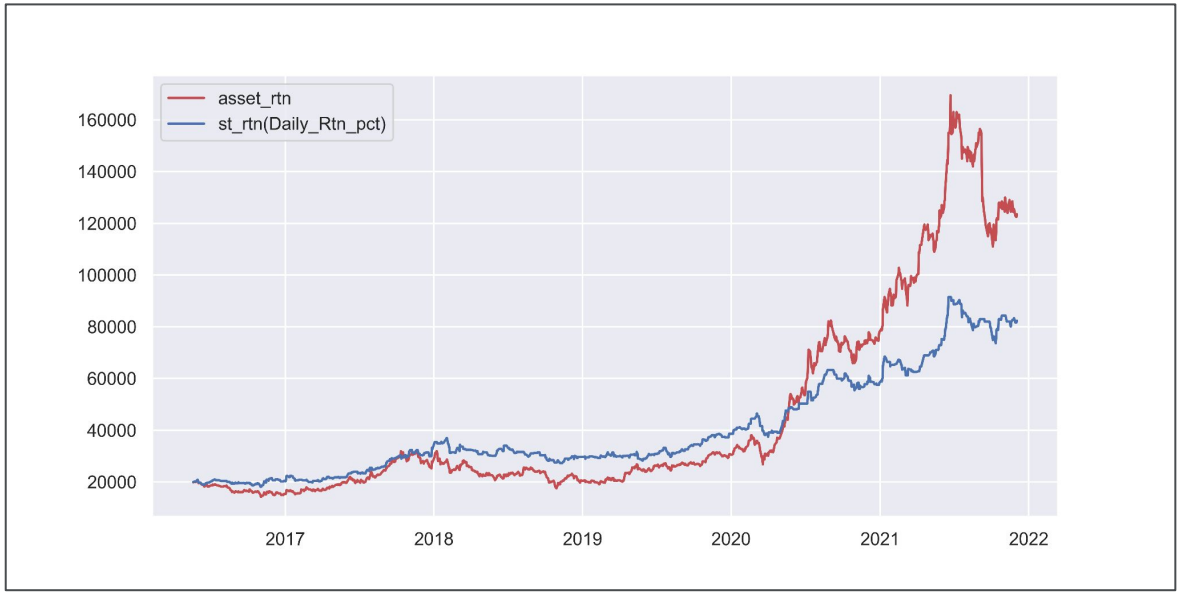
분석결과 (3) - 현대자동차

현대자동차's Buy and Hold strategy return: 53.53%
현대자동차's KNN strategy trading return: 72.45%
현대자동차's Sharpe Ratio: 0.8454370354935857

분석결과 (4) - 엔씨소프트

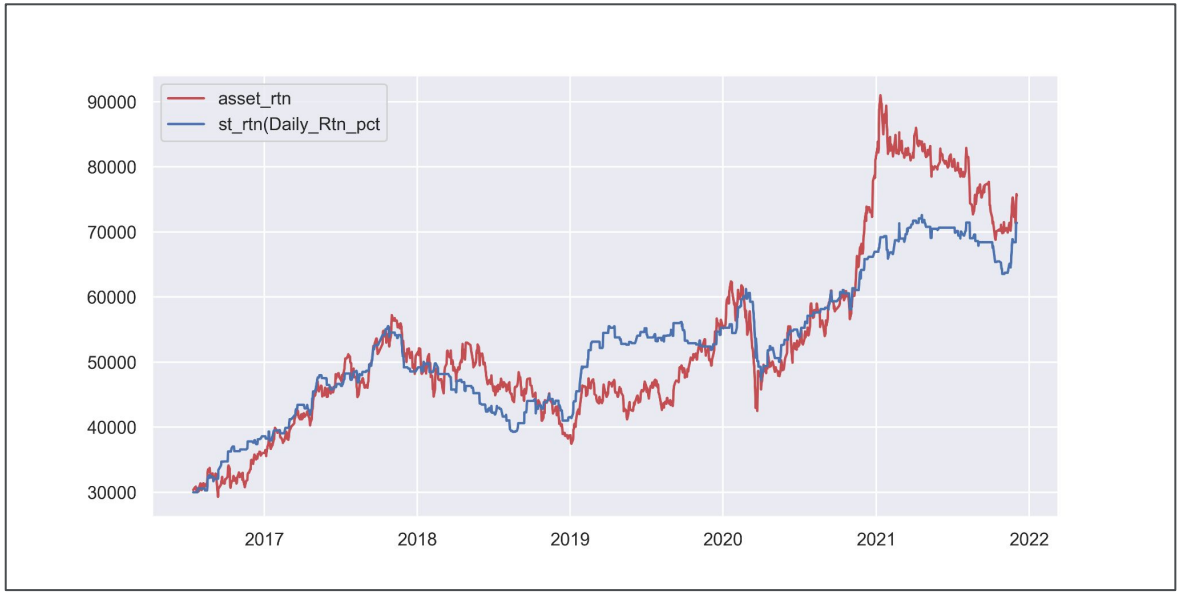
엔씨소프트's Buy and Hold strategy return: 184.23%
엔씨소프트's KNN strategy trading return: 213.64%
엔씨소프트's Sharpe Ratio: 0.07974255317318303

분석결과 (5) - 카카오



카카오's Buy and Hold strategy return: 521.85%
카카오's KNN strategy trading return: 312.14%
카카오's Sharpe Ratio: -0.28231666797858024

분석결과 (6) - 삼성전자



삼성전자's Buy and Hold strategy return: 149.01%
삼성전자's KNN strategy trading return: 137.96%
삼성전자's Sharpe Ratio: -0.07155610613730216

시사점 및 한계 (1)

- KNN 모델의 성능적 한계
  - 실제 주가에 영향을 주는 계절성과 추세를 반영하지 못함.
  - 차원의 저주 (충분한 길이의 학습데이터가 필요함)
  - 학습 시간이 오래 걸림
  - 신규 데이터 추가시, 기존 예측치 변동 가능성 존재
  - 학습된 등락 패턴에 대한 설명력이 낮음
    (∴일치비율(승률)이 낮은 종목에 대한 보완이 어려움)

시사점 및 한계 (2)

- 예측의 과거 데이터 의존성
  - 현실과 미래의 예측 불가능한 상황에 대한 데이터 추정의 한계

- 이론과 실제 트레이딩 사이의 괴리
  - 슬리피지(Slippage) 존재

    (주가 가격 결정 시점, 구매 시점, 판매 시점에 따라 수익변동 가능성 有)
  - 거래비용 존재

    (거래 수수료, 세금 등의 추가 비용으로 인한 수익 변동 가능성 有)

개선 방안 (1)

- KNN의 성능적 한계에 대한 방안
  - 계절성과 추세를 반영할 수 있다고 알려진 ARIMA 모델 사용

⟹ 예측의 과거 데이터 의존성 문제 또한 ARIMA 모델에서 발생
주가의 외부요인 추적 불가능 > 초과수익 달성 어려움

- 예측의 과거 데이터 의존성
  - 현실의 많은 지표를 객관적으로 수치화할 수단과 방법 부족
    실시간으로 변하는 부분을 반영하기가 어려움
    미결인 상태로 남겨둠.

개선 방안 (2)

- 슬리피지를 고려한 수정 전략 사용

(매수시점을 당일 시초가로 변경하여 슬리피지 회피 가정)

ex) 현대자동차 주가

** 전날 종가와 당일 시초가 간의 갭으로 인한 수익률 악화, 승률 감소

KOSPI 전체종목 승률
기존 전략 : 73.0198%
수정 전략 : 54.3316%

Thank you

[Q&A]