효과적인 우리사주 운영을 위한 주가 예측 모델 개발

2025. 01. 24.

신재욱 사원

목차

Part 1

분석 개요

1.1 분석 배경 및 목적

분석 배경 및 목적

분석 배경

- 우리사주는 기업의 장기적인 성장과 조합원의 경제적 안정성을 동시에 추구할 수 있는 중요한 제도임
- 운영 과정에서 주가의 급격한 변동성이 리스크 요인으로 작용하며, 매입 및 매도 시점 선택의 어려움이 존재
- 임직원들의 재산 증식과 리스크 관리 측면에서 전략적 접근이 필요한 상황임
- 주가 예측 기술의 발전은 보다 정교하고 정확한 예측을 가능하게 하며, 주가와 같은 금융 데이터의 특성을 효과적으로 반영할 수 있음

분석 목적

- 주가 변동성을 기반으로 매입 및 매도 시점을 최적화하여 조합원의 재산 증식과 기업의 안정적인 운영을 동시에 달성
- 예측 데이터를 조합원에게 투명하게 공유함으로써, 우리사주 제도에 대한 신뢰를 높이고, 참여율을 제고할 수 있음
- 우리사주 운영뿐만 아니라, 재무 관리, 리스크 평가, 투자 전략 등 다양한 영역에 AI 기술을 확장할 가능성을 모색

Part 2

데이터 수집 및 EDA

2.1 데이터 수집
2.2 파생변수 생성
2.3 데이터 EDA

데이터 수집

분석 데이터

- 데이터의 양을 고려해 코리안리 주가 데이터를 수집
- 한국거래소(KRX) 정보데이터시스템을 통해 2000.01.01 ~ 2025.01.22 기간의 데이터 확보
- 데이터가 충분해야 좋은 성능을 발휘하므로 많은 데이터를 확보하고자 함

날짜	종가	시가	고가	저가	거래량	등락률
2000-01-04	764	671	764	671	1.79M	14.71%
2000-01-05	740	714	777	714	1.79M	-3.14%
2000-01-06	716	782	782	714	1.33M	-3.24%
2000-01-07	735	740	751	716	2.79M	2.65%
...	...	...	...	...	...	...
2025-01-17	8,250	8,110	8,250	8,100	131.72K	1.73%
2025-01-20	8,150	8,300	8,300	8,100	124.09K	-1.21%
2025-01-21	8,100	8,210	8,210	8,080	176.81K	-0.61%
2025-01-22	8,000	8,100	8,160	7,970	255.80K	-1.23%

파생변수 생성

ta 패키지를 활용한 주가 보조지표 생성

- 시계열 특성을 사용하는 DL 모델은 '종가'만을 사용하기에 파생변수가 필요 없음
- 하지만 만약 ML 모델을 사용한다면 다양한 주가 보조지표는 주가 예측 및 데이터 이해에 도움이 될 것이라고 판단

주가 보조지표

- Momentum Indicators(모멘텀 지표)
  - Relative Strength Index (RSI), Ultimate Oscillator (UO), Stochastic Oscillator (SR), Williams %R (WR), Rate of Change (ROC)
- Volume Indicator(거래량 지표)
  - On-Balance Volume (OBV)
- Volatility Indicators(변동성 지표)
  - Bollinger High Bands (BHB), Bollinger Low Bands (BLB)
- Trend Indicators(추세 지표)
  - Exponential Moving Average (EMA), Weighted Moving Average (WMA), Moving Average Convergence Divergence (MACD), Average Directional Movement Index (ADX), Commodity Channel Index (CCI)

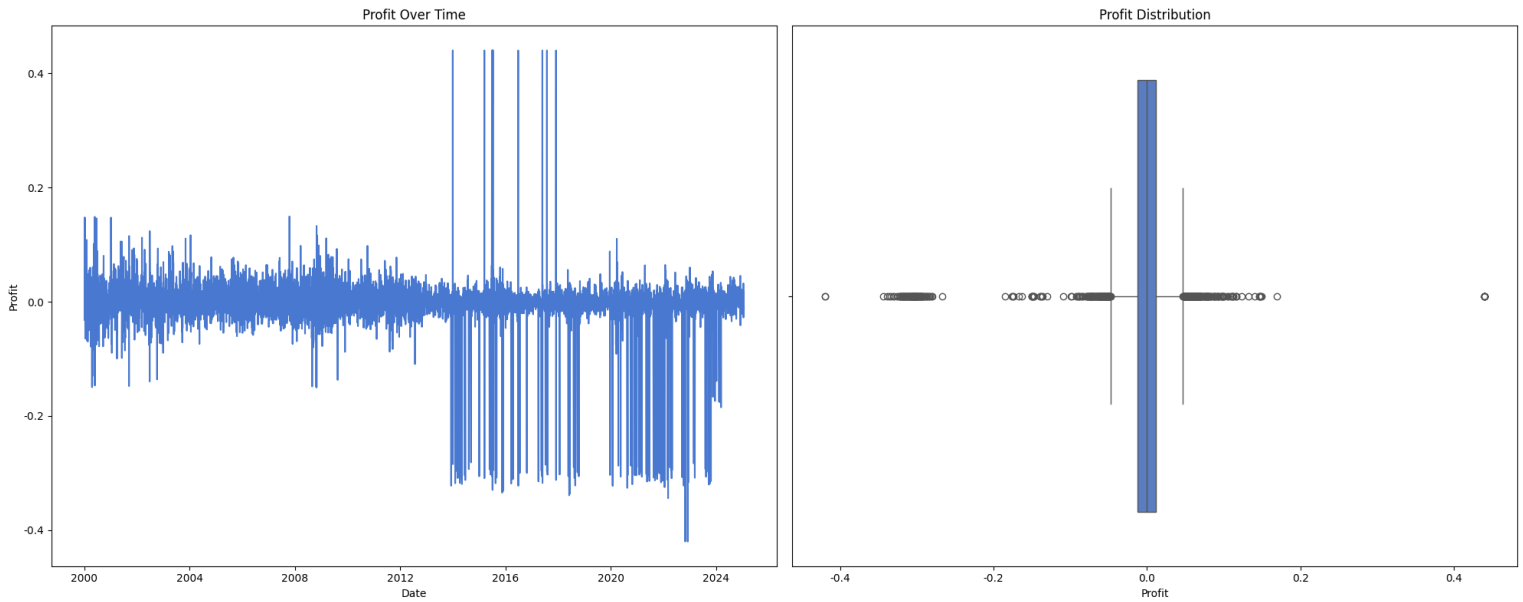
데이터 EDA

결측치 및 수치 분포 확인

- 결측치는 존재하지 않았음
- 평균, 표준편차, 사분위수, 최댓값, 최솟값 등을 확인

[등락률]

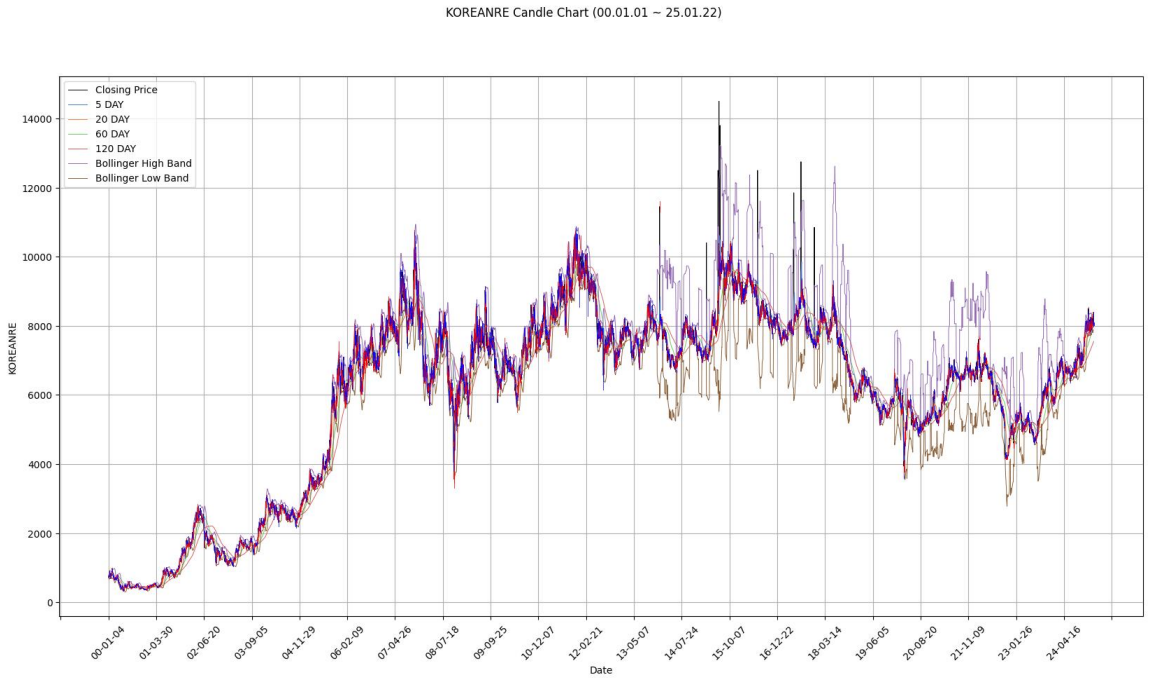
- 진동 형태로, outlier가 존재하며 2024년부터는 안정적이게 변동함



데이터 EDA

[시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균

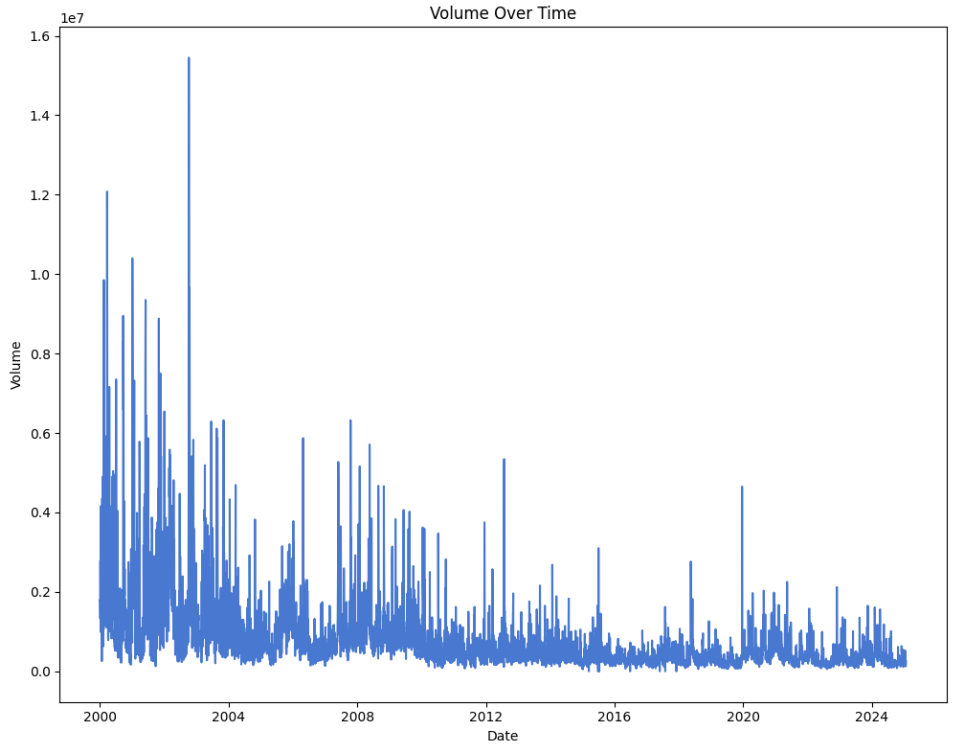
- 주가를 캔들 차트로 시각화하여 확인 (전체 기간, 2024년~)

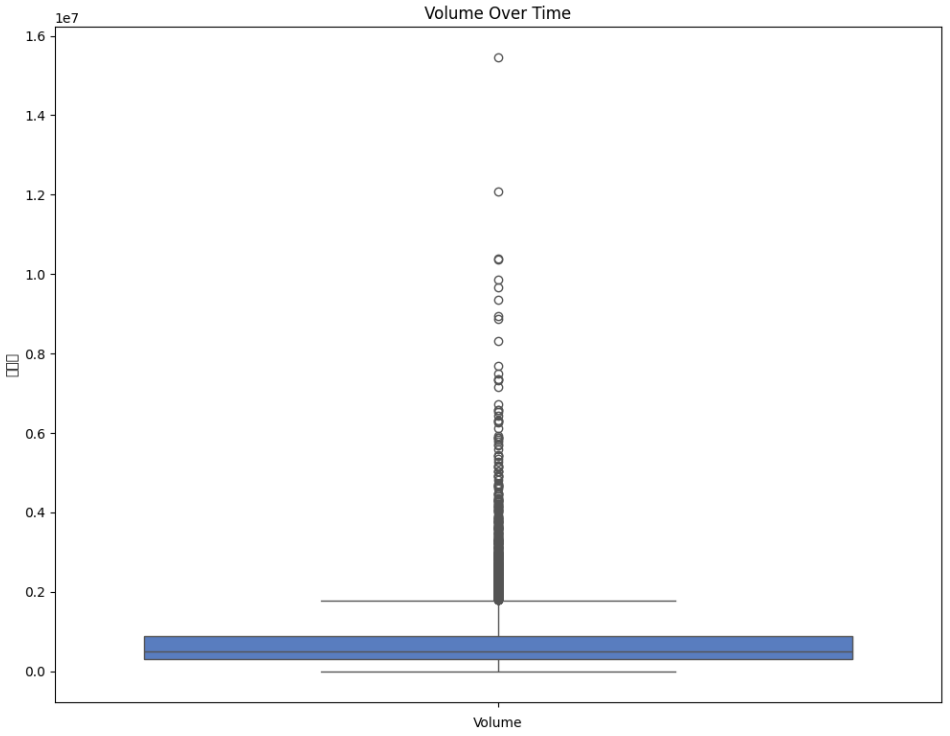


데이터 EDA

[거래량]

- 진동 형태로, outlier가 존재하며 2000년대 초반 거래량이 상대적으로 활발했음





데이터 EDA

VIF, OLS 확인

- 10 이상이면 다중공선성이 존재하는 것으로 간주되며, 지나치게 큰 값들이 있는 걸 보아 ML 모델 사용 시 변수 제거가 불가피함
- OLS의 p-value 또한 0.05보다 지나치게 높은 변수들이 존재

	features	VIF Factor
0	시가	1278.45097
1	고가	1046.21115
2	저가	1083.01789
3	거래량	1.67095
4	수익률	1.67528
5	RSI	6.38534
6	UO	1.84296
7	SR	71.37700
8	WR	78.72572
9	ROC	2.64889
10	OBV	3.71138
11	BHB	820.09293
12	BLB	590.90640
13	EMA	12954.99836
14	WMA	5217.34441
15	MACD	4.70483
16	ADX	1.32335
17	CCI	6.43671

[VIF 결과]

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0224	0.002	-11.296	0.000	-0.026	-0.018
시가	-0.3730	0.009	-39.363	0.000	-0.392	-0.354
고가	0.6554	0.008	77.438	0.000	0.639	0.672
저가	0.6145	0.009	69.432	0.000	0.597	0.632
거래량	-2.522e-06	9.97e-07	-2.530	0.011	-4.48e-06	-5.68e-07
수익률	382.0288	17.801	21.462	0.000	347.134	416.924
RSI	1.0928	0.181	6.054	0.000	0.739	1.447
UO	1.3439	0.092	14.686	0.000	1.165	1.523
SR	-0.6055	0.109	-5.571	0.000	-0.819	-0.392
WR	1.6310	0.095	17.204	0.000	1.445	1.817
ROC	-0.5266	0.122	-4.309	0.000	-0.766	-0.287
OBV	-2.262e-09	7.79e-09	-0.290	0.771	-1.75e-08	1.3e-08
BHB	-0.0681	0.007	-10.111	0.000	-0.081	-0.055
BLB	-0.0814	0.007	-11.694	0.000	-0.095	-0.068
EMA	0.3748	0.030	12.466	0.000	0.316	0.434
WMA	-0.1232	0.019	-6.466	0.000	-0.160	-0.086
MACD	-0.0186	0.012	-1.555	0.120	-0.042	0.005
ADX	-0.3879	0.087	-4.449	0.000	-0.559	-0.217
CCI	-0.2435	0.017	-14.181	0.000	-0.277	-0.210

[OLS 결과]

Part 3

데이터 전처리

3.1 데이터 축소
3.2 최종 데이터 생성

데이터 축소

VIF를 활용한 다중공선성 제거 (ML 모델 사용 시)

- VIF(분산팽창계수)가 10 이상일 경우 다중공선성이 존재한다고 파악 → 해당 변수들 제거
- 다중공선성을 제거한다면 회귀 분석 시 계수의 불안정성을 해결할 수 있음

OLS를 이용한 유의미한 변수 파악 (ML 모델 사용 시)

- OLS Regression을 통해 P-value가 0.05 미만인 변수들을 유의미하다고 판단 후 나머지 변수들 제거
- 무의미한 변수 제거를 통해 모델의 과적합을 방지할 수 있음

최종 데이터 생성

전처리 이후 최종 데이터

- 거래량 0인 row(즉, 거래 정지 기간)를 제거하여 최종적으로 6290개 → 6188로 축소

날짜	종가	시가	고가	저가	거래량	등락률
2000-01-04	764	671	764	671	1.79M	14.71%
2000-01-05	740	714	777	714	1.79M	-3.14%
2000-01-06	716	782	782	714	1.33M	-3.24%
2000-01-07	735	740	751	716	2.79M	2.65%
...	...	...	...	...	...	...
2025-01-17	8,250	8,110	8,250	8,100	131.72K	1.73%
2025-01-20	8,150	8,300	8,300	8,100	124.09K	-1.21%
2025-01-21	8,100	8,210	8,210	8,080	176.81K	-0.61%
2025-01-22	8,000	8,100	8,160	7,970	255.80K	-1.23%

df.shape = (6290, 7)

→

날짜	종가
2000-01-04	764
2000-01-05	740
2000-01-06	716
2000-01-07	735
...	...
2025-01-17	8,250
2025-01-20	8,150
2025-01-21	8,100
2025-01-22	8,000

df.shape = (6188, 2)

Part 4

모델링

4.1 모델 소개
4.2 최적 모델 선정
4.3 최종 예측 결과

모델 소개

모델 비교 및 선택

- 00.01.04 ~ 25.01.22 까지의 데이터를 사용
- 2019년까지의 데이터를 Train, 2020년 이후의 데이터를 Test로 설정해 MSE 비교를 통해 최종 모델을 선택

최종 예측 방식

- 등락률을 바로 예측하는 모델을 시도해보았으나 지나친 진동함수를 예측하는 것은 어려움을 깨달음
- (25.01.23, 25.01.24, 25.01.31)의 종가를 예측한 후 이를 통해 등락률을 계산할 예정

DL 모델 concept

- 전처리 완료된 데이터의 column들 중 '종가' 만을 사용
- time steps=5로 설정하여 과거 5일의 데이터를 활용해 미래 1일의 가격 예측하는 many-to-one 방식을 선택
- 10일 반영 1일 예측, 5일 반영 2일 예측 등 다양한 방식을 시도했지만 5일 반영 1일 예측이 가장 안정적
- RNN, LSTM, GRU 는 3차원의 배열을 입력값으로 요구: [# of samples, # of time steps, # of features]

모델 소개

모델 훈련을 위한 데이터 변환

① df 맨 아래에 예측을 위해 '일자'를 지정하고 나머지 column들의 값은 0으로 지정한 row들 추가 생성

② Train과 Test 셋 분리

③ MinMaxScaler을 활용한 데이터 scaling 진행

④ 과거 5일의 데이터를 활용해 미래 1일의 가격 예측을 위한 변환(time_steps, for_periods)

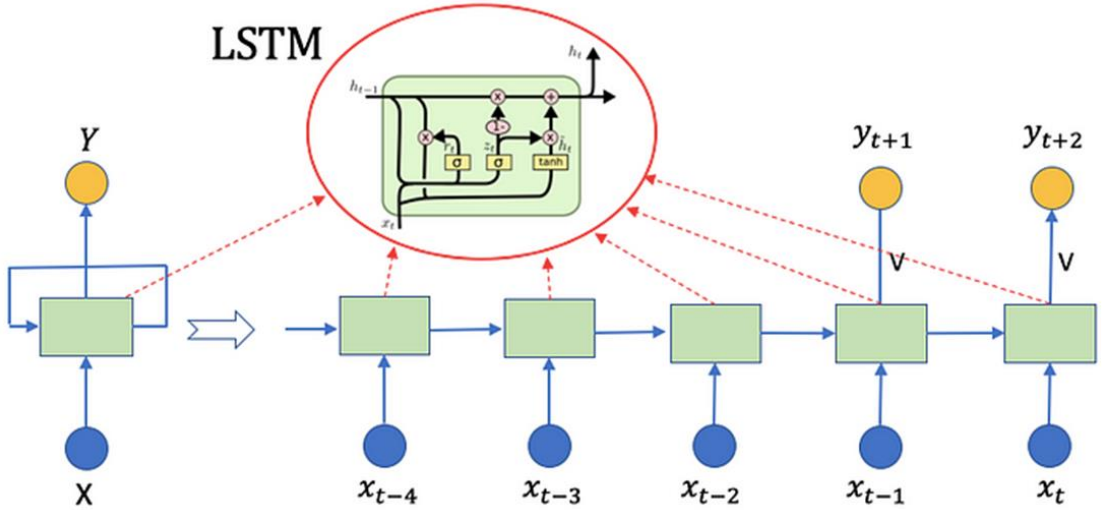
⑤ LSTM, GRU는 3차원의 배열을 입력값으로 요구하기에 구조 reshape 진행

```
def ts_train_test_normalize(all_data, time_steps, for_periods):
    """
    input:
        data: dataframe with dates and price data
    output:
        X_train, y_train: data from 2000-2019
        X_test : data from 2020
        sc : MinMaxScaler fit to the training data
    """
    # train, test set 생성
    ts_train = all_data[:'2019'].iloc[:,0:1].values
    ts_test = all_data['2020':].iloc[:,0:1].values
    ts_train_len = len(ts_train)
    ts_test_len = len(ts_test)

    # data scaling
    from sklearn.preprocessing import MinMaxScaler
    sc = MinMaxScaler(feature_range=(0,1))
    ts_train_scaled = sc.fit_transform(ts_train)

    # s samples and t time steps에 대한 training data 생성
    X_train = []
    y_train = []
    for i in range(time_steps, ts_train_len-1):
        X_train.append(ts_train_scaled[i-time_steps:i, 0])
        y_train.append(ts_train_scaled[i:i+for_periods, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)

    # X_train 구조 reshape
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1 ))

    inputs = pd.concat((all_data["종가"][:'2019'], all_data["종가"]['2020':]), axis=0).values
    inputs = inputs[len(inputs)-len(ts_test)-time_steps:]
    inputs = inputs.reshape(-1,1)
    inputs = sc.transform(inputs)

    # X_test 생성
    X_test = []
    for i in range(time_steps, ts_test_len + time_steps - for_periods):
        X_test.append(inputs[i-time_steps:i,0])

    X_test = np.array(X_test)
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

    return X_train, y_train , X_test, sc
```

모델 소개

LSTM 모델

- 좀 더 긴 시간동안 RNN의 메모리를 유지하기 위해 제안된 구조
- 추가적인 게이트, 입력 및 출력 게이트를 도입함으로써 기울기 소실 문제를 해결
- Hidden state와 Cell state에서 과거 정보가 순환
- Cell state는 Hidden state와 달리 다음 층으로 전달되지 않으며 또한 LSTM은 Gate를 통해서 정보를 통제



모델 소개

LSTM 모델 생성

① 모델 구조 생성

- Sequential 지정 후 .add()를 통해 LSTM 층 추가
- LSTM 층에 input_shape 인자를 전달
- 출력 뉴런과 연결된 Dense 층을 마지막에 추가

② Compile

- 모델 학습 전 모델 학습 환경 설정
- optimizer로 확률적 경사 하강법(SGD)을 사용
- 손실함수 loss로 mean_squared_error 사용

③ Fitting

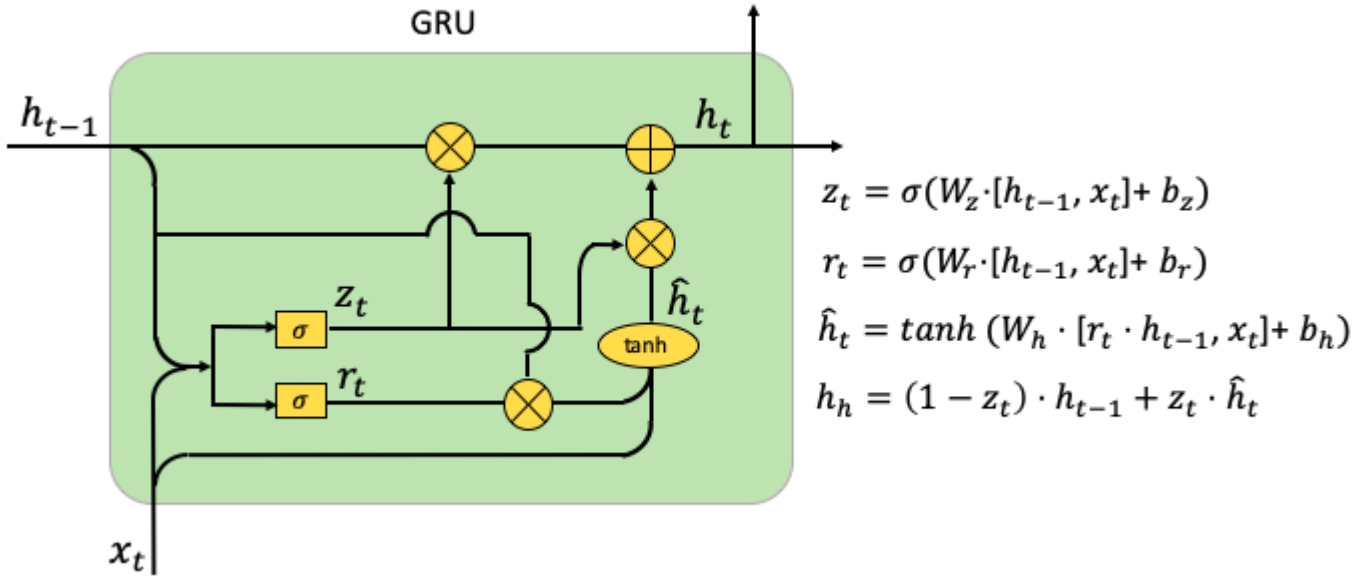
- epoch와 batch_size를 지정 후 모델을 학습

```python
def LSTM_model(X_train, y_train, X_test, sc):
    # 필요한 모듈 임포트
    from keras.models import Sequential
    from keras.layers import Dense, LSTM
    from keras.optimizers import SGD

    # LSTM 모델 정의
    my_LSTM_model = Sequential()
    my_LSTM_model.add(LSTM(units=50,
                           return_sequences=True,
                           input_shape=(X_train.shape[1], 1),
                           activation='tanh'))
    my_LSTM_model.add(LSTM(units=50, activation='tanh'))
    my_LSTM_model.add(Dense(units=2))

    # 옵티마이저 정의 (lr 대신 learning_rate 사용)
    optimizer = SGD(learning_rate=0.01, momentum=0.9, nesterov=False)

    # 모델 컴파일
    my_LSTM_model.compile(optimizer=optimizer, loss='mean_squared_error')

    # train set fitting
    my_LSTM_model.fit(X_train, y_train, epochs=50, batch_size=150, verbose=0)

    # 예측
    LSTM_prediction = my_LSTM_model.predict(X_test)
    LSTM_prediction = sc.inverse_transform(LSTM_prediction)

    return my_LSTM_model, LSTM_prediction
```

모델 소개

GRU 모델

- 기울기 소실 문제(vanishing gradient problem)를 해결하는 것이 목적
- LSTM을 개선한 모델로 LSTM보다 파라미터가 더욱 적어 연산 비용도 적고, 모델도 간단해 학습 속도가 더 빠르지만 비슷한 성능을 내는 모델
- Forget Gate와 Input Gate를 합쳐 Update Gate를 만들고, Reset Gate를 추가함



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\hat{h}_t = tanh\,(W_h \cdot [r_t \cdot h_{t-1}, x_t] + b_h)$$

$$h_h = (1 - z_t) \cdot h_{t-1} + z_t \cdot \hat{h}_t$$

모델 소개

GRU 모델 생성

① 모델 구조 생성

- Sequential 지정 후 .add()를 통해 GRU 층 추가
- GRU 층에 input_shape 인자를 전달
- 출력 뉴런과 연결된 Dense 층을 마지막에 추가

② Compile

- 모델 학습 전 모델 학습 환경 설정
- optimizer로 확률적 경사 하강법(SGD)을 사용
- 손실함수 loss로 mean_squared_error 사용

③ Fitting

- epoch와 batch_size를 지정 후 모델을 학습

```python
def GRU_model(X_train, y_train, X_test, sc):
    # 모델 생성
    from keras.models import Sequential
    from keras.layers import Dense, GRU
    from keras.optimizers import SGD

    # GRU층 구조
    my_GRU_model = Sequential()
    my_GRU_model.add(GRU(units=50,
                         return_sequences=True,
                         input_shape=(X_train.shape[1], 1),
                         activation='tanh'))
    my_GRU_model.add(GRU(units=50, activation='tanh'))
    my_GRU_model.add(Dense(units=2))

    # 옵티마이저 정의 (lr 대신 learning_rate 사용)
    optimizer = SGD(learning_rate=0.01, momentum=0.9, nesterov=False)

    # 모델 컴파일
    my_GRU_model.compile(optimizer=optimizer, loss='mean_squared_error')

    # train set fitting
    my_GRU_model.fit(X_train, y_train, epochs=50, batch_size=150, verbose=0)

    # 예측
    GRU_prediction = my_GRU_model.predict(X_test)
    GRU_prediction = sc.inverse_transform(GRU_prediction)

    return my_GRU_model, GRU_prediction
```

최적 모델 선정

최종 모델

- 2020.01.01 ~ 2025.01.22 기간의 실제 예측에 대한 종가 MSE 비교를 통해 DL 모델들 중 GRU 모델을 선택
- **GRU 모델 MSE = 19290.90085**
- LSTM 모델 MSE = 27645.14467

[GRU 모델 예측 결과]

[LSTM 모델 예측 결과]

최종 예측 결과

예측 과정 및 결과

① 3일치 예측을 위해 0을 담고 있는 row들 생성

② 과거 5일의 데이터를 이용해 1일을 예측하도록 하는 데이터 변환 (e.g. 23일을 예측하도록 Train, Test 지정)

③ GRU 모델 생성

④ GRU 모델 훈련 이후 23일 종가 예측 진행

⑤ 예측된 종가를 0으로 지정된 23일 종가에 입력하고 다시 ①부터 반복해 그 다음 날을 예측

일자	종가	등락률(%)
2025-01-22	8000.00000	-1.235
2025-01-23	8103.62842	1.295
2025-01-24	8111.99658	0.103
2025-01-31	8084.63232	-0.337

Part 5
결론

5.1 활용방안 및 기대효과
5.2 의의 및 한계

활용방안 및 기대효과

활용방안

- 우리사주 조합의 주식 매입 및 매도 시점을 효율적으로 결정할 수 있도록 예측 결과를 활용
- 조합원들에게 예측된 주가 동향을 제공할 수 있음
- 예측 결과를 통해 하락 구간에서 자금 투입을 조정하거나, 상승 구간에서 매도 기회를 포착해 리스크를 관리
- 인공지능 기반의 예측 모델을 활용하여 의사결정 과정을 객관화하고 투명성을 강화

기대효과

- 주가 예측을 통해 매입 타이밍을 최적화함으로써 조합원이 실질적인 자산 증식을 경험할 가능성을 높임
- 조합원의 신뢰도와 만족도를 향상시켜 우리사주 운영 참여율 증가
- 급격한 하락이나 외부 충격 발생 시 사전 대응 가능성을 높일 수 있음
- AI 기반 예측 모델 도입 사례를 축적하여 기업 내 디지털 전환 가속화에 기여

의의 및 한계

의의

- 딥러닝 알고리즘을 적용하여 기업 실무에 직접 활용되는 모델 개발
- 주가 예측 모델 개발을 통해 기업 내 데이터 기반 의사결정 문화를 조성하는 기회를 마련
- 더 발전된 SOTA 모델 적용을 통해 보다 더 정확한 예측의 가능성을 기대할 수 있음

한계

- GRU 및 LSTM은 과거 데이터를 기반으로 예측하지만, 돌발적인 외부 변수(정책 변화, 경제 위기 등)에 대한 예측은 한계가 있음
- 딥러닝 모델은 중·장기적인 관점보다는 단기적인 예측에 강점이 있어, 장기적인 전략 수립에는 한계가 있을 수 있음

Part 6

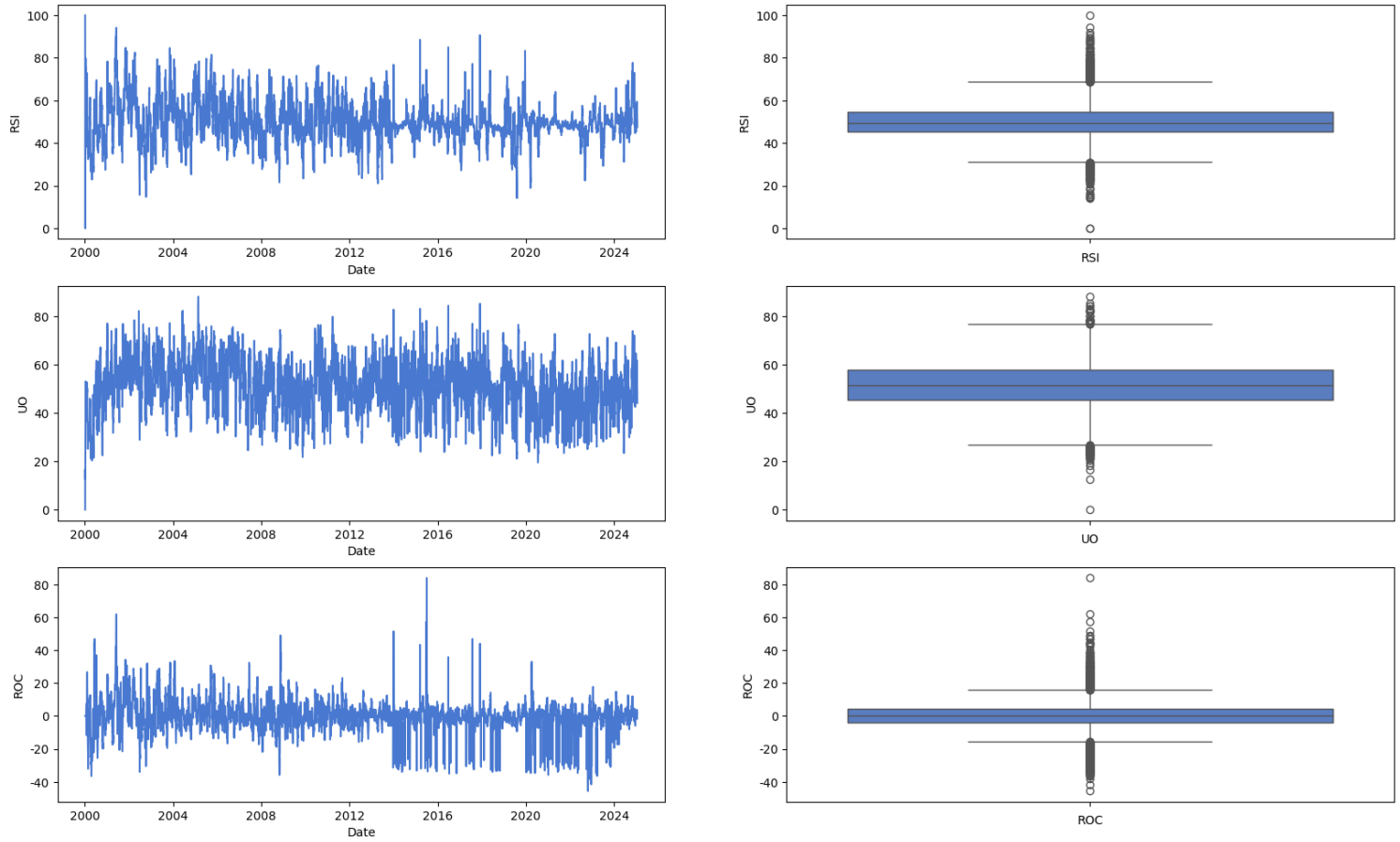
추가 자료

6.1 데이터 EDA

데이터 EDA

[Relative Strength Index (RSI), Ultimate Oscillator (UO), Rate of Change (ROC)]

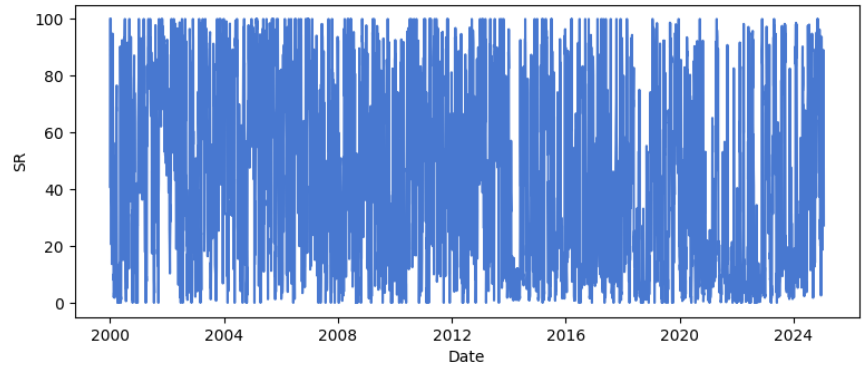
- 진동 형태로, outlier가 존재하며, RSI, UO와 달리 ROC는 outlier가 더 심함

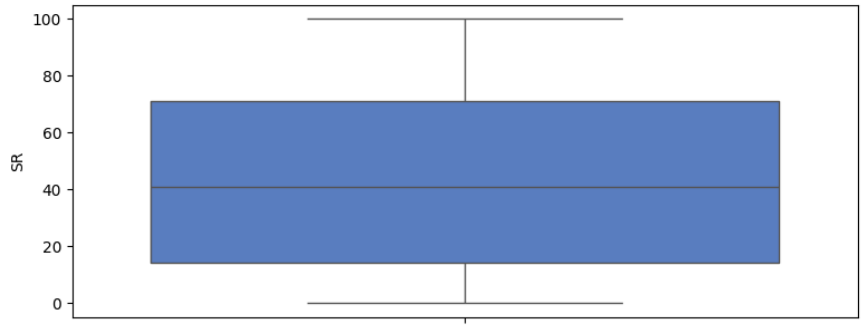


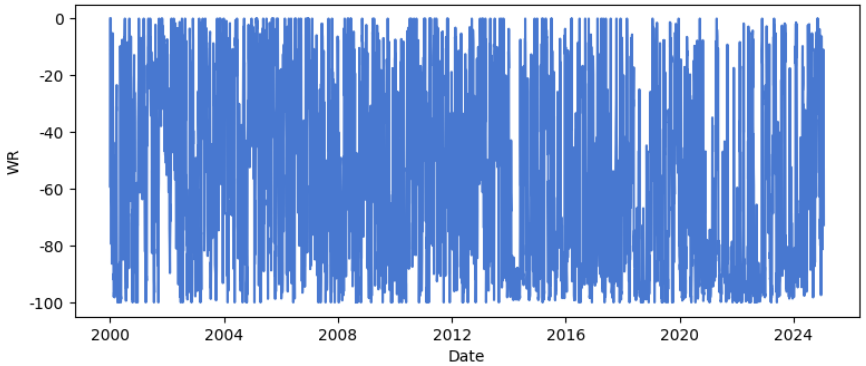
데이터 EDA

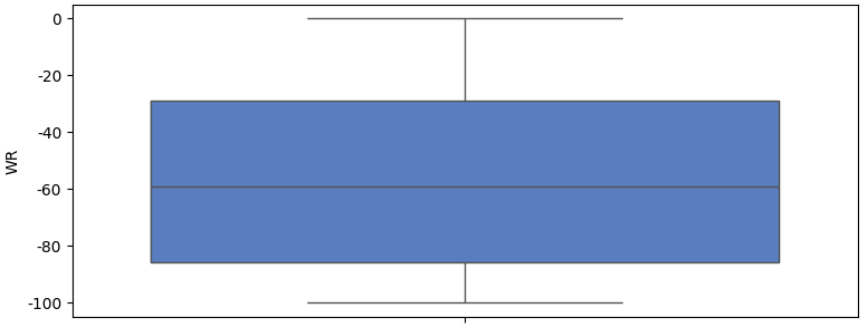
[Stochastic Oscillator (SR), Williams %R (WR)]

- 가장 진동성이 큰 지표로 확인되며, outlier가 존재하지 않음





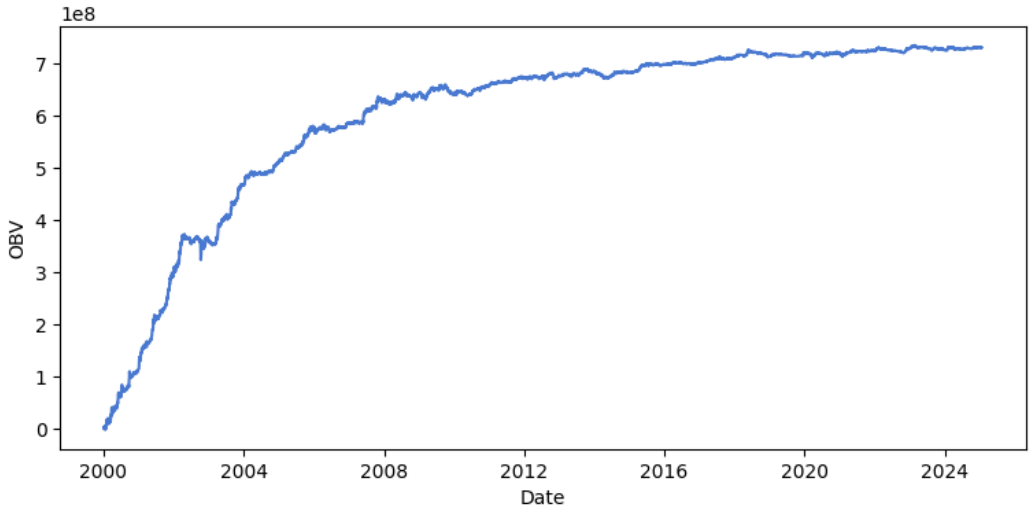


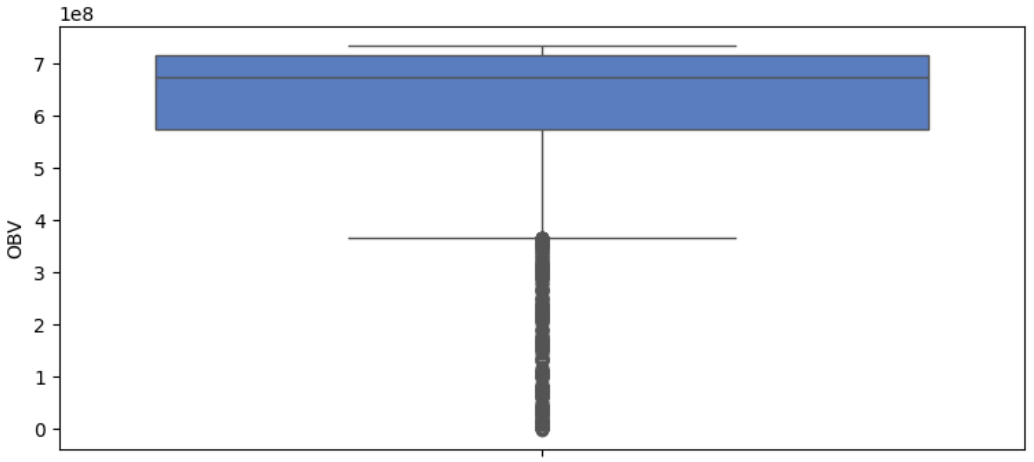


데이터 EDA

[On-Balance Volume (OBV)]

- 우상향의 형태를 띄며, boxplot으로 확인했을 때 낮은 값들이 outlier로 판정됨

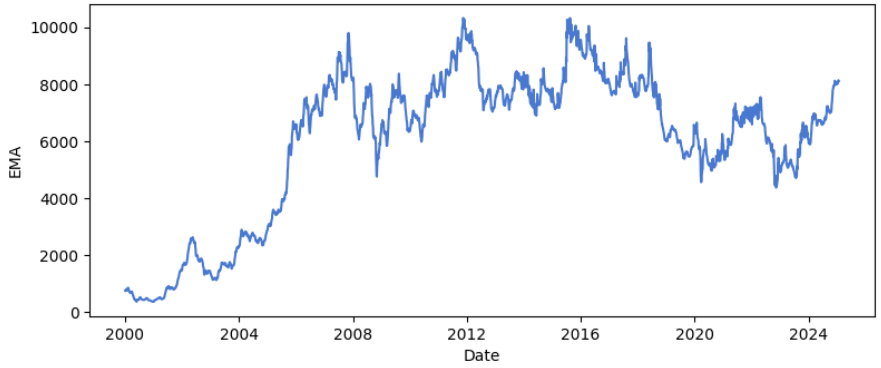


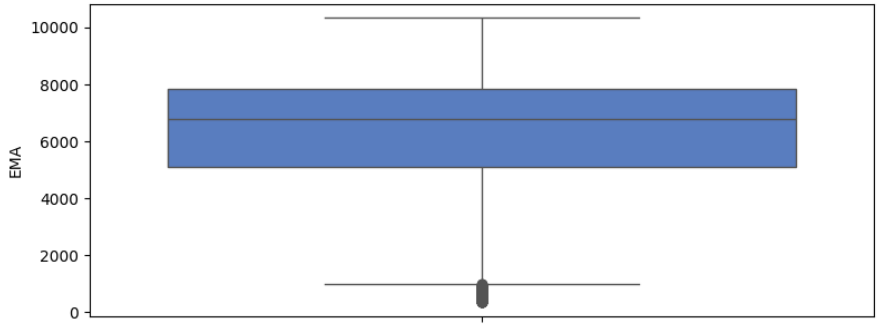


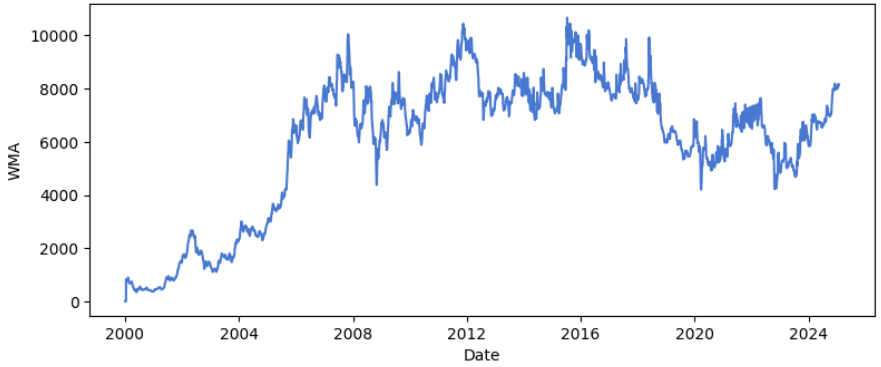
데이터 EDA

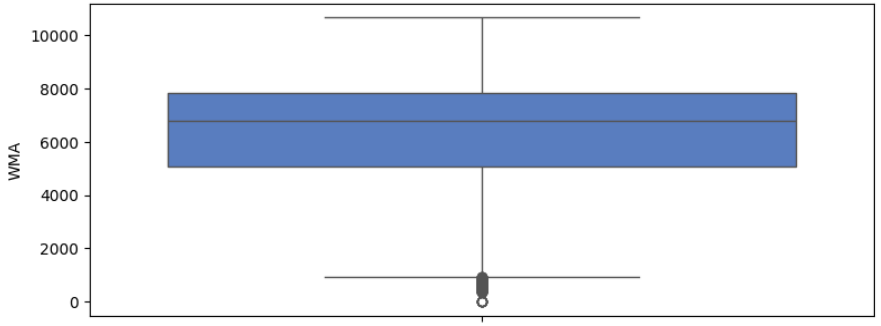
[Exponential Moving Average (EMA), Weighted Moving Average (WMA)]

- 이동평균 지표인 만큼 앞서 확인한 실제 캔들 차트의 모습과 매우 유사한 것을 확인 가능함





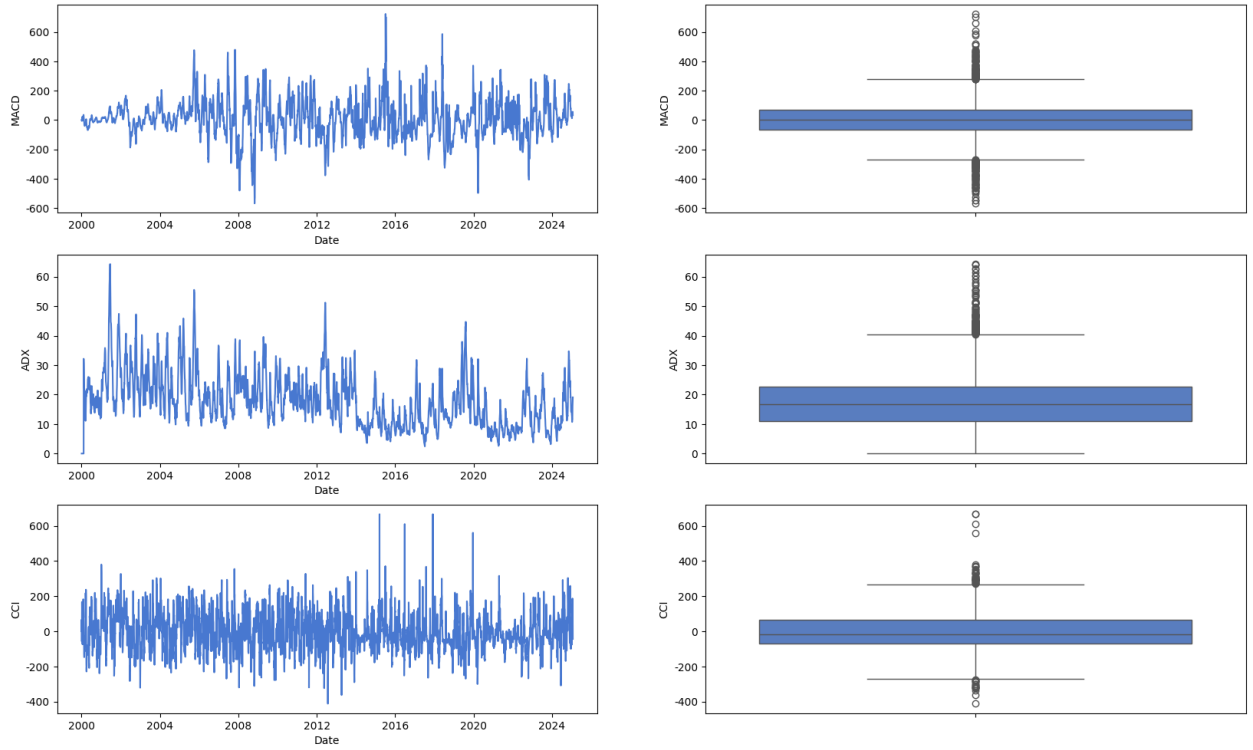




데이터 EDA

[Moving Average Convergence Divergence (MACD), Average Directional Movement Index (ADX), Commodity Channel Index (CCI)]

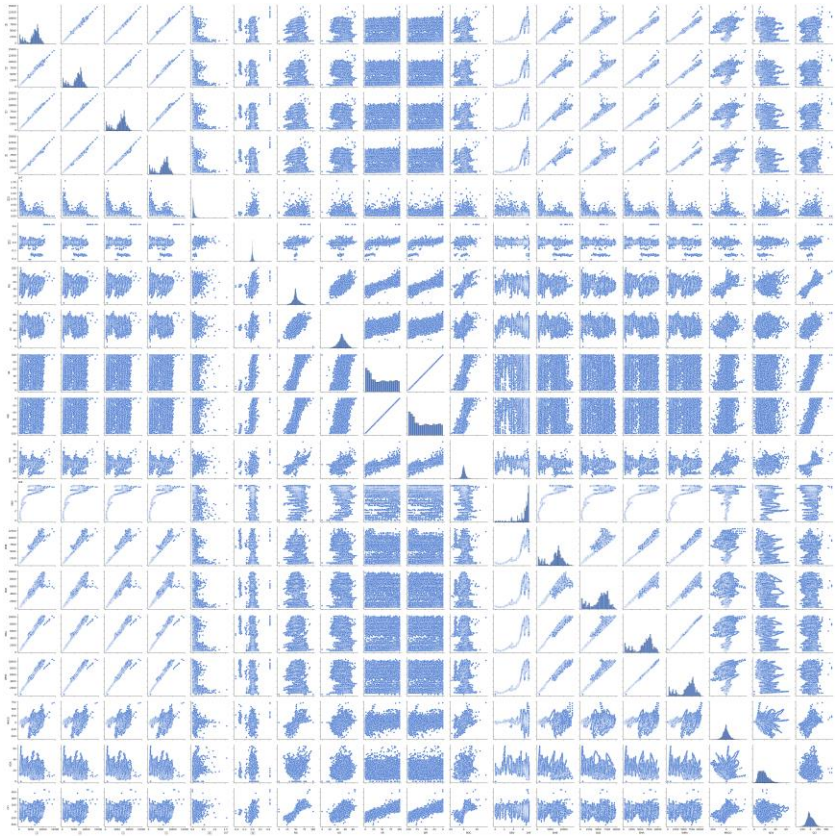
- 진동 형태로, outlier가 존재함



데이터 EDA

Pairplot 생성

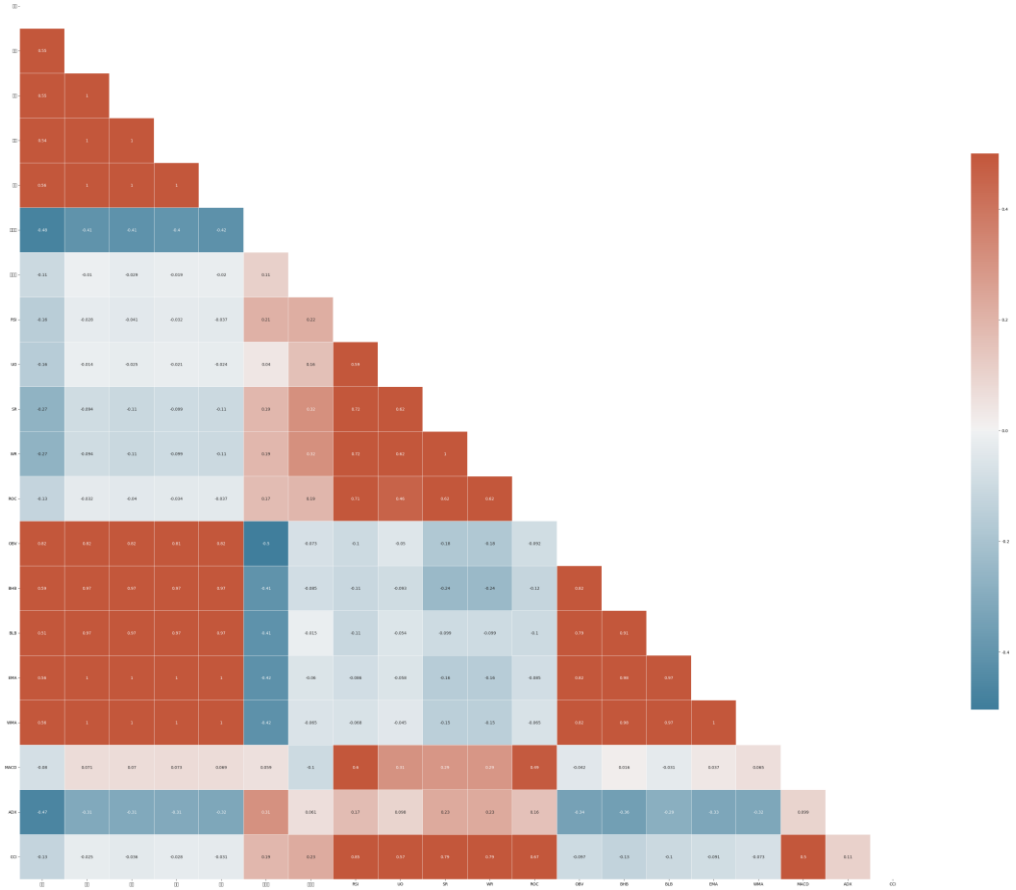
- 지나치게 선형인 그래프를 통해 데이터 전처리를 통한 변수 제거가 불가피함을 확인



데이터 EDA

Heatmap 생성

- 수치가 1.00이 나오는 경우가 존재하며, 데이터 전처리를 통한 변수 제거가 불가피함을 확인



End of Document