

수익률 예측

학번 : 2019147025
이름 : 신재욱

Contents

- I. 데이터 선택
- II. 데이터 탐색
- III. 전처리 방법
- IV. 모델 소개
- V. 분류 예측
 - I. 전처리 적용
 - II. 모델 학습
 - III. 최종 예측

데이터 선택

01. 종목 선택 및 근거

(1) 코스피 200 지수

(2) 코스피 및 코스닥 종목

- 시가총액이 높은 순서로 선정 : 시총이 낮은 주식의 경우 작전의 빈도가 높아 예측이 불가능하다고 판단
- 데이터의 양을 고려해 2000년 부근에 상장되어 있던 주식만을 고려
- 주식의 상승 및 하락이 충분히 다양한 수치로 반영이 되어있어야 좋은 성능을 낼 것이라고 생각

코스피 : 삼성전자(코스피 시가총액 1위), SK 하이닉스(코스피 시가총액 3위)

코스닥 : 리노공업(코스닥 시가총액 10위), CJ ENM(코스닥 시가총액 15위)

데이터 선택

02. 분석에 사용할 입력 데이터 및 근거

(1) 분석 로드맵

- ML 모델 : RandomForest, LightGBM, XGBoost, GradientBoost 모델 구현
 - DL 모델 : GRU, LSTM 모델 구현
- 코스피 200 지수 데이터를 기반으로 ML 모델, DL 모델 총 6가지를 비교하여 최종 모델 선정
- 종가 데이터를 예측한 후 이를 토대로 수익률 계산이 목표

(2) 기본 주가 데이터(코스피 200)

- 한국거래소(KRX) 정보데이터시스템을 통해 99.06.01~22.10.26 기간의 데이터 다운로드
- DL 모델의 경우 데이터가 충분해야 좋은 성능을 발휘하므로 많은 데이터를 확보하고자 함

	일자	종가	대비	등락률	시가	고가	저가	거래량	거래대금	상장시가총액
0	2022/10/26	293.85000	2.27000	0.78000	292.21000	295.28000	291.40000	120348.00000	5804827.00000	1560104520.00000
1	2022/10/25	291.58000	0.11000	0.04000	291.29000	293.75000	290.53000	116351.00000	5506859.00000	1547458881.00000
2	2022/10/24	291.47000	2.90000	1.00000	293.60000	294.24000	290.52000	116176.00000	5372981.00000	1547506547.00000
3	2022/10/21	288.57000	-0.04000	-0.01000	287.68000	290.38000	287.44000	102369.00000	4677515.00000	1531346564.00000
4	2022/10/20	288.61000	-2.68000	-0.92000	289.33000	290.07000	286.50000	136904.00000	6043941.00000	1532345342.00000

▲ 코스피 200 기본 주가 데이터

데이터 선택

02. 분석에 사용할 입력 데이터 및 근거

(3) 파생 입력변수 생성

- 시계열 특성을 사용하는 DL 모델은 '종가'만을 사용하기에 파생변수가 필요 없음
- ML 모델의 경우 다양한 추가 파생변수들을 통해 다중회귀분석을 시도하고자 함
→ 주가 보조지표를 사용한다면 주가 예측에 도움이 될 것이라고 판단

ta 패키지를 활용한 주가 보조지표 생성

- 보조지표의 경우 기술적 분석 강의안을 통해 배웠던 지표들을 생성

① Momentum Indicators(모멘텀 지표)

: Relative Strength Index (RSI), Ultimate Oscillator (UO), Stochastic Oscillator (SR), Williams %R (WR), Rate of Change (ROC)

② Volume Indicator(거래량 지표) : On-Balance Volume (OBV)

③ Volatility Indicators(변동성 지표) : Bollinger High Bands (BHB), Bollinger Low Bands (BLB)

④ 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 선택

02. 분석에 사용할 입력 데이터 및 근거

(3) 파생 입력변수 생성

- ta 패키지를 통해 생성된 추가 입력 변수들의 결과와 관련 code
- EDA와 데이터 전처리 과정을 거쳐 data reducing 작업 예정

```
In [8]: # Relative Strength Index (RSI)
df['RSI'] = ta.momentum.rsi(close=C, fillna=True)
# Ultimate Oscillator (UO)
df['UO'] = ta.momentum.ultimate_oscillator(high=H, low=L, close=C, fillna=True)
# Stochastic Oscillator (SR)
df['SR'] = ta.momentum.stoch(close=C, high=H, low=L, fillna=True)
# Williams %R (WR)
df['WR'] = ta.momentum.williams_r(high=H, low=L, close=C, fillna=True)
# Rate of Change (ROC)
df['ROC'] = ta.momentum.roc(close=C, fillna=True)
```

Volume Indicator 생성

```
In [9]: # On-Balance Volume (OBV)
df['OBV'] = ta.volume.on_balance_volume(close=C, volume=V, fillna=True)
```

Volatility Indicators 생성

```
In [10]: # Bollinger High Bands (BHB)
df['BHB'] = ta.volatility.bollinger_hband(close=C, fillna=True)
# Bollinger Low Bands (BLB)
df['BLB'] = ta.volatility.bollinger_lband(close=C, fillna=True)
```

Trend Indicators 생성

```
In [11]: # Exponential Moving Average (EMA)
df['EMA'] = ta.trend.ema_indicator(close=C, fillna=True)
# Weighted Moving Average (WMA)
df['WMA'] = ta.trend.wma_indicator(close=C, fillna=True)
# Moving Average Convergence Divergence (MACD)
df['MACD'] = ta.trend.macd(close=C, fillna=True)
# Average Directional Movement Index (ADX)
df['ADX'] = ta.trend.adx(high=H, low=L, close=C, fillna=True)
# Commodity Channel Index (CCI)
df['CCI'] = ta.trend.cci(high=H, low=L, close=C, fillna=True)
```

In [12]: df.head()

Out[12]:

날짜	거래대금	상장 시가 증액	RSI	UO	SR	WR	ROC	OBV	BHB	BLB	EMA	WMA	MACD	ADX	CCI
000	1644763.00000	NaN	100.00000	0.00000	77.38359	-22.61641	0.00000	99007.00000	88.27000	88.27000	88.27000	0.00000	0.00000	0.00000	
000	2472286.00000	NaN	100.00000	31.01796	82.94679	-17.05321	0.00000	248231.00000	92.15500	86.97500	88.66846	0.00000	0.20661	0.00000	
000	2272050.00000	NaN	100.00000	41.84100	84.91762	-15.08238	0.00000	396517.00000	92.98395	87.42272	89.10101	0.00000	0.41559	0.00000	
000	2366483.00000	NaN	100.00000	50.23102	94.04040	-5.95960	0.00000	530486.00000	95.31366	87.03634	89.86854	0.00000	0.78279	0.00000	
000	3246448.00000	NaN	100.00000	64.12940	100.00000	-0.00000	0.00000	714953.00000	100.87527	84.98873	91.42108	0.00000	1.52962	0.00000	

데이터 탐색

01. EDA

(1) 결측치 파악 – df.isnull().sum()

- 코스피200, 삼성전자, SK하이닉스, 리노공업, CJ ENM 순서
- 코스피200에만 결측치가 존재함을 확인

결측치 파악	결측치 파악	결측치 파악	결측치 파악	결측치 파악
In [13]: df.isnull().sum()	In [13]: df.isnull().sum()	In [13]: df.isnull().sum()	In [13]: df.isnull().sum()	In [13]: df.isnull().sum()
Out[13]: 일자 0 종가 0 대비 0 수익률 0 시가 0 고가 0 저가 0 거래량 0 거래대금 0 상장시가총액 497 RSI 0 UO 0 SR 0 WR 0 ROC 0 OBV 0 BHB 0 BLB 0 EMA 0 WMA 0 MACD 0 ADX 0 CCI 0 dtype: int64	Out[13]: 일자 0 종가 0 대비 0 수익률 0 시가 0 고가 0 저가 0 거래량 0 거래대금 0 시가총액 0 상장주식수 0 RSI 0 UO 0 SR 0 WR 0 ROC 0 OBV 0 BHB 0 BLB 0 EMA 0 WMA 0 MACD 0 ADX 0 CCI 0 dtype: int64	Out[13]: 일자 0 종가 0 대비 0 수익률 0 시가 0 고가 0 저가 0 거래량 0 거래대금 0 시가총액 0 상장주식수 0 RSI 0 UO 0 SR 0 WR 0 ROC 0 OBV 0 BHB 0 BLB 0 EMA 0 WMA 0 MACD 0 ADX 0 CCI 0 dtype: int64	Out[13]: 일자 0 종가 0 대비 0 수익률 0 시가 0 고가 0 저가 0 거래량 0 거래대금 0 시가총액 0 상장주식수 0 RSI 0 UO 0 SR 0 WR 0 ROC 0 OBV 0 BHB 0 BLB 0 EMA 0 WMA 0 MACD 0 ADX 0 CCI 0 dtype: int64	Out[13]: 일자 0 종가 0 대비 0 수익률 0 시가 0 고가 0 저가 0 거래량 0 거래대금 0 시가총액 0 상장주식수 0 RSI 0 UO 0 SR 0 WR 0 ROC 0 OBV 0 BHB 0 BLB 0 EMA 0 WMA 0 MACD 0 ADX 0 CCI 0 dtype: int64

데이터 탐색

01. EDA

(2) 수치 분포 파악 – df.describe()

- 평균, 표준편차, 사분위수, 최댓값, 최솟값 등을 확인

전반적인 데이터 수치 분포 파악

	증가	대비	수익률	시가	고가	저가	거래량	거래대금	상장시가총액	RSI	UO
count	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000	5284.00000	5781.00000	5781.00000	5781.00000	5781.00000
mean	215.80319	0.03601	0.00033	215.89303	217.28582	214.22824	154121.04497	3997982.42519	800290650.23543	52.37866	52.65553
std	89.38316	2.92959	0.01558	89.44953	89.70806	89.08640	13943.94961	254259.92695	48592654.4822	12.39929	19.03914
min	58.03000	-16.85000	-0.11960	57.44000	58.03000	57.02000	36813.00000	558857.00000	147860826.00000	12.76033	0.00000
25%	125.82000	-1.32000	-0.09660	125.65000	126.98000	124.37000	82005.00000	247228.00000	55310907.00000	43.63931	45.48391
50%	236.82000	0.14000	0.00070	239.13000	240.46000	239.86000	115968.00000	3429558.00000	959009715.50000	52.45882	52.68727
75%	267.96000	1.50000	0.00780	268.03000	269.36000	269.44000	174516.00000	4847581.00000	1146187400.00000	61.33167	59.71033
max	440.40000	19.41000	0.12230	440.78000	449.04000	438.75000	202137.00000	37379964.00000	2010647147.00000	100.00000	81.96923

▲ 코스피 200

전반적인 데이터 수치 분포 파악

	증가	대비	수익률	시가	고가	저가	거래량	거래대금	시가총액
count	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000	5781.00000
mean	215.80319	0.03601	0.00033	215.89303	217.28582	214.22824	154121.04497	3997982.42519	800290650.23543
std	89.38316	2.92959	0.01558	89.44953	89.70806	89.08640	13943.94961	254259.92695	48592654.4822
min	58.03000	-16.85000	-0.11960	57.44000	58.03000	57.02000	36813.00000	558857.00000	147860826.00000
25%	125.82000	-1.32000	-0.09660	125.65000	126.98000	124.37000	82005.00000	247228.00000	55310907.00000
50%	236.82000	0.14000	0.00070	239.13000	240.46000	239.86000	115968.00000	3429558.00000	959009715.50000
75%	267.96000	1.50000	0.00780	268.03000	269.36000	269.44000	174516.00000	4847581.00000	1146187400.00000
max	440.40000	19.41000	0.12230	440.78000	449.04000	438.75000	202137.00000	37379964.00000	2010647147.00000

▲ SK 하이닉스

전반적인 데이터 수치 분포 파악

	증가	대비	수익률	시가	고가	저가	거래량	거래대금	시가총액	상장주
count	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000
mean	37656.82010	26.39103	0.00107	67158.38315	68771.70022	65528.58743	31655869.52915	22334459520.40008	250966041012.12691	781
std	9690.28673	6336.02486	0.03903	100227.85284	103672.52756	99776.71222	1161975.07547718	17039944020.40073	2489374194455.83303	871
min	2650.00000	-8252.00000	-0.15000	0.00000	0.00000	0.00000	0.00000	0.00000	460121596180.00000	11
25%	23650.00000	-589.00000	-0.01790	22860.00000	23100.00000	22250.00000	333024.00000	11107816400.00000	790316798600.00000	451
50%	37700.00000	0.00000	0.00000	32800.00000	33300.00000	32300.00000	546524.00000	178385618650.00000	157956426000.00000	694
75%	77000.00000	590.00000	0.01170	77200.00000	78400.00000	79100.00000	1063526.00000	27881960400.00000	35473131057250.00000	721
max	71996.00000	8164.00000	0.15000	75452.00000	77048.00000	713670.00000	18372468.00000	1936648003175.00000	10101035120500.00000	5231

▲ 리노공업

전반적인 데이터 수치 분포 파악

	증가	대비	수익률	시가	고가	저가	거래량	거래대금	시가총액	상장주
count	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000	5153.00000
mean	47765.85786	1362.92562	0.02583	47715.21745	48525.57326	46985.29542	74658.71491	2111366535.42150	873309643231.40698	11045406.30
std	47765.85786	1362.92562	0.02583	47715.21745	48525.57326	46985.29542	74658.71491	2111366535.42150	873309643231.40698	11045406.30
min	1477.00000	-10100.00000	-0.09490	1373.00000	1525.00000	1373.00000	2103.00000	3454650.00000	22210500000.00000	6630000.00000
25%	6741.00000	-184.00000	-0.01230	6741.00000	6849.00000	6609.00000	20257.00000	376614650.00000	10228425000.00000	8022300.00000
50%	13420.00000	0.00000	0.00000	12429.00000	12614.00000	12192.00000	34967.00000	950862750.00000	18772120000.00000	8022300.00000
75%	48850.00000	195.00000	0.01280	48750.00000	49550.00000	48500.00000	9961.00000	2101725800.00000	445889774500.00000	15242370.00000
max	21260.00000	18700.00000	0.04490	214100.00000	216700.00000	216700.00000	207700.00000	1670577.00000	49137506500.00000	3240527862000.00000

▲ CJ ENM

전반적인 데이터 수치 분포 파악

	증가	대비	수익률	시가	고가	저가	거래량	거래대금	시가총액	상장주
count	5658.00000	5658.00000	5658.00000	5658.00000	5658.00000	5658.00000	5658.00000	5658.00000	5658.00000	5658.00000
mean	16552.44079	4.14483	0.02971	81231.11567	82083.93632	80867.30182	112682.54865	11054724921.91353	143535119153.01010	10045452.8
std	80782.43486	4351.04989	0.02971	81231.11567	82083.93632	80867.30182	112682.54865	11054724921.91353	15221830971.40091	5653517.7
min	22787.00000	-35800.00000	-0.14970	0.00000	0.00000	0.00000	0.00000	0.00000	10752516550.00000	6021968.00000
25%	11084.25000	-22000.00000	-0.01490	11096.00000	112722.00000	107387.00000	29197.50000	357692875.00000	63230496570.00000	6215518.0
50%	153019.00000	0.00000	0.00000	154884.50000	149650.00000	149650.00000	5801.00000	6393474450.00000	112738968800.00000	8243871.0
75%	215820.25000	2000.00000	0.01450	214886.50000	218100.00000	211113.50000	1080212.75000	10790838575.00000	186208038650.00000	11004531.0
max	428100.00000	25900.00000	0.01207	426100.00000	431600.00000	415400.00000	172937.00000	398678324300.00000	5643381336000.00000	21924154.0

▲ 삼성전자

데이터 탐색

01. EDA

(3) [대비, 수익률] EDA – sns.lineplot(), sns.boxplot()

- [대비, 수익률] 모두 진동 형태를 띠고 있으며 outlier가 존재함
- SK 하이닉스, 리노공업의 경우 두 변수 모두 나머지 3개의 종목과는 다른 lineplot, boxplot을 보여줌

```
[대비, 수익률] EDA
In [15]: plt.figure(figsize=(20,8))

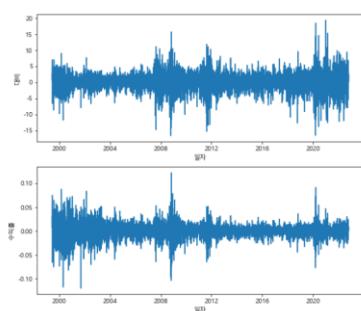
# 대비 lineplot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='대비')

# 대비 boxplot
plt.subplot(2,2,2)
sns.boxplot(df['대비'])

# 수익률 lineplot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='수익률')

# 수익률 boxplot
plt.subplot(2,2,4)
sns.boxplot(df['수익률'])

Out[15]: <AxesSubplot:...
```



▲ 코스피 200

```
[대비, 수익률] EDA
In [15]: plt.figure(figsize=(20,8))

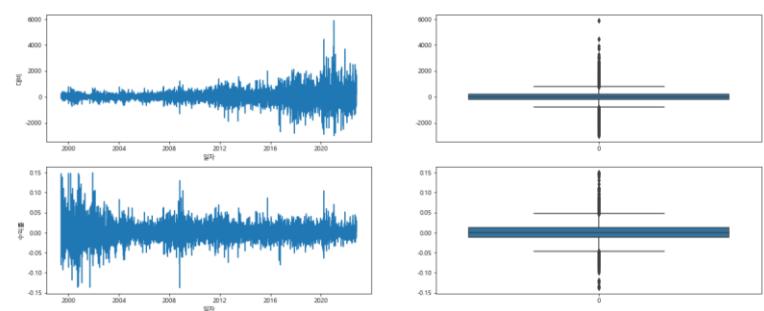
# 대비 lineplot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='대비')

# 대비 boxplot
plt.subplot(2,2,2)
sns.boxplot(df['대비'])

# 수익률 lineplot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='수익률')

# 수익률 boxplot
plt.subplot(2,2,4)
sns.boxplot(df['수익률'])

Out[15]: <AxesSubplot:...
```



▲ 삼성전자

데이터 탐색

01. EDA

(3) [대비, 수익률] EDA – sns.lineplot(), sns.boxplot()

- [대비, 수익률] 모두 진동 형태를 띠고 있으며 outlier가 존재함
- SK 하이닉스, 리노공업의 경우 두 변수 모두 나머지 3개의 종목과는 다른 lineplot, boxplot을 보여줌

[대비, 수익률] EDA

```
In [15]: plt.figure(figsize=(20,8))

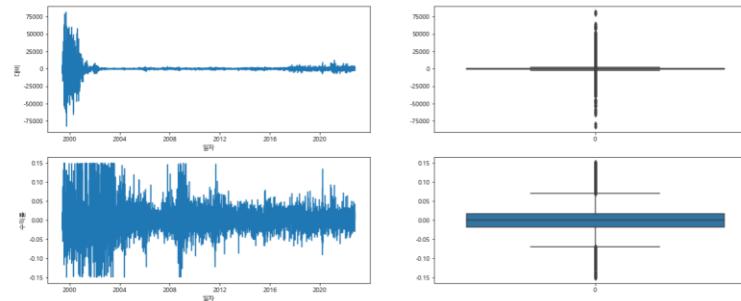
# 대비 Lineplot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='대비')

# 대비 boxplot
plt.subplot(2,2,2)
sns.boxplot(df['대비'])

# 수익률 Lineplot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='수익률')

# 수익률 boxplot
plt.subplot(2,2,4)
sns.boxplot(df['수익률'])

Out[15]: <AxesSubplot:
```



▲ SK 하이닉스

[대비, 수익률] EDA

```
In [15]: plt.figure(figsize=(20,8))

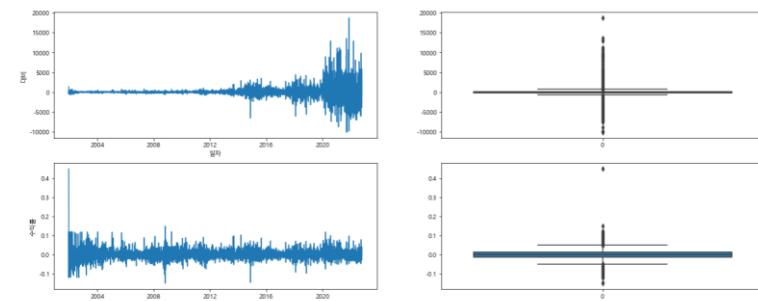
# 대비 Lineplot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='대비')

# 대비 boxplot
plt.subplot(2,2,2)
sns.boxplot(df['대비'])

# 수익률 Lineplot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='수익률')

# 수익률 boxplot
plt.subplot(2,2,4)
sns.boxplot(df['수익률'])

Out[15]: <AxesSubplot:
```



▲ 리노공업

데이터 탐색

01. EDA

(3) [대비, 수익률] EDA – sns.lineplot(), sns.boxplot()

- [대비, 수익률] 모두 진동 형태를 띠고 있으며 outlier가 존재함
- SK 하이닉스, 리노공업의 경우 두 변수 모두 나머지 3개의 종목과는 다른 lineplot, boxplot을 보여줌



▲ CJ ENM

데이터 탐색

01. EDA

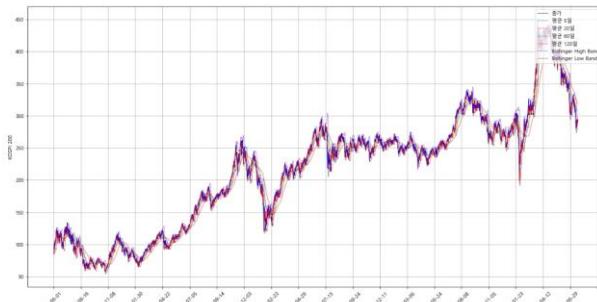
(4) [시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균 EDA – plot()

- 종목별 차트를 시각화하여 확인(전체 기간, 2022년)

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

In [16]: # 전체 데이터 시각화
fig, ax = plt.subplots(figsize=(20,10))
# x축 날짜
xdate = df['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][1:] # 2020-01-01 => 01-01
# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df['종가'], label="종가", linewidth=0.7, color='k')
ax.plot(xdate, df['종가'].rolling(window=5).mean(), label="평균 5일", linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=20).mean(), label="평균 20일", linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=60).mean(), label="평균 60일", linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=120).mean(), label="평균 120일", linewidth=0.7)
ax.plot(xdate, df['종가'], label="Bollinger High Band", linewidth=0.7)
ax.plot(xdate, df['종가'], label="Bollinger Low Band", linewidth=0.7)
candlestick_ohlc(ax,[df['시가'],df['고가'],df['저가'],df['종가']], width=0.5, colorup='r', colordown='b')
fig.suptitle("KOSPI 200 Candle Chart (99.06.01 ~ 22.10.21)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보울 ticker 개수 ~20개이면 1달
ax.yaxis.set_major_locator(ticker.MaxNLocator(10)) # y-축에 보울
plt.xticks(rotation=45) # x-축 끝에 45도 회전
plt.grid() # 그리드 표시
plt.show()

KOSPI_200_Candle_Chart (99.06.01 ~ 22.10.21)
```



```
# 2022년 데이터 시각화
df_2022 = df[df['일자'] >='2022-01-01']
df_2022.reset_index(drop=True, inplace=True)
fig, ax = plt.subplots(figsize=(20,10))

# x축 날짜
xdate = df_2022['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][1:] # 2020-01-01 => 01-01
# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df_2022['종가'], label="종가", linewidth=0.7, color='k')
ax.plot(xdate, df_2022['종가'].rolling(window=5).mean(), label="평균 5일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=20).mean(), label="평균 20일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=60).mean(), label="평균 60일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=120).mean(), label="평균 120일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'], label="Bollinger High Band", linewidth=0.7)
ax.plot(xdate, df_2022['종가'], label="Bollinger Low Band", linewidth=0.7)
candlestick_ohlc(ax,[df_2022['시가'],df_2022['고가'],df_2022['저가'],df_2022['종가']], width=0.5, colorup='r', colordown='b')

fig.suptitle("KOSPI 200 Candle Chart (2022)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보울 ticker 개수 ~20개이면 1달
ax.yaxis.set_major_locator(ticker.MaxNLocator(10)) # y-축에 보울
plt.legend(loc=1) # 레전드 위치
plt.xticks(rotation=45) # x-축 끝에 45도 회전
plt.grid() # 그리드 표시
plt.show()

KOSPI_200_Candle_Chart (2022)
```



데이터 탐색

01. EDA

(4) [시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균 EDA – plot()

- 종목별 차트를 시각화하여 확인(전체 기간, 2022년)

```
In [16]: # 전체 데이터 시각화
fig, ax = plt.subplots(figsize=(20,10))

# x축 설정
xdate = df['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:] # 2020-01-01 => 20-01-01

# 종가 및 5,20,60,120일 이동평균, Bollinger Band 표시
ax.plot(xdate, df['종가'], label="종가", linewidth=0.7, color='k')
ax.plot(xdate, df['종가'].rolling(window=5).mean(), label="평균 5일", linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=20).mean(), label="평균 20일", linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=60).mean(), label="평균 60일", linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=120).mean(), label="평균 120일", linewidth=0.7)
ax.plot(xdate, df['BBL'], label="Bollinger High Band", linewidth=0.7)
ax.plot(xdate, df['BBL'], label="Bollinger Low Band", linewidth=0.7)
candlestick2_ohlc(ax,df['시가'],df['저가'],df['종가'],width=0.5, colorup='r', colordown='b')

fig.suptitle("KOSPI 200 Candle Chart (99.06.01 ~ 22.10.21)")
ax.set_xlabel("Date")
ax.set_ylabel("Close Price (KRW)")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보울 ticker 개수 ~20개이면 1달
ax.legend(loc=1) # legend 위치
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()

KOSPI 200 Candle Chart (99.06.01 ~ 22.10.21)
```



```
In [17]: # 2022년 데이터 시각화
df_2022 = df[df['일자'] >='2022-01-01']
df_2022.reset_index(drop=True, inplace=True)

fig, ax = plt.subplots(figsize=(20,10))

# x축 설정
xdate = df_2022['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:] # 2020-01-01 => 20-01-01

# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df_2022['종가'], label="종가", linewidth=0.7, color='k')
ax.plot(xdate, df_2022['종가'].rolling(window=5).mean(), label="평균 5일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=20).mean(), label="평균 20일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=60).mean(), label="평균 60일", linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=120).mean(), label="평균 120일", linewidth=0.7)
ax.plot(xdate, df['BBL'], label="Bollinger High Band", linewidth=0.7)
ax.plot(xdate, df['BBL'], label="Bollinger Low Band", linewidth=0.7)
candlestick2_ohlc(ax,df_2022['시가'],df_2022['저가'],df_2022['종가'],width=0.5, colorup='r', colordown='b')

fig.suptitle("KOSPI 200 Candle Chart (2022)")
ax.set_xlabel("Date")
ax.set_ylabel("Close Price (KRW)")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보울 ticker 개수 ~20개이면 1달
ax.legend(loc=1) # legend 위치
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()

KOSPI 200 Candle Chart (2022)
```



데이터 탐색

01. EDA

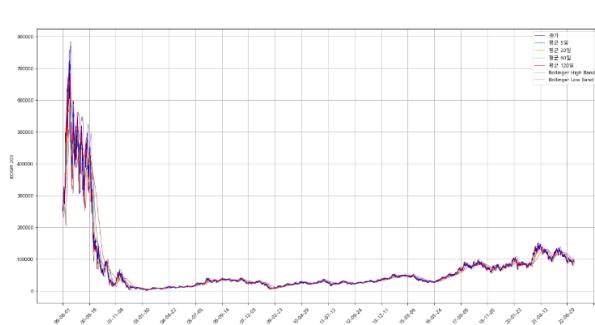
(4) [시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균 EDA – plot()

- 종목별 차트를 시각화하여 확인(전체 기간, 2022년)

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

In [16]: # 현재 데이터 시각화
fig, ax = plt.subplots(figsize=(20,10))
xdate = np.array(['2022-01-01', '2022-01-02'])
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:] # 2020-01-01 => 20-01-01

# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df['종가'], label='종가', linewidth=0.7,color='k')
ax.plot(xdate, df['종가'].rolling(window=5).mean(), label='평균 5일', linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=20).mean(), label='평균 20일', linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=60).mean(), label='평균 60일', linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=120).mean(), label='평균 120일', linewidth=0.7)
ax.plot(xdate, df['종가'], label='Bollinger High Band', linewidth=0.7)
ax.plot(xdate, df['종가'], label='Bollinger Low Band', linewidth=0.7)
candlestick2c_ohlc(ax,df['시가'],df['고가'],df['저가'],df['종가'], width=0.5, colorup='r', colordown='b')
fig.suptitle("KOSPI 200 Candle Chart (99.06.01 ~ 22.10.21)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보일 ticker 개수 ~20개이면 1줄
ax.legend(loc='upper right', frameon=False)
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()
```



```
# 2022년 데이터 시각화
df_2022 = df[df['일자']>='2022-01-01']
df_2022.reset_index(drop=True, inplace=True)
fig, ax = plt.subplots(figsize=(20,10))

# x축 날짜
xdate=df_2022['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:] # 2020-01-01 => 20-01-01

# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df_2022['종가'], label='종가', linewidth=0.7,color='k')
ax.plot(xdate, df_2022['종가'].rolling(window=5).mean(), label='평균 5일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=20).mean(), label='평균 20일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=60).mean(), label='평균 60일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=120).mean(), label='평균 120일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'], label='Bollinger High Band', linewidth=0.7)
ax.plot(xdate, df_2022['종가'], label='Bollinger Low Band', linewidth=0.7)
candlestick2c_ohlc(ax,df_2022['시가'],df_2022['고가'],df_2022['저가'],df_2022['종가'], width=0.5, colorup='r', colordown='b')

fig.suptitle("KOSPI 200 Candle Chart (2022)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보일 ticker 개수 ~20개이면 1줄
ax.legend(loc='upper right', frameon=False)
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()
```



데이터 탐색

01. EDA

(4) [시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균 EDA – plot()

- 종목별 차트를 시각화하여 확인(전체 기간, 2022년)

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

In [16]: # 전체 데이터 시각화
fig, ax = plt.subplots(figsize=(20,10))
# x축 날짜
xdate = df['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:6] # 2020-01-01 => 20-01-01
    # 5,20,60,120일 이동평균
    ax.plot(date, dff['종가'],label="종가", linewidth=0.7)
    ax.plot(date, dff['종가'].rolling(window=5).mean(), label="평균 5일", linewidth=0.7)
    ax.plot(date, dff['종가'].rolling(window=20).mean(), label="평균 20일", linewidth=0.7)
    ax.plot(date, dff['종가'].rolling(window=60).mean(), label="평균 60일", linewidth=0.7)
    ax.plot(date, dff['종가'].rolling(window=120).mean(), label="평균 120일", linewidth=0.7)
    ax.plot(date, df['B5'], label="Bollinger High Band", linewidth=0.7)
    ax.plot(date, df['B5'], label="Bollinger Low Band", linewidth=0.7)
    candlestick2_ohlc(ax,df['시가'],df['고가'],df['저가'],df['종가'], width=0.5, colorup='r', colordown='b')
    fig.suptitle("KOSPI 200 Candle Chart (09.06.01 ~ 22.10.21)")
    ax.set_xlabel("Date")
    ax.set_ylabel("KOSPI 200")
    ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보일 ticker 개수 ~20개이면 1달
    ax.legend(loc=1) # legend 위치
    plt.xticks(rotation = 45) # x-축 글씨 45도 회전
    plt.grid() # 그리드 표시
    plt.show()
```



```
# 2022년 데이터 시작화
In [17]: df_2022 = df[df['일자']>='2022-01-01']
df_2022.reset_index(inplace=True)

fig, ax = plt.subplots(figsize=(20,10))
# x축 날짜
xdate = df_2022['일자'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:6] # 2020-01-01 => 20-01-01
    # 5,20,60,120일 이동평균
    ax.plot(date, df_2022['종가'],label="종가", linewidth=0.7, color='k')
    ax.plot(date, df_2022['종가'].rolling(window=5).mean(), label="평균 5일", linewidth=0.7)
    ax.plot(date, df_2022['종가'].rolling(window=20).mean(), label="평균 20일", linewidth=0.7)
    ax.plot(date, df_2022['종가'].rolling(window=60).mean(), label="평균 60일", linewidth=0.7)
    ax.plot(date, df_2022['종가'].rolling(window=120).mean(), label="평균 120일", linewidth=0.7)
    ax.plot(date, df_2022['H5'], label="Bollinger High Band", linewidth=0.7)
    ax.plot(date, df_2022['L5'], label="Bollinger Low Band", linewidth=0.7)
    candlestick2_ohlc(ax,df_2022['시가'],df_2022['고가'],df_2022['저가'],df_2022['종가'], width=0.5, colorup='r', colordown='b')

fig.suptitle("KOSPI 200 Candle Chart (2022)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보일 ticker 개수 ~20개이면 1달
ax.legend(loc=1) # legend 위치
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()
```



데이터 탐색

01. EDA

(4) [시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균 EDA – plot()

- 종목별 차트를 시각화하여 확인(전체 기간, 2022년)

```
[시가, 고가, 저가, 종가, bollinger band] + 5,20,60,120일 이동평균 EDA
In [18]: # 원본 데이터로 시작화
fig, ax = plt.subplots(figsize=(20,10))
# x 축 날짜
xdate = df['날짜'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:] # 2020-01-01 => 20-01-01
# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df['종가'], label='종가', linewidth=0.7,color='k')
ax.plot(xdate, df['종가'].rolling(window=5).mean(), label='평균 5일', linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=20).mean(), label='평균 20일', linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=60).mean(), label='평균 60일', linewidth=0.7)
ax.plot(xdate, df['종가'].rolling(window=120).mean(), label='평균 120일', linewidth=0.7)
ax.plot(xdate, df['BHB'], label='Bollinger High Band', linewidth=0.7)
ax.plot(xdate, df['BBL'], label='Bollinger Low Band', linewidth=0.7)
candlestick_ohlc(ax,df[['시가','고가','저가','종가']], width=0.5, colorup='r', colordown='b')
plt.title("KOSPI 200 Candle Chart (99.06.01 ~ 22.10.21)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보일 ticker 개수 ->20개이면 1개
ax.legend(loc=1) # legend 위치
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()
```



```
# 2022년 데이터 시작화
In [17]: # 2022년 데이터 시작화
df_2022 = df[df['날짜'] >='2022-01-01']
df_2022.reset_index(drop=True, inplace=True)
fig, ax = plt.subplots(figsize=(20,10))
# x 축 날짜
xdate = df_2022['날짜'].astype('str')
for i in range(len(xdate)):
    xdate[i] = xdate[i][2:] # 2020-01-01 => 20-01-01
# 종가 및 5,20,60,120일 이동평균
ax.plot(xdate, df_2022['종가'], label='종가', linewidth=0.7,color='k')
ax.plot(xdate, df_2022['종가'].rolling(window=5).mean(), label='평균 5일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=20).mean(), label='평균 20일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=60).mean(), label='평균 60일', linewidth=0.7)
ax.plot(xdate, df_2022['종가'].rolling(window=120).mean(), label='평균 120일', linewidth=0.7)
ax.plot(xdate, df_2022['BHB'], label='Bollinger High Band', linewidth=0.7)
ax.plot(xdate, df_2022['BBL'], label='Bollinger Low Band', linewidth=0.7)
candlestick_ohlc(ax,df_2022[['시가','고가','저가','종가']], width=0.5, colorup='r', colordown='b')
fig.suptitle("KOSPI 200 Candle Chart (2022)")
ax.set_xlabel("Date")
ax.set_ylabel("KOSPI 200")
ax.xaxis.set_major_locator(ticker.MaxNLocator(25)) # x-축에 보일 ticker 개수 ->20개이면 1개
ax.legend(loc=1) # legend 위치
plt.xticks(rotation = 45) # x-축 글씨 45도 회전
plt.grid() # 그리드 표시
plt.show()
```

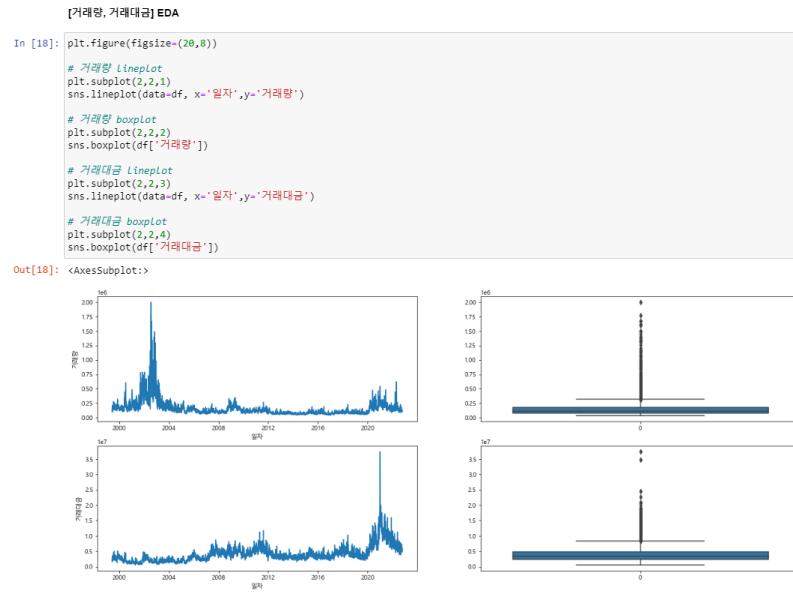


데이터 탐색

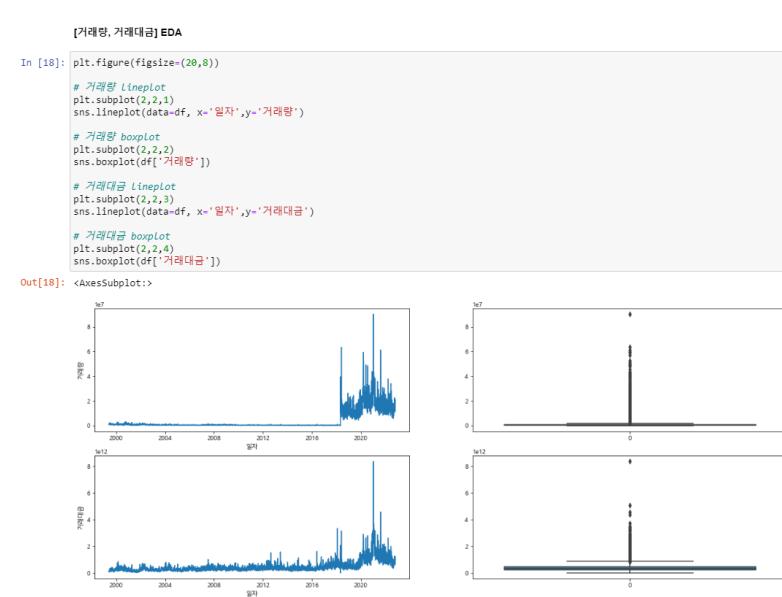
01. EDA

(5) [거래량, 거래대금] EDA – sns.lineplot(), sns.boxplot()

- [거래량, 거래대금] 모두 outlier가 존재함
- 코스피200, SK 하이닉스, 리노공업은 2000년대 초, 삼성전자는 2020년대, CJ ENM은 골고루 거래량이 활발함



▲ 코스피 200



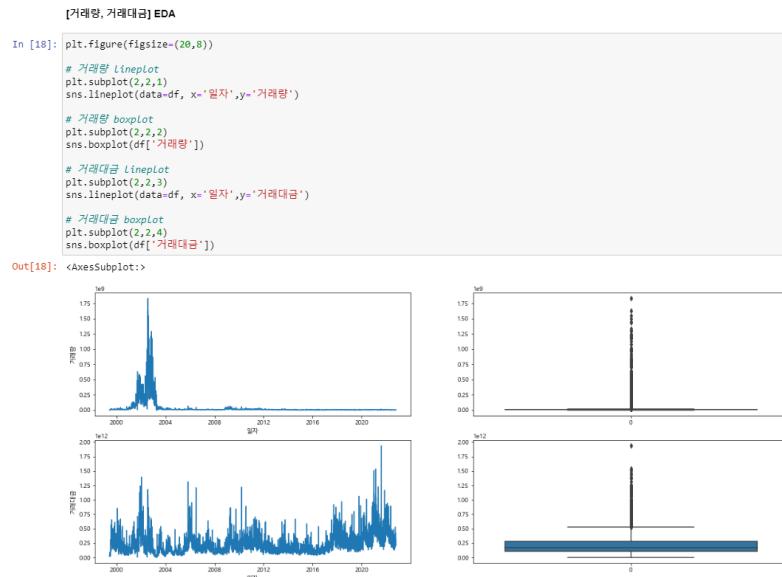
▲ 삼성전자

데이터 탐색

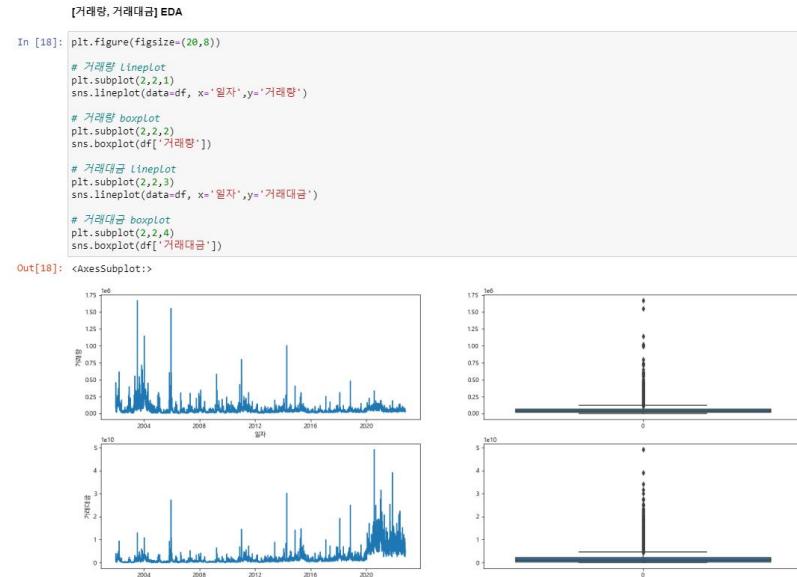
01. EDA

(5) [거래량, 거래대금] EDA – sns.lineplot(), sns.boxplot()

- [거래량, 거래대금] 모두 outlier가 존재함
- 코스피200, SK 하이닉스, 리노공업은 2000년대 초, 삼성전자는 2020년대, CJ ENM은 골고루 거래량이 활발함



▲ SK 하이닉스



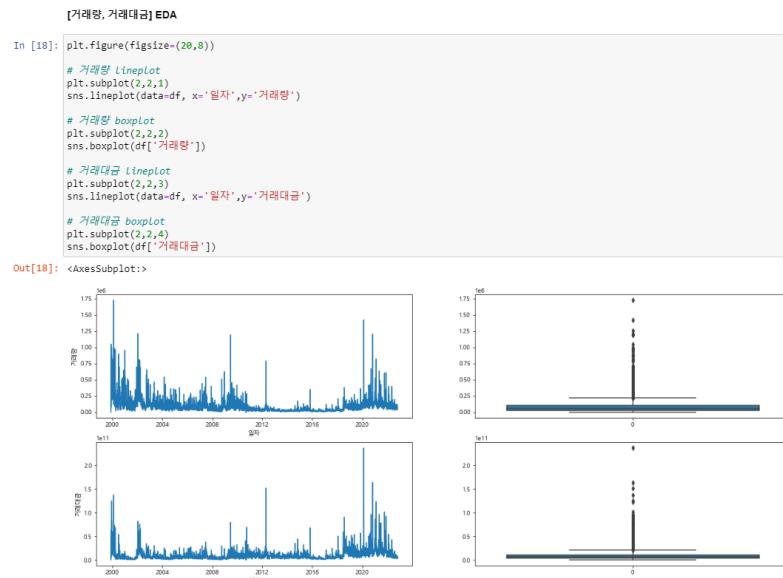
▲ 리노공업

데이터 탐색

01. EDA

(5) [거래량, 거래대금] EDA – sns.lineplot(), sns.boxplot()

- [거래량, 거래대금] 모두 outlier가 존재함
- 코스피200, SK 하이닉스, 리노공업은 2000년대 초, 삼성전자는 2020년대, CJ ENM은 골고루 거래량이 활발함



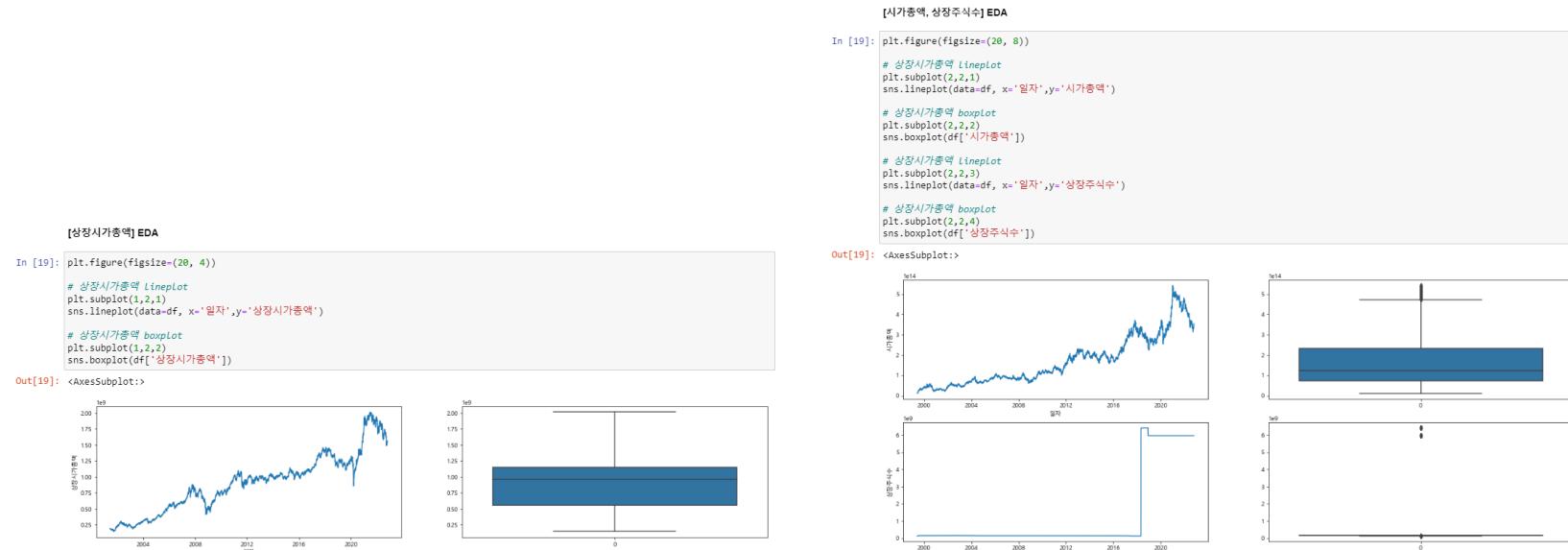
▲ CJ ENM

데이터 탐색

01. EDA

(6) [시가총액, 상장주식수] EDA – sns.lineplot(), sns.boxplot()

- 코스피 200의 경우 '상장시가총액'이라는 다른 변수 존재
- 시가총액은 대부분 우상향하며, 상장주식수는 계단형을 띠고 있음



▲ 코스피 200

▲ 삼성전자

데이터 탐색

01. EDA

(6) [시가총액, 상장주식수] EDA – sns.lineplot(), sns.boxplot()

- 코스피 200의 경우 '상장시가총액'이라는 다른 변수 존재
- 시가총액은 대부분 우상향하며, 상장주식수는 계단형을 띠고 있음

[시가총액, 상장주식수] EDA

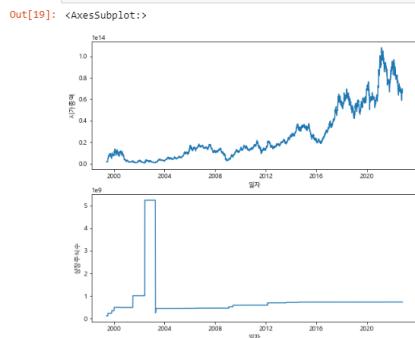
```
In [19]: plt.figure(figsize=(20, 8))

# 상장시가총액 LinePlot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='시가총액')

# 상장시가총액 boxplot
plt.subplot(2,2,2)
sns.boxplot(df['시가총액'])

# 상장시가총액 LinePlot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='상장주식수')

# 상장시가총액 boxplot
plt.subplot(2,2,4)
sns.boxplot(df['상장주식수'])
```



▲ SK 하이닉스

[시가총액, 상장주식수] EDA

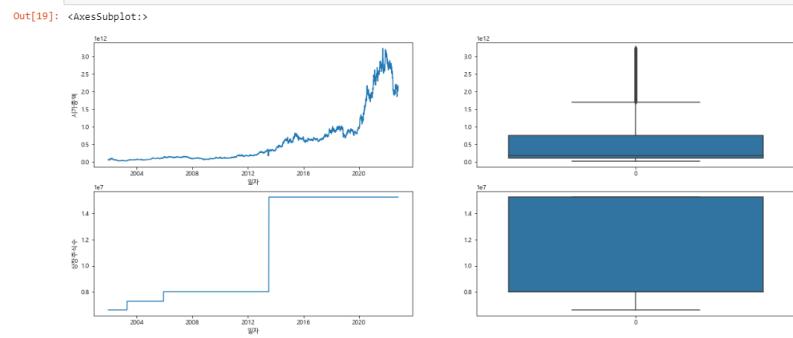
```
In [19]: plt.figure(figsize=(20, 8))

# 상장시가총액 LinePlot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='시가총액')

# 상장시가총액 boxplot
plt.subplot(2,2,2)
sns.boxplot(df['시가총액'])

# 상장시가총액 LinePlot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='상장주식수')

# 상장시가총액 boxplot
plt.subplot(2,2,4)
sns.boxplot(df['상장주식수'])
```



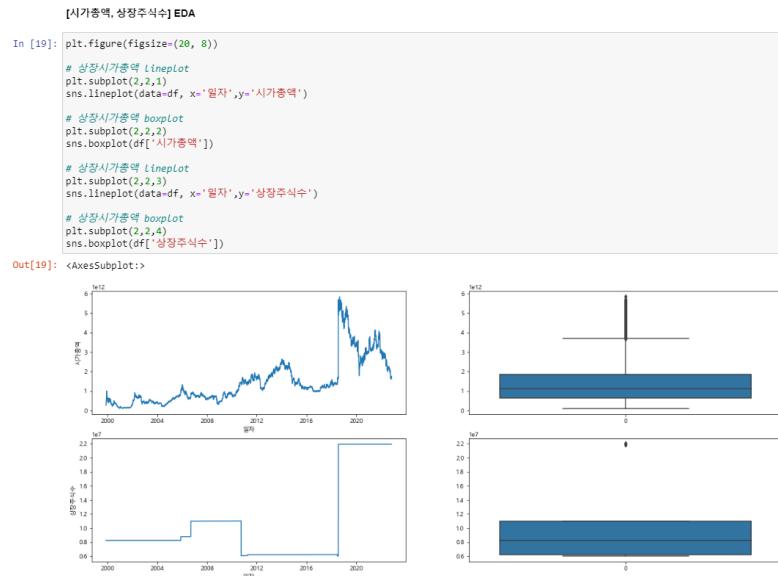
▲ 리노공업

데이터 탐색

01. EDA

(6) [시가총액, 상장주식수] EDA – `sns.lineplot()`, `sns.boxplot()`

- 코스피 200의 경우 '상장시가총액'이라는 다른 변수 존재
- 시가총액은 대부분 우상향하며, 상장주식수는 계단형을 띠고 있음



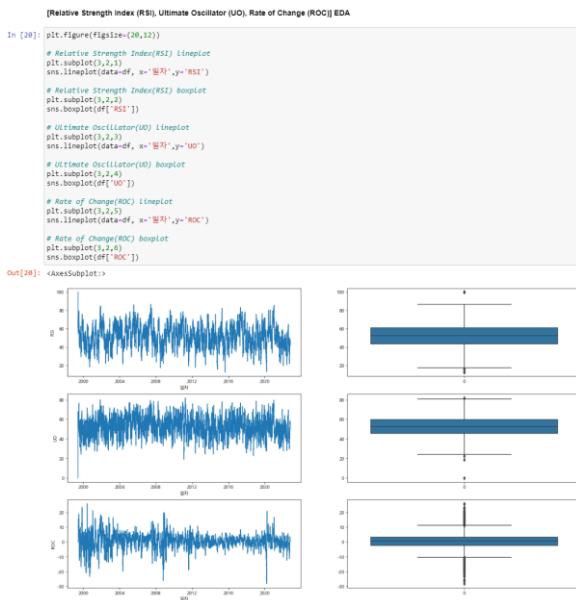
▲ CJ ENM

데이터 탐색

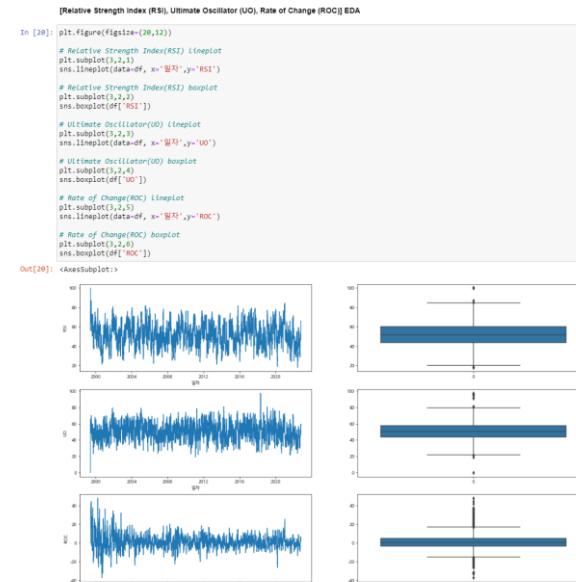
01. EDA

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

- 모든 종목에서 진동함수 형태를 띠고 있음
- RSI, UO와 달리 ROC는 outlier가 더 심함



▲ 코스피 200



▲ 삼성전자

데이터 탐색

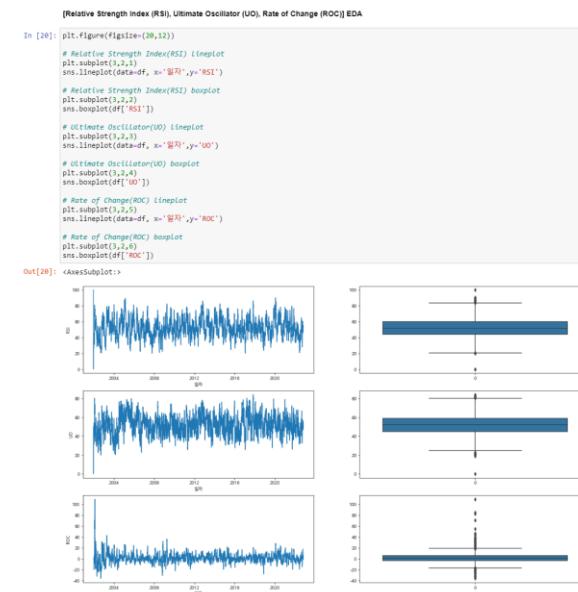
01. EDA

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

- 모든 종목에서 진동함수 형태를 띠고 있음
- RSI, UO와 달리 ROC는 outlier가 더 심함



▲ SK 하이닉스



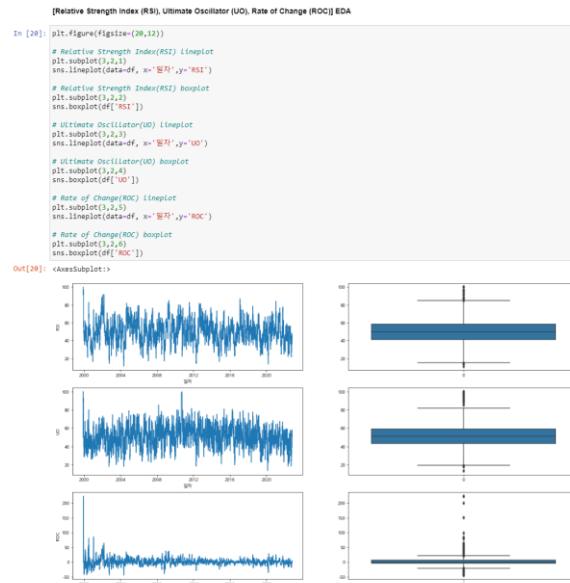
▲ 리노공업

데이터 탐색

01. EDA

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

- 모든 종목에서 진동함수 형태를 띠고 있음
- RSI, UO와 달리 ROC는 outlier가 더 심함



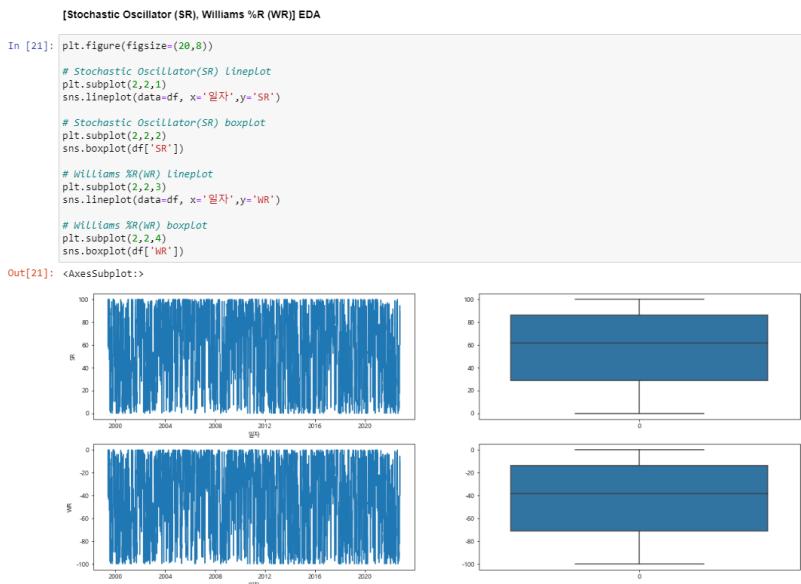
▲ CJ ENM

데이터 탐색

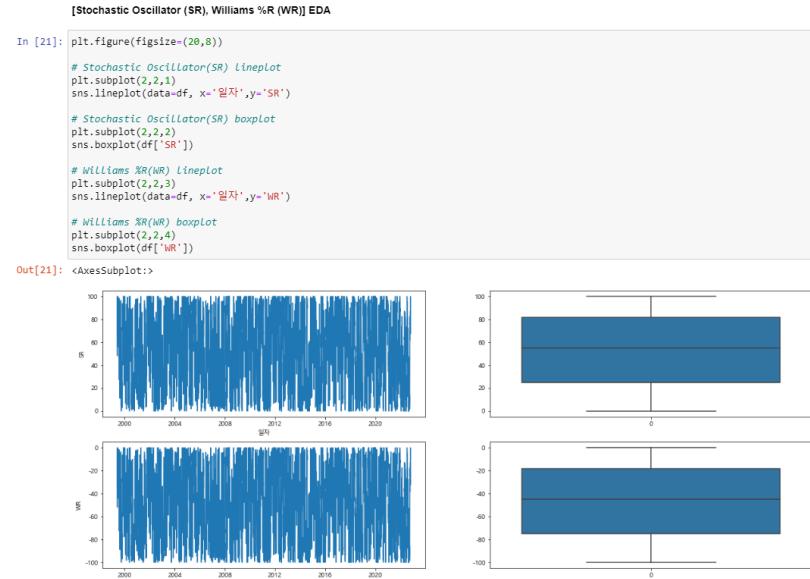
01. EDA

(8) [Stochastic Oscillator (SR), Williams %R (WR)] EDA – sns.lineplot(), sns.boxplot()

- 가장 진동성이 큰 지표로 확인되며, outlier가 존재하지 않음



▲ 코스피 200



▲ 삼성전자

데이터 탐색

01. EDA

(8) [Stochastic Oscillator (SR), Williams %R (WR)] EDA – sns.lineplot(), sns.boxplot()

- 가장 진동성이 큰 지표로 확인되며, outlier가 존재하지 않음

```
[Stochastic Oscillator (SR), Williams %R (WR)] EDA
In [21]: plt.figure(figsize=(20,8))

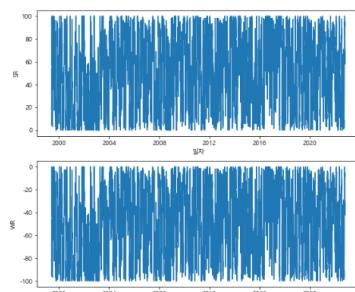
# Stochastic Oscillator(SR) Lineplot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='SR')

# Stochastic Oscillator(SR) boxplot
plt.subplot(2,2,2)
sns.boxplot(df['SR'])

# Williams %R(WR) Lineplot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='WR')

# Williams %R(WR) boxplot
plt.subplot(2,2,4)
sns.boxplot(df['WR'])

Out[21]: <AxesSubplot:>
```



▲ SK 하이닉스

```
[Stochastic Oscillator (SR), Williams %R (WR)] EDA
In [21]: plt.figure(figsize=(20,8))

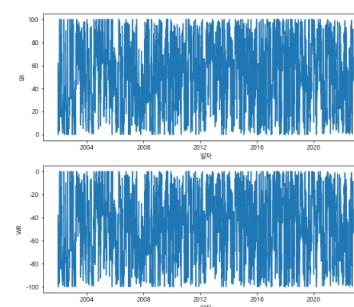
# Stochastic Oscillator(SR) Lineplot
plt.subplot(2,2,1)
sns.lineplot(data=df, x='일자',y='SR')

# Stochastic Oscillator(SR) boxplot
plt.subplot(2,2,2)
sns.boxplot(df['SR'])

# Williams %R(WR) Lineplot
plt.subplot(2,2,3)
sns.lineplot(data=df, x='일자',y='WR')

# Williams %R(WR) boxplot
plt.subplot(2,2,4)
sns.boxplot(df['WR'])

Out[21]: <AxesSubplot:>
```



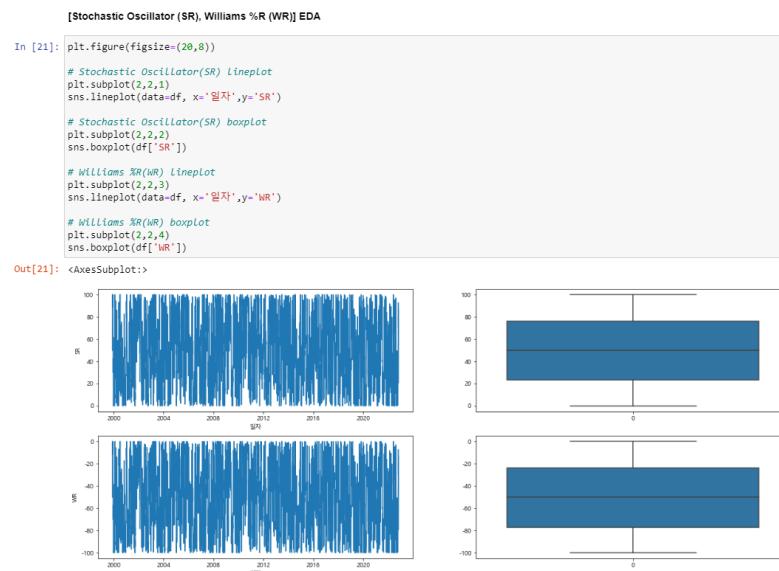
▲ 리노공업

데이터 탐색

01. EDA

(8) [Stochastic Oscillator (SR), Williams %R (WR)] EDA – sns.lineplot(), sns.boxplot()

- 가장 진동성이 큰 지표로 확인되며, outlier가 존재하지 않음



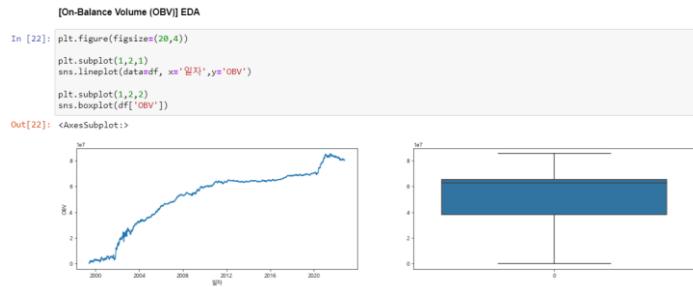
▲ CJ ENM

데이터 탐색

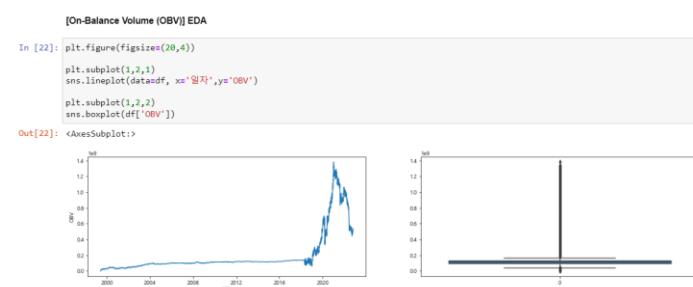
01. EDA

(9) [On-Balance Volume (OBV)] EDA – `sns.lineplot()`, `sns.boxplot()`

- 대부분 우상향의 형태를 띠며, boxplot으로 확인한 outlier 정도는 종목에 따라 상이함



▲ 코스피 200



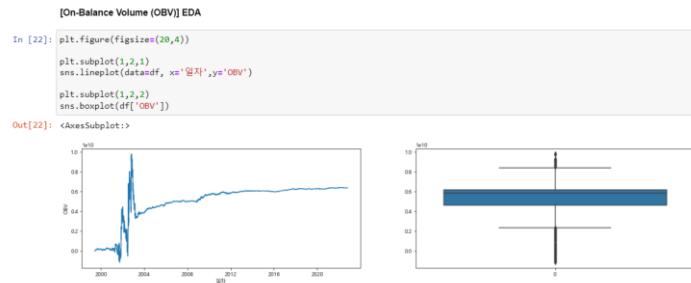
▲ 삼성전자

데이터 탐색

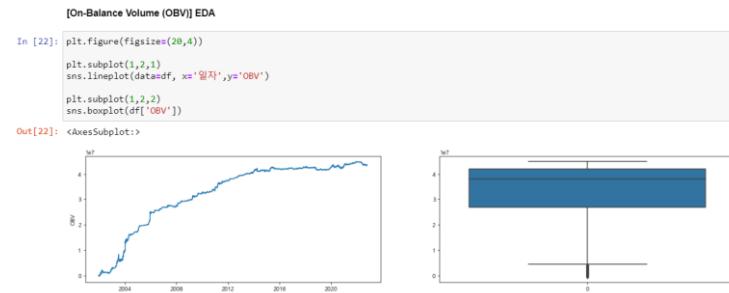
01. EDA

(9) [On-Balance Volume (OBV)] EDA – `sns.lineplot()`, `sns.boxplot()`

- 대부분 우상향의 형태를 띠며, boxplot으로 확인한 outlier 정도는 종목에 따라 상이함



▲ SK 하이닉스



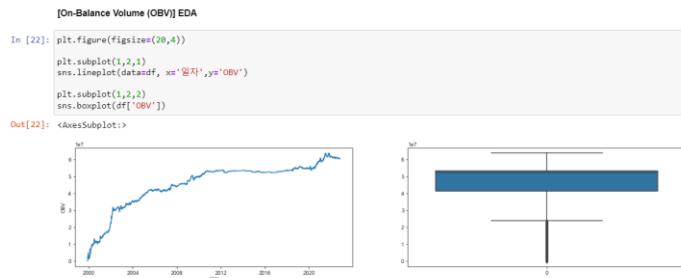
▲ 리노공업

데이터 탐색

01. EDA

(9) [On-Balance Volume (OBV)] EDA – `sns.lineplot()`, `sns.boxplot()`

- 대부분 우상향의 형태를 띠며, boxplot으로 확인한 outlier 정도는 종목에 따라 상이함



▲ CJ ENM

데이터 탐색

01. EDA

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

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



▲ 코스피 200



▲ 삼성전자

데이터 탐색

01. EDA

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

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



▲ SK 하이닉스



▲ 리노공업

데이터 탐색

01. EDA

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

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



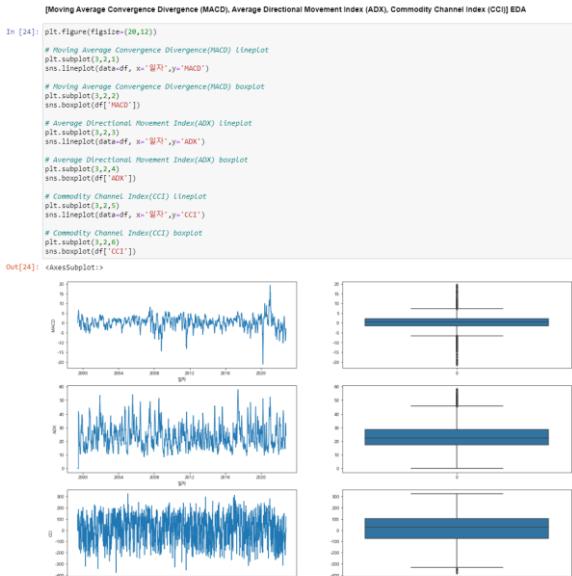
▲ CJ ENM

데이터 탐색

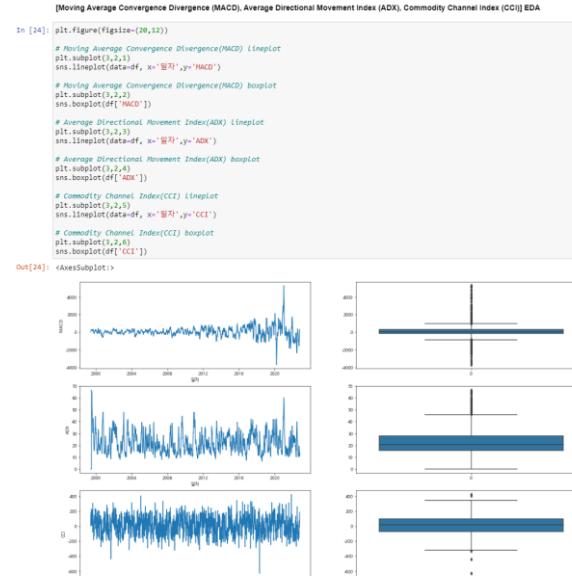
01. EDA

(11) [Moving Average Convergence Divergence (MACD), Average Directional Movement Index (ADX), Commodity Channel Index (CCI)] EDA – sns.lineplot(), sns.boxplot()

- MACD의 경우 삼성전자와 리노공업은 2020년대, SK 하이닉스는 2000년대에 진동성이 심해짐



▲ 코스피 200



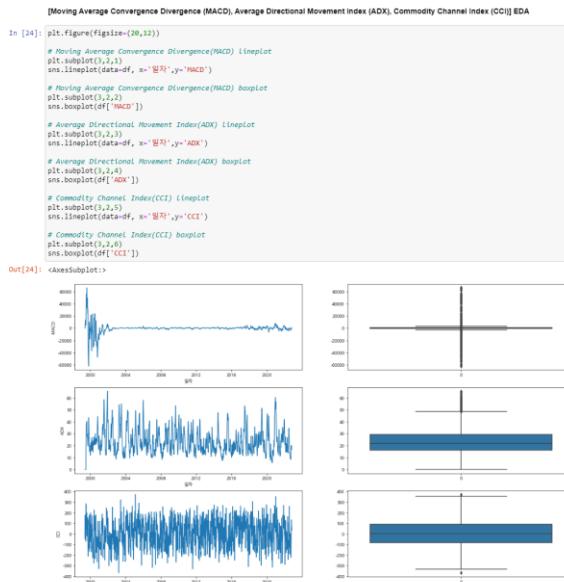
▲ 삼성전자

데이터 탐색

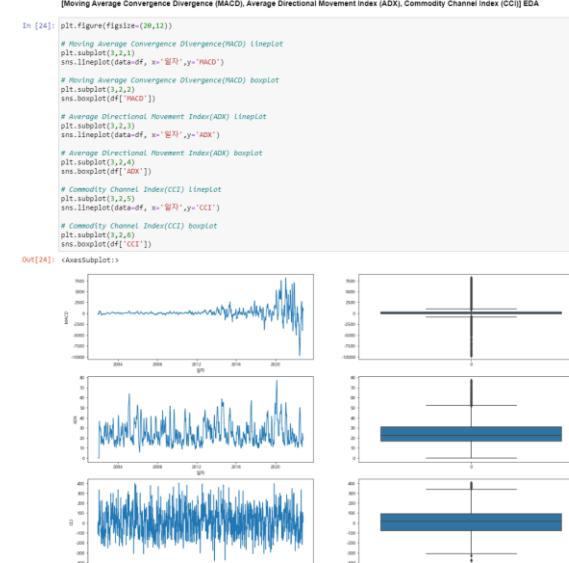
01. EDA

(11) [Moving Average Convergence Divergence (MACD), Average Directional Movement Index (ADX), Commodity Channel Index (CCI)] EDA – sns.lineplot(), sns.boxplot()

- MACD의 경우 삼성전자와 리노공업은 2020년대, SK 하이닉스는 2000년대에 진동성이 심해짐



▲ SK 하이닉스



▲ 리노공업

데이터 탐색

01. EDA

(11) [Moving Average Convergence Divergence (MACD), Average Directional Movement Index (ADX), Commodity Channel Index (CCI)] EDA – sns.lineplot(), sns.boxplot()

- MACD의 경우 삼성전자와 리노공업은 2020년대, SK 하이닉스는 2000년대에 진동성이 심해짐



▲ CJ ENM

데이터 탐색

01. EDA

(12) pairplot 생성 – sns.pairplot()

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



▲ 코스피 200

데이터 탐색

01. EDA

(12) pairplot 생성 – sns.pairplot()

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



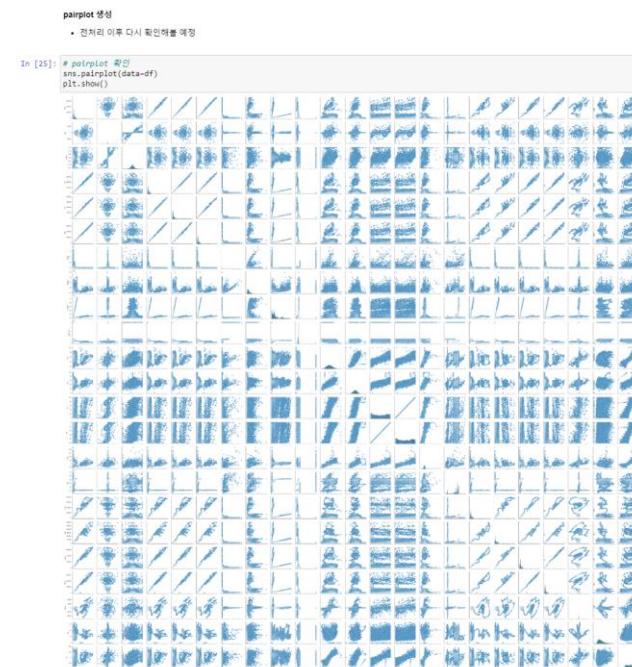
▲ 삼성전자

데이터 탐색

01. EDA

(12) pairplot 생성 – sns.pairplot()

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



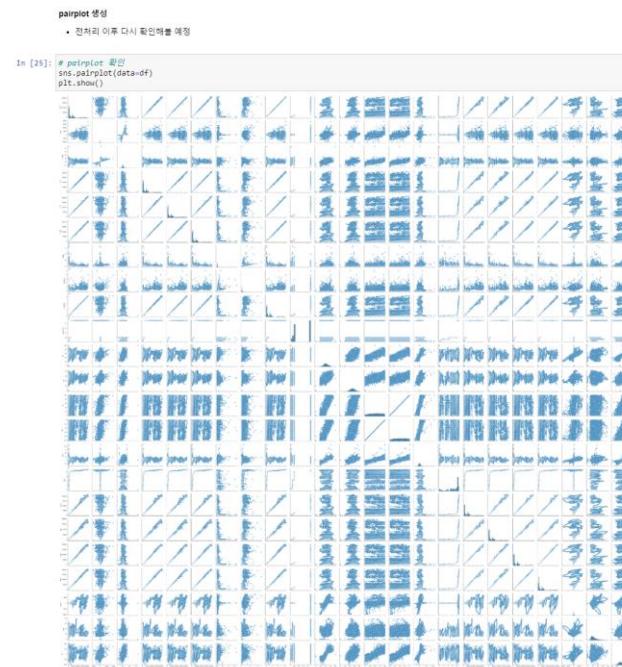
▲ SK 하이닉스

데이터 탐색

01. EDA

(12) pairplot 생성 – sns.pairplot()

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



▲ 리노공업

데이터 탐색

01. EDA

(12) pairplot 생성 – sns.pairplot()

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



▲ CJ ENM

데이터 탐색

02. 통계적 분석

(1) heatmap 생성 – sns.heatmap()

- 수치가 1.0이 나오는 경우가 존재하며, 데이터 전처리를 통한 변수 제거가 불가피함을 확인



▲ 코스피 200

데이터 탐색

02. 통계적 분석

(1) heatmap 생성 – sns.heatmap()

- 수치가 1.0이 나오는 경우가 존재하며, 데이터 전처리를 통한 변수 제거가 불가피함을 확인



▲ 삼성전자

데이터 탐색

02. 통계적 분석

(1) heatmap 생성 – sns.heatmap()

- 수치가 1.0이 나오는 경우가 존재하며, 데이터 전처리를 통한 변수 제거가 불가피함을 확인



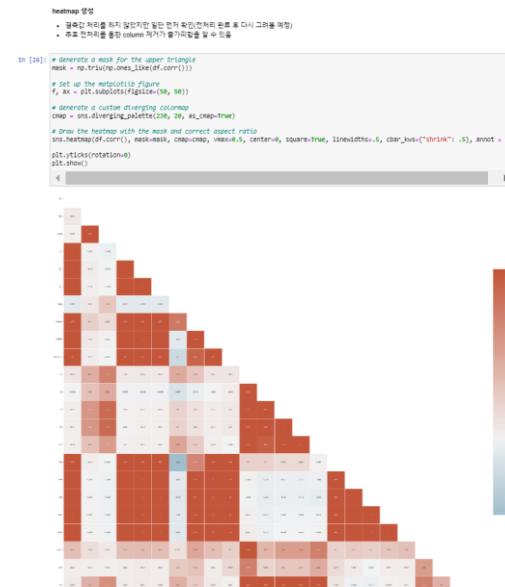
▲ SK 하이닉스

데이터 탐색

02. 통계적 분석

(1) heatmap 생성 – sns.heatmap()

- 수치가 1.0이 나오는 경우가 존재하며, 데이터 전처리를 통한 변수 제거가 불가피함을 확인



▲ 리노공업

데이터 탐색

02. 통계적 분석

(1) heatmap 생성 – sns.heatmap()

- 수치가 1.0이 나오는 경우가 존재하며, 데이터 전처리를 통한 변수 제거가 불가피함을 확인



▲ CJ ENM

데이터 탐색

02. 통계적 분석

(2) VIF, OLS 확인 – variance_inflation_factor(), sm.OLS()

- 10 이상이면 다중공선성으로 간주하게 되는데, 지나치게 큰 값들이 있는 걸 보아 변수 제거가 불가피함
- OLS의 p-value 또한 지나치게 높은 변수들이 존재

```
VIF 다중공선성 확인
• 중복값 처리를 하지 않았지만 일단 먼저 확인한거리 원로 후 다시 분석할 예정)
• 주로 전처리를 통한 column 제거가 불가피함을 알 수 있음

In [27]: from statsmodels.stats.outliers_influence import variance_inflation_factor
        import statsmodels.api as sm
        import statsmodels.formula.api as smf

# 일단 null인 column은 vif 확인을 위해 잠시 제거
X_value = df.drop(['일자','종가','상장시기증명'],axis=1)
y_value = df['종가']

In [28]: vif = pd.DataFrame()
        vif["Features"] = X_value.columns
        vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]

Out[29]:
   Features  VIF Factor
0   대비    0.10054
1   수익률    4.94089
2   시가    9879.34428
3   고가    10005.55470
4   저가    7257.52848
5   거래량    140954
6   거래대금    5.01098
7   RSI    11.68878
8   UO    3.46362
9   SR    145.55540
10  WR    65.87980
11  ROC    4.02143
12  OBV    5.82454
13  BHB    1745.83240
14  BLB    1545.61040
15  EMA    11735.78908
16  WMA    833.10782
17  MACD    7.82673
18  ADX    1.26900
19  CCI    10.20928
```

```
In [30]: X_value_2 = X_value.iloc[:,1:28]
In [31]: VIFX = sm.add_constant(X_value_2, has_constant="add")
        lin_model_VIF = sm.OLS(y_value, VIFX)
        lin_model_VIF = lin_model_VIF.fit()

In [32]: lin_model_VIF.summary()

Out[32]:
OLS Regression Results
Dep. Variable: 종가  R-squared:  1.000
Model: OLS  Adj. R-squared:  1.000
Method: Least Squares  F-statistic:  5.19e+05
Date: Wed, 23 Dec 2023  P-value (F-statistic):  0.00
Time: 10:51:14  Log-Likelihood: -1000.0
No. Observations: 5761  AIC: 1.204e+04
Df Residuals: 5760  BIC: 1.217e+04
Df Model: 10
Covariance Type: nonrobust

coef std err t P>|t| [0.025 0.975]
const -0.0005 2.6e-05 -19.712 0.000 -0.025 0.975
대비 0.1042 0.008 12.953 0.000 0.099 0.219
수익률 0.1184 0.010 11.943 0.010 -0.009 0.239
시가 9879.34428 0.010 -92.470 0.000 -0.244 0.266
고가 10005.55470 0.010 -95.437 0.000 0.406 0.556
저가 7257.52848 0.010 -82.454 0.000 0.521 0.556
거래량 140954 0.010 -12.765 0.000 0.219 0.236
거래대금 5.01098 0.001 -1.726 0.088 -1.726 0.088 1.39e-07
거래금액 11735.78908 7.63e-09 -1.219 0.320 -0.219 0.320
R^2 0.0172 0.003 4.721 0.000 0.007 0.018
UO 0.0342 0.002 20.918 0.000 0.031 0.038
SR -0.0179 0.002 -10.973 0.000 -0.021 -0.015
WR 0.0312 0.001 22.378 0.000 0.031 0.034
ROC 0.0312 0.001 22.378 0.000 0.031 0.034
OBV 1.733e-11 1.0e-09 0.073 0.945 -0.094 0.073 2.11e-05
BHB -0.0493 0.004 -11.834 0.000 -0.057 -0.041
BLB -0.0505 0.004 -12.410 0.000 -0.058 -0.042
EMA 0.2737 0.011 25.010 0.000 0.252 0.298
WMA -0.0005 0.003 -2.080 0.036 -0.012 -0.000
MACD -0.0001 0.008 -0.014 0.988 -0.018 0.018
ADX -0.0002 0.007 -0.232 0.918 -0.003 0.002
CCI -0.0046 0.001 -17.457 0.000 -0.005 0.000

Omnibus: 877.617  Durbin-Watson: 0.000
Prob(Durbin-Watson): 0.000 Jarque-Bera (JB): 11417.837
Skew: 0.039  Prob(JB): 0.000
Kurtosis: 3.833  Cond. No. 1.38e+23
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.02e-21. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

데이터 탐색

02. 통계적 분석

(2) VIF, OLS 확인 – variance_inflation_factor(), sm.OLS()

- 10 이상이면 다중공선성으로 간주하게 되는데, 지나치게 큰 값들이 있는 걸 보아 변수 제거가 불가피함
- OLS의 p-value 또한 지나치게 높은 변수들이 존재

```
VIF 다중공선성 확인
• 결합값 처리를 하지 않았지만 일단 먼저 확인(결처리 완료 후 다시 분석할 예정)
• 추후 결처리를 통한 column 개수가 불가피함을 알 수 있음

In [27]: from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.formula.api as smf

In [28]: X_value = df.drop(['일자','종가'],axis=1)
y_value = df['종가']

In [29]: vif = pd.DataFrame()
vif["Features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]

Out[29]:
   features      VIF Factor
0      대비    2.07997
1      수익률    2.00415
2      시가    11575.59000
3      고가    10456.92567
4      저가    11345.66930
5      거래량    10.29112
6      거래대금    11.23652
7      시가총액    208.26801
8      시장주식수    14.17620
9      RSI    11.26640
10     UO    3.06854
11     SR    33.08918
12     WR    22.43491
13     ROC    3.24722
14     OBV    7.71481
15     BHB    5552.38777
16     BLB    4470.50832
17     EMA    132524.88915
18     WMA    59551.89549
19     MACD    7.08283
20     ADX    1.30148
21     CCI    8.65043

In [30]: X_value2 = X_value.drop(['일자'],axis=1)

In [31]: VIFX = sm.add_constant(X_value2, has_constant='add')
lin_model_VIF = sm.OLS(y_value, VIFX)
lin_model_VIF = lin_model_VIF.fit()

In [32]: lin_model_VIF.summary()

Out[32]:
OLS Regression Results
Dep. Variable: 종가  R-squared:  1.000
Model: OLS  Adj. R-squared:  1.000
Method: Least Squares  F-statistic:  2.25e+26
Date: Wed, 20 Oct 2022  Prob (F-statistic):  0.00
Time: 17:45:20  Log-likelihood: -4168.0
No. Observations: 5702  AIC: 8.346e+04
Df Residuals: 5700  BIC: 8.346e+04
Df Model: 15
Covariance Type: nonrobust

coef std err t P>|t| [90.0% CI] [99.0% CI]
const -0.2021 0.015 -13.777 0.000 -0.227 -0.177
대비 0.0559 0.015 41.473 0.000 0.058 0.053
수익률 -0.0014 0.000 -4.308 0.000 -0.002 -0.001
시가 0.1944 0.022 8.732 0.000 0.150 0.209
고가 0.0487 0.022 2.213 0.032 0.024 0.070
저가 -0.0497 0.022 -2.233 0.032 -0.062 -0.034
거래량 -1.0204 2.146e-05 -4.653 0.000 -1.06e-05 -5.0e-06
거래대금 1.25e-10 1.6e-11 3.814 0.000 1.7e-11 1.6e-10
시가총액 1.37e-11 1.e-11 22.479 0.000 1.25e-11 1.4e-11
장당주식수 0.93e-09 0.9e-09 0.265 0.000 0.7e-09 1.0e-07

R-sq: 12.4581 1.127 11.125 0.000 10.270 14.000
Adj R-sq: 3.5751 0.723 2.407 0.000 2.145 5.000
SR: 0.0000 0.000 0.000 0.000 0.000 0.000
RM: 2.12e-05 0.795 2.651 0.000 0.000 0.000
WC: 2.12e-05 0.795 2.651 0.000 0.000 0.000
ROC: -11.6878 1.985 -5.983 0.000 -13.731 4.654
OBV: 7.32e-10 4.39e-09 -16.211 0.000 4.4e-07 2.24e-07
EWR: -0.2405 0.015 -15.383 0.000 -0.298 -0.211
BLB: -0.2787 0.015 -16.031 0.000 -0.307 -0.250
EMA: 0.0001 0.015 1.027 0.000 0.042 1.108
SMMA: 0.0001 0.015 1.027 0.000 0.042 1.108
MACD: -0.1281 0.019 6.603 0.000 -0.164 -0.085
QStatistics: 2062.817 Durbin-Watson: 5.946
Prob(Durbin-Watson): 0.000 Jarque-Bera (JB): 7323.738
Skew: 1.099 Prob(JB): 0.000
Kurtosis: 22.357 Cond. No: 1.17e+00
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.02e-26. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

데이터 탐색

02. 통계적 분석

(2) VIF, OLS 확인 – variance_inflation_factor(), sm.OLS()

- 10 이상이면 다중공선성으로 간주하게 되는데, 지나치게 큰 값들이 있는 걸 보아 변수 제거가 불가피함
- OLS의 p-value 또한 지나치게 높은 변수들이 존재

```
In [27]: from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.formula.api as smf

In [28]: X_value = df.drop(['일자', '종가'], axis=1)
y_value = df['종가']

In [29]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]

Out[29]:
   features  VIF Factor
0      대비    1.93241
1     수익률    1.64761
2      시가  2350.48700
3      고가  1870.13931
4      저가  1940.10973
5     거래량    2.88594
6    거래대금    1.88548
7    시가총액    3.20511
8  상장주식수    2.33475
9      RSI    0.02639
10     UO    2.83653
11      SR    10.81640
12      WR    12.80004
13      ROC    2.55471
14      OBV    3.42929
15      BHB    450.45705
16      BLB    220.52061
17      EMA    2494.32119
18      WMA    123.53634
19      MACD    3.83413
20      ADX    1.23044
21      CCI    7.70963
```

```
In [30]: X_value_1 = X_value.iloc[:,1:28]
X_value_1 = sm.add_constant(X_value_1, has_constant=True)
lin_model = sm.OLS(y_value, X_value_1).fit()
lin_model.vif
```

```
In [31]: lin_model.vif.summary()
```

```
OLS Regression Results
Dep. Variable: X_value_1 = X_value.iloc[:,1:28]
Model: OLS
Method: Least Squares
F-statistic: 1.374e+05
Date: Wed, 20 Oct 2022
Time: 17:47:26
Log-Likelihood: -45757
No. Observations: 8781
AIC: 1.013e+05
DF Residuals: 8762
BIC: 1.013e+05
DF Model: 18
Covariance Type: nonrobust

```

	coef	std err	t	P> t	OR 95%	OR 90%
const	-0.2204	2.348	-0.933	0.388	0.004	0.004
B1	0.3011	0.004	85.490	0.000	0.371	0.388
수익률	0.0032	0.001	2.819	0.008	0.001	0.008
시가	-0.1115	0.010	-11.718	0.000	-0.132	-0.093
고가	0.4594	0.009	51.487	0.000	0.441	0.472
저가	0.1188	0.008	61.394	0.000	0.100	0.033
거래량	-2.417e-07	2.263e-07	-1.039	0.425	-0.18e-07	0.20e-07
거래대금	-1.93e-15	1.57e-15	-1.237	0.219	-4.01e-15	1.19e-15
시가총액	4.02e-13	1.44e-12	2.772	0.003	1.18e-12	1.24e-12
상장주식수	5.19e-08	3.46e-08	1.478	0.139	-1.06e-08	1.24e-07
R1	-22.2508	1.509	-14.667	0.000	27.252	-11.288
UO	-0.0001	0.000	-0.000	1.000	0.000	0.000
WR	0.7387	1.091	0.698	0.470	0.149	0.149
ROC	3.4343	1.078	3.130	0.007	-0.244	7.715
OBV	0.3508	2.142	0.164	0.871	-3.348	4.548
BHB	-0.0402	0.204	-12.089	0.000	-0.053	-0.039
BLB	-0.0392	0.204	-18.750	0.000	-0.044	-0.044
EMA	2.2347	0.215	10.479	0.000	2.117	2.058
WMA	-0.0032	0.002	-1.400	0.153	-0.008	0.001
MACD	0.0430	0.008	7.882	0.000	0.000	0.054

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.4e-25. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

데이터 탐색

02. 통계적 분석

(2) VIF, OLS 확인 – variance_inflation_factor(), sm.OLS()

- 10 이상이면 다중공선성으로 간주하게 되는데, 지나치게 큰 값들이 있는 걸 보아 변수 제거가 불가피함
- OLS의 p-value 또한 지나치게 높은 변수들이 존재

```
In [27]: from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.formula.api as smf

In [28]: X_value = df.drop(['일자','종가'],axis=1)
y_value = df['종가']

In [29]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]
vif
```

```
Out[29]:
   features    VIF Factor
0      CBI  3.57209
1      금값  1.74575
2      시가  8861.12060
3      고가  8700.81963
4      저가  8150.19805
5      거래량  2.12096
6      거래대금  6.33848
7      시가율  4888.47859
8      상장주식수  3.48460
9       RSI  7.18338
10      UO  1.02478
11       SR  15.99105
12       WR  10.98774
13       ROC  3.05985
14       OBV  2.30498
15       BHB  5324.71324
16       BLB  4090.67442
17       EMA  117552.38454
18       WMA  46208.59771
19       MACD  4.75442
20       ADX  1.39521
21       CCI  8.10988
```

```
In [30]: X_value_1 = X_value.iloc[:,1:38]
In [31]: VIF_X = sm.add_constant(X_value_1, has_constant='add')
lin_model1_VIF = sm.OLS(y_value, VIF_X).fit()
lin_model1_VIF.summary()

In [32]: lin_model1_VIF.summary()
Out[32]:
OLS Regression Results
Dep. Variable: 종가  R-squared:  0.000
Model: OLS  Adj. R-squared:  0.000
Method: Least Squares  F-statistic:  0.01e+00
Date: Wed, 26 Oct 2016  Prob (F-statistic):  0.00
Time: 19:10:37  Log-Likelihood: -1.0721
No. Observations: 6115  AIC: 4.145e+04
Df Residuals: 6113  BIC: 4.743e+04
Df Model: 18
Covariance Type: nonrobust
            coef  std err      t  P>|t|    [0.025  0.975]
常数项  -0.0459  0.010  -3.958  0.000  -0.081  -0.020
CBI  0.3832  0.008  81.388  0.000  0.371  0.398
수익률 -885.7115  227.854  -3.913  0.000  -115.405  -422.040
시가  0.1098  0.008  12.543  0.000  0.063  0.157
고가  0.2115  0.008  25.916  0.000  0.154  0.258
저가  0.2115  0.008  25.916  0.000  0.154  0.258
거래량  8.0000  8.78e-05  2.703  0.000  7.25e-05  1.64e-04
거래대금 -1374.92  3.42e+00  -3.959  0.000  -2.21e+00  -6.95e-01
시가율  1.87e+00  4.16e-01  44.845  0.000  1.6e+00  2.0e+00
상장주식수 -7.40e+00  2.22e+00  -3.381  0.001  -1.16e+00  -3.14e+00
BLB  -1.3524  0.384  -3.447  0.001  -2.707  0.802
UO  5.7877  1.003  5.886  0.000  4.856  6.850
SR  -1.0308  0.012  -85.945  0.000  -2.278  0.199
AR  3.0000  0.045  65.556  0.000  0.000  3.000
MACD  0.2715  0.008  33.818  0.000  4.200  4.449
OBV  1.05e+00  8.04e-01  2.145  0.032  1.1e-01  2.47e-05
BHB  -0.0419  0.008  -6.543  0.000  -0.007  -0.039
BLB  -0.0428  0.008  -6.628  0.000  -0.008  -0.030
EMA  0.1620  0.032  6.957  0.000  0.103  0.229
WMA  0.0307  0.021  1.477  0.140  -0.005  0.072
MACD  0.0191  0.007  3.028  0.022  0.000  0.028
```

```
Omnibus: 334.345 Durbin-Watson: 1.129
Prob(Omnibus): 0.000 Jarque-Bera (JB): 7145.07
Skew: -0.234 Prob(JB): 0.00
Kurtosis: 47.015 Cond. No. 2.25e+27
```

▲ 리노공업

데이터 탐색

02. 통계적 분석

(2) VIF, OLS 확인 – variance_inflation_factor(), sm.OLS()

- 10 이상이면 다중공선성으로 간주하게 되는데, 지나치게 큰 값들이 있는 걸 보아 변수 제거가 불가피함
- OLS의 p-value 또한 지나치게 높은 변수들이 존재

```
In [27]: from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.formula.api as smf

In [28]: X_value = df.drop(['일자','종가'],axis=1)
y_value = df['종가']

In [29]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]
vif

Out[29]:
   features    VIF Factor
0      OI  5.85171
1     수출  4.74411
2     시가  2351.2021
3     고가  1548.4358
4     저가  1366.77912
5     거래량  5.85374
6     거래고금  5.95155
7     시가율  13.01811
8     상장주식  13.06389
9      RSI  13.98868
10     UO  3.10165
11      SR  34.85202
12      WR  24.00422
13      ROC  3.18587
14      OBV  3.73580
15      EHB  403.18009
16      BLB  455.37380
17      EMA  3988.93001
18      WMA  1178.04684
19      MACD  7.27018
20      ADX  1.41389
21      CCI  6.70909
```

```
In [30]: X_value_1 = X_value.iloc[:,1:28]

In [31]: VIF = sm.OLS(endog=y_value, exog=X_value_1).fit().get_vif()
ols_model_VIF = sm.OLS(endog=y_value, exog=X_value_1).fit()

In [32]: ols_model_VIF.summary()

Out[32]:
OLS Regression Results
Dep. Variable: 종가  R-squared:  0.000
Model: OLS  Adj. R-squared:  0.000
Method: Least Squares  F-statistic:  2.454e+05
Date: Wed, 29 Oct 2014  Prob (F-statistic):  0.00
Time: 10:25:21  Log-Likelihood: -552.24
No. Observations: 6000  AIC: 1.01e+05
Df Residuals: 5999  BIC: 1.005e+05
Df Model: 10
Covariance Type: nonrobust
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-2.4723	0.120	-20.600	<0.000	-2.707	-2.238
OI	0.7138	0.021	34.061	0.000	0.673	0.755
수출	2.346e-04	2780.036	-0.042	0.964	-0.24e-04	2.44e-04
시가	0.1278	0.023	5.577	0.000	0.084	0.172
고가	0.1278	0.023	5.577	0.000	0.084	0.172
저가	-0.2642	0.016	-1.630	0.109	-0.379	0.000
거래량	0.0208	0.001	3.091	0.001	0.001	0.004
거래고금	2.39e-05	8.73e-05	-0.049	0.993	-0.39e-05	0.00e+00
시가율	1.05e-10	1.27e-10	-0.100	0.999	-0.19e-10	0.00e+00
상장주식	2.95e-05	2.33e-05	0.875	0.382	0.24e-05	0.52e-05
EHB	175.4717	8.911	19.450	0.000	165.933	194.124
UO	87.4597	5.551	12.154	0.000	55.570	70.341
SR	-110.4483	7.082	-15.840	0.000	-142.392	-69.808
WR	126.7538	5.598	22.480	0.000	107.717	145.780
ROC	0.471	0.471	1.010	0.312	-0.472	0.772
OBV	2.88e-07	5.3e-07	-0.554	0.973	-0.34e-05	7.47e-06
BIN	-0.2049	0.010	-2.008	0.020	-0.224	-0.185
BLB	-0.2147	0.010	-20.070	0.000	-0.238	-0.195
EMI	1.3747	0.030	49.319	0.000	1.317	1.433
WMA	-0.0217	0.010	-1.388	0.172	-0.053	0.009
MACD	-0.1044	0.010	-4.427	0.000	-0.142	-0.087

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.96e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

전처리 방법

01. Data Reducing

(1) 사용 데이터 재설정

- 파생변수 생성으로 인해 결측치 생성 가능성 존재 → 2000년부터의 데이터 사용하는 것으로 통일

(2) VIF를 활용한 다중공선성 제거

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

```
from statsmodels.stats.outliers_influence import variance_inflation_factor  
import statsmodels.api as sm  
import statsmodels.formula.api as smf
```

(3) OLS를 이용한 유의미한 변수 파악

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

전처리 방법

02. 최종 데이터 생성 및 재시각화

(1) heatmap, pairplot

- heatmap과 pairplot을 통해 최종 데이터의 모습 재시각화

(2) ML 모델 사용을 위한 데이터 변경(DL은 그대로 유지)

- ML 회귀 분석을 위해서는 독립변수 값들이 존재해야 종속변수 값을 추론할 수 있음
- 기존 데이터로는 3일 치의 독립변수를 알 수 없다는 한계 인식
 - 종가를 3일치 위로 Shift 하는 작업을 수행
 - 종가_y 생성

In [53]: df3.tail()

Out[53]:

	일자	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX	종가_y
5624	2022-10-17	289.57000	0.21000	0.00070	111597.00000	4981197.00000	52.86745	-0.21709	80366684.00000	32.20128	288.61000
5625	2022-10-18	293.59000	4.02000	0.01390	117118.00000	5739328.00000	57.88878	3.67245	80483802.00000	31.04973	288.85000
5626	2022-10-19	291.29000	-2.30000	-0.00780	110851.00000	5688842.00000	53.23347	3.05678	80372951.00000	29.66979	NaN
5627	2022-10-20	288.61000	-2.68000	-0.00220	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262	NaN
5628	2022-10-21	288.85000	0.24000	0.00080	72821.00000	3358988.00000	55.13946	0.20120	80308868.00000	28.37462	NaN

모델 소개

01. 모델 비교 및 선택

(1) 모델 비교 방안

- 코스피 200 데이터를 기준으로 모델 성능 비교 후 모델을 선택
- 초기에 train, val, test를 5:3:2로 시간 순서대로 분리하였으나 test에 대한 성능이 상당히 떨어지는 것을 확인
- 최근의 주가 흐름을 더 잘 반영시키기 위해서는 최근 데이터 또한 훈련에 많이 포함되어야 한다고 생각
→ **train, val, test set 세 가지로 분리하는 것보다 train과 test 만으로 분리해 훈련 및 테스트 진행**
- 00.01.04 ~ 22.10.21(오전)까지의 데이터를 사용해 ML, DL 모델을 비교
- **최종적으로 22.10.20~22.10.21 기간의 실제 예측에 대한 MSE 비교를 통해 ML과 DL 중 하나의 방식 선택**

(2) 최종 예측 방식

- 수익률을 바로 예측하는 모델을 시도해보았으나 지나친 진동함수를 예측하는 것은 어려움을 깨달음
→ 27, 28, 31일의 종가를 예측한 후 이를 통해 수익률을 계산하는 방식을 사용할 예정

모델 소개

01. 모델 비교 및 선택

(3) ML 모델 concept

- 전처리 완료된 데이터의 column들 모두 사용
- ML 모델은 여러 X 변수들을 통해 Y 변수를 예측하는 다중회귀분석 모델
- X : 대비, 수익률, 거래량, 거래대금, UO, ROC, OBV, ADX
- Y : 종가_y (전처리 과정에서 진행한 3일 위로 shift한 종가 데이터)

In [53]: df3.tail()

Out[53]:

	일자	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX	종가_y
5624	2022-10-17	289.57000	0.21000	0.00070	111597.00000	4981197.00000	52.86745	-0.21709	80366684.00000	32.20128	288.61000
5625	2022-10-18	293.59000	4.02000	0.01390	117118.00000	5739328.00000	57.88878	3.67245	80483802.00000	31.04973	288.85000
5626	2022-10-19	291.29000	-2.30000	-0.00780	110851.00000	5688842.00000	53.23347	3.05678	80372951.00000	29.66979	NaN
5627	2022-10-20	288.61000	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262	NaN
5628	2022-10-21	288.85000	0.24000	0.00080	72821.00000	3358988.00000	55.13946	0.20120	80308868.00000	28.37462	NaN

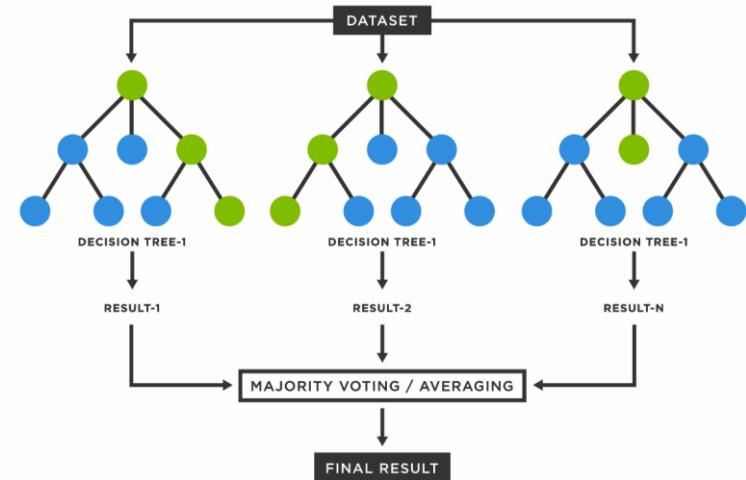
모델 소개

01. 모델 비교 및 선택

(3) ML 모델 : RandomForest

- 기존 Bagging의 이점을 살리고 변수를 랜덤으로 선택하는 과정을 추가함으로써 나무들의 상관성을 줄여 예측력을 향상한 앙상블 모형
- Regression, Classification 모두 지원
- 장점 : 복잡한 관계 모델링 가능, 예측력이 좋으며 이상치에 강함, scaling 및 정규화 과정이 필요 없음
- 단점 : 해석의 어려움, 유연성이 떨어짐, 계산량이 많고 소요되는 시간이 길

```
from sklearn.ensemble import RandomForestRegressor
```



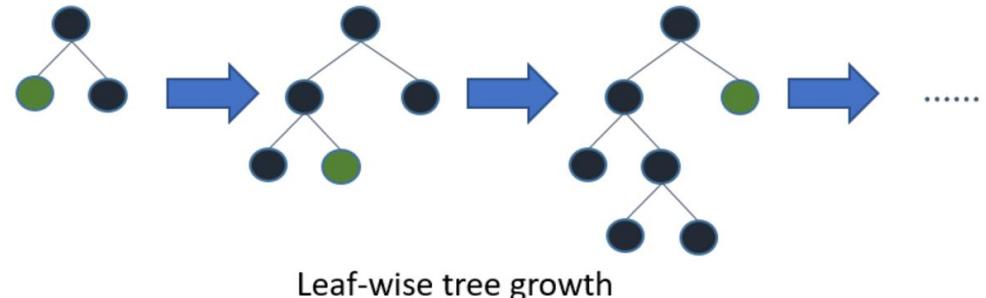
모델 소개

01. 모델 비교 및 선택

(3) ML 모델 : LightGBM

- Gradient Boosting 프레임워크로 트리 기반 학습 알고리즘
- 기존 GBM과 다른점은 GBM은 균형 트리분할(Level Wise) 방식, LightGBM은 리프중심 트리분할(Leaf Wise) 방식
- Regression, Classification 문제를 모두 지원
- 장점 : 학습하는데 걸리는 시간이 적음, 메모리 사용량이 상대적으로 적은 편
- 단점 : 작은 데이터셋을 사용할 경우 과적합 가능성이 큼

```
from lightgbm import LGBMRegressor
```



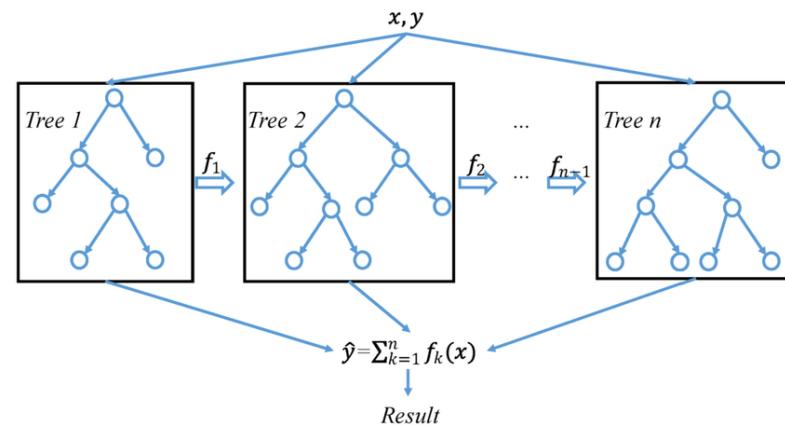
모델 소개

01. 모델 비교 및 선택

(3) ML 모델 : XGBoost

- Extreme Gradient Boosting의 약자로 Gradient Boost를 병렬 학습이 지원되도록 구현한 라이브러리
- Regression, Classification 문제를 모두 지원
- 장점 : GBM 대비 빠른 수행시간, 과적합 규제, 뛰어난 예측 성능 발휘
- 단점 : 작은 데이터에 대해 과적합 가능성이 있음, 해석의 어려움

```
from xgboost import XGBRegressor
```



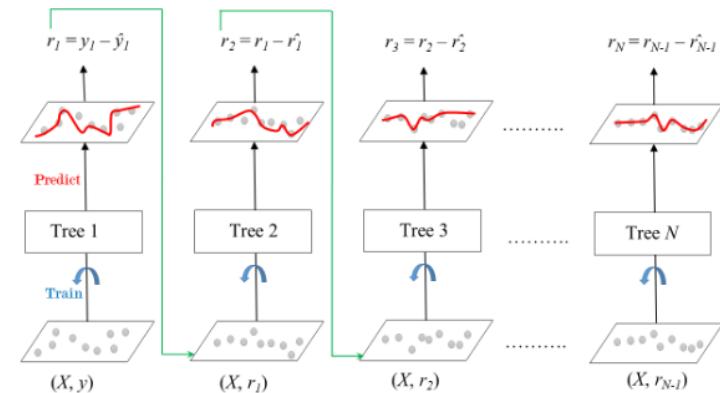
모델 소개

01. 모델 비교 및 선택

(3) ML 모델 : GradientBoost

- Gradient Boosting는 Gradient를 이용하여 이전 모형의 약점을 보완하는 새로운 모형을 순차적으로 적합한 뒤 이들을 선형 결합하여 얻어진 모형을 생성하는 지도 학습 알고리즘
- Regression, Classification 문제를 모두 지원
- 장점 : 구현이 쉽고, 정확도가 좋으며, 유연성이 큼
- 단점 : 과적합 발생의 위험, 메모리 문제, 해석의 어려움

```
from sklearn.ensemble import GradientBoostingRegressor
```



모델 소개

01. 모델 비교 및 선택

(4) 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]

In [159]: df3.head(10)

Out[159]:

	일자	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX
0	2000-01-04	133.66000	3.64000	0.02800	121258.00000	3067398.00000	54.21263	6.36639	3522692.00000	26.50646
1	2000-01-05	123.86000	-9.80000	-0.07330	153265.00000	3921476.00000	45.75913	0.16983	3369427.00000	26.10429
2	2000-01-06	120. 3 0000	-3.06000	-0.02470	116958.00000	2988051.00000	39.25693	1.77774	3252469.00000	24.80416
3	2000-01-07	119.10000	-1.70000	-0.01410	130572.00000	3009494.00000	40.28562	1.85581	3121897.00000	23.17014
4	2000-01-10	124.11000	5.01000	0.04210	144169.00000	3028380.00000	44.31994	5.72451	3266066.00000	22.01308
5	2000-01-11	123. Y 1000	-1.00000	-0.00810	143686.00000	3191206.00000	41.73704	2.47212	3122380.00000	21.25887
6	2000-01-12	119.81000	-3.30000	-0.02680	121366.00000	2353350.00000	37.96667	0.11699	3001014.00000	19.81714
7	2000-01-13	119.22000	-0.59000	-0.00490	133203.00000	2635958.00000	33.99939	-1.54431	2867811.00000	18.75312
8	2000-01-14	118.95000	-0.27000	-0.00230	129885.00000	2580768.00000	38.70961	-2.74712	2737926.00000	17.45339
9	2000-01-17	123.85000	4.90000	0.04120	129139.00000	2663204.00000	48.19821	-0.66570	2867065.00000	16.65280

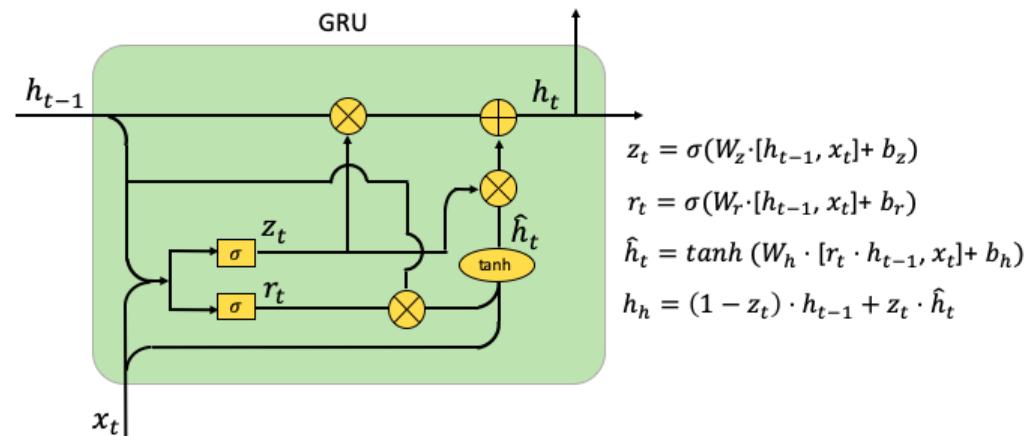
모델 소개

01. 모델 비교 및 선택

(4) DL 모델 : GRU

- 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

```
from tensorflow import keras
model = keras.Sequential()
model.add(keras.layers.GRU())
```



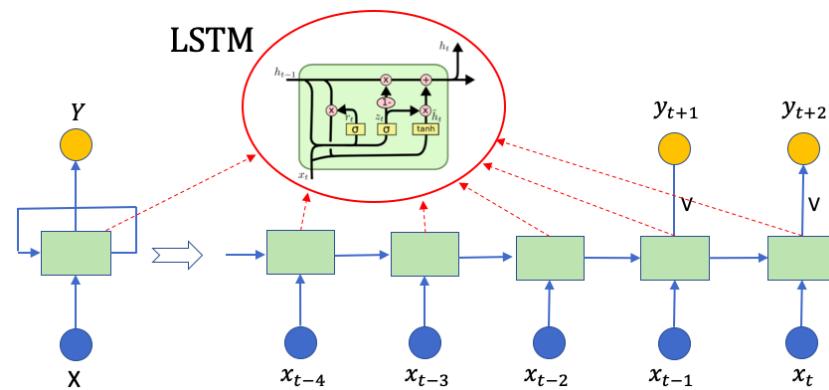
모델 소개

01. 모델 비교 및 선택

(4) DL 모델 : LSTM

- 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

```
from tensorflow import keras
model = keras.Sequential()
model.add(keras.layers.LSTM())
```



모델 소개

01. 모델 비교 및 선택

(5) 최종 모델 선택

- 22.10.20~22.10.21 기간의 실제 예측에 대한 종가 MSE 비교를 통해 DL 모델을 선택

모델	MSE
RandomForest	952.98297
LightGBM	1094.53706
XGBoost	896.81190
GradientBoost	652.52863
GRU	6.43257
LSTM	5.13567

- 최근 1주일(10.17~10.21) 기간의 실제 예측에 대한 종가 MSE 비교를 통해 DL 모델들 중 GRU 모델을 선택

모델	MSE
GRU	10.36801
LSTM	11.07327

* 해당 부분 코드 '모델 학습' 부분에 첨부

수익률 예측 - 전처리 적용

01. Data Reducing

(1) 사용 데이터 재설정

- 파생변수 생성으로 인해 결측치 생성 가능성 존재

→ 2000년부터의 데이터 사용(리노공업은 상장일 2001년 12월인 관계로 2002년부터 사용)

```
In [33]: # 앞서 말했듯이 2000년부터의 데이터만을 사용할 예정
df1 = df[df['일자'] >='2000-01-01']
df1 = df1.sort_values(by = '일자', ascending = True)
df1.reset_index(drop=True, inplace=True)
df1
```

Out[33]:

일자	종가	대비	수익률	시가	고가	저가	거래량	거래대금	상장시기총액	RSI	UO	SR
0 2000-01-04	133.660000	3.640000	0.02800	130.08000	134.76000	128.30000	121256.00000	3067396.00000	NanN	70.37929	54.21263	94.32990
1 2000-01-05	123.860000	-9.80000	-0.07330	126.65000	132.30000	123.55000	153265.00000	3921476.00000	NanN	52.50474	45.75913	43.81443
2 2000-01-06	120.800000	-3.06000	-0.02470	127.45000	127.54000	119.86000	115958.00000	2986051.00000	NanN	48.37352	39.25693	28.04124
3 2000-01-07	119.100000	-1.70000	-0.01410	119.18000	121.97000	116.66000	130572.00000	30059494.00000	NanN	46.19871	40.28562	19.27655
4 2000-01-10	124.110000	5.01000	0.04210	122.99000	125.09000	120.37000	144169.00000	3026380.00000	NanN	52.91690	43.31994	45.10309
...
5627 2022-10-20	288.610000	-2.68000	-0.00920	289.33000	290.07000	286.50000	136904.00000	6043941.00000	1532345342.00000	41.89999	56.07018	60.29499
5628 2022-10-21	288.570000	-0.04000	-0.00010	287.66000	290.38000	287.44000	102369.00000	4677515.00000	1531346564.00000	41.85910	54.53237	60.05900
5629 2022-10-24	291.470000	2.90000	0.01000	293.60000	294.24000	290.52000	116176.00000	5372981.00000	1547506547.00000	45.97492	58.26677	70.90225
5630 2022-10-25	291.580000	0.11000	0.00040	291.29000	293.75000	290.53000	116351.00000	5506859.00000	1547456881.00000	46.13070	50.95743	71.72932
5631 2022-10-26	293.850000	2.27000	0.00780	292.21000	295.26000	291.40000	120348.00000	5804827.00000	1560104520.00000	49.37473	49.06677	88.79699

5632 rows × 23 columns

▲ 코스피 200, 삼성전자, SK 하이닉스, CJ ENM
공통 코드(수치는 상이)

```
In [33]: # 앞서 말했듯이 2002년부터의 데이터만을 사용할 예정
df1 = df[df['일자'] >='2002-01-01']
df1 = df1.sort_values(by = '일자', ascending = True)
df1.reset_index(drop=True, inplace=True)
df1
```

Out[33]:

일자	종가	대비	수익률	시가	고가	저가	거래량	거래대금	시가총액	상장주식 수	RSI	UO	SR	WR	ROC
0 2002-01-02	3256	-65	-0.01970	3334	3351	3229	27530	208088870	49526100000	6630000	27.11126	38.97427	12.19124	-87.80876	0.00000
1 2002-01-03	3421	165	0.05090	3256	3530	3256	47415	368108520	52045500000	6630000	34.24158	41.42926	25.33865	-74.66135	0.00000
2 2002-01-04	3491	70	0.02040	3487	3552	3429	37407	298991590	53106300000	6630000	37.05484	31.57977	30.91633	-69.09367	0.00000
3 2002-01-07	3391	-100	-0.02870	3530	3574	3277	16050	126772460	51581400000	6630000	34.76657	37.43962	22.94821	-77.05179	0.00000
4 2002-01-08	3487	96	0.02830	3399	3487	3312	23268	181523100	53040000000	6630000	38.68137	47.41133	30.59761	-69.40239	-19.98621
...
5140 2022-10-19	133000	-2800	-0.02060	135100	136100	131300	64574	8606886160	2027235210000	15242370	46.34883	47.80269	46.31148	-53.68852	6.82731
5141 2022-10-21	135100	2100	0.01580	131700	135400	131700	18346	245526000	2059244187000	15242370	49.19757	50.44476	54.91803	-45.08197	0.59566
5142 2022-10-24	140000	4900	0.03630	136400	141100	136200	41607	5849318900	2129391800000	15242370	55.17791	55.01627	67.02703	-32.97297	6.62606
5143 2022-10-25	139700	-300	-0.00210	140000	141200	138300	31777	4456453100	2118669430000	15242370	54.75294	48.28835	57.04986	-42.95302	-0.56940
5144 2022-10-26	139000	-700	-0.00500	140000	143000	138200	33589	4710692300	2118669430000	15242370	53.71341	47.08970	52.02703	-47.97297	-1.48831

5145 rows × 24 columns

▲ 리노 공업 코드

수익률 예측 - 전처리 적용

01. Data Reducing

(2) VIF를 활용한 다중공선성 제거

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

2000년부터로 데이터가 변경되었기에 VIF 재확인

```
In [34]: # 일단 null값 column은 vif 확인을 위해 잠시 제거
X_value = df1.drop(['일자','종가','상장시기총액'],axis=1)
y_value = df1['종가']
```

```
In [35]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]
vif
```

Out[35]:

	features	VIF Factor
0	대비	6.17263
1	수익률	4.94861
2	시가	9706.87923
3	고가	10514.66694
4	저가	7182.24945
5	거래량	1.52975
6	거래대금	5.06102
7	RSI	13.97485
8	UO	3.61653
9	SR	146.95559
10	WR	66.75554
11	ROC	4.39025
12	OBV	7.50443
13	BHB	5740.88292
14	BLB	5243.10151
15	EMA	172343.90310
16	WMA	79763.84229
17	MACD	10.37110
18	ADX	1.27886
19	CCI	10.36923

→ '시가','고가','저가','상장시기총액','RSI','SR','WR','BHB','BLB','EMA','WMA','MACD','CCI' 제거

df2 : df1에서 VIF 10 이상의 변수들 제거한 데이터

```
In [36]: # df1 데이터의 VIF가 10이상인 변수들 제거
# 상장시기총액의 경우 종가의 영향을 받는 변수이기에 drop 가능하다고 판단
df2 = df1.drop(['시가','고가','저가','상장시기총액','RSI','SR','WR','BHB','BLB','EMA','WMA','MACD','CCI'], axis=1)
```

```
In [37]: # 결측값 해결
df2.isnull().sum()
```

```
Out[37]: 일자      0
종가      0
대비      0
수익률    0
거래량    0
거래대금  0
UO       0
ROC      0
OBV      0
ADX      0
dtype: int64
```

▲ 코스피 200

수익률 예측 - 전처리 적용

01. Data Reducing

(2) VIF를 활용한 다중공선성 제거

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

2000년부터로 데이터가 변경되었기에 VIF 재확인

```
In [34]: X_value = df1.drop(['일자','증가'],axis=1)
y_value = df1['증가']
```

```
In [35]: vif = pd.DataFrame()
vif['features'] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]
vif
```

Out[35]:

	features	VIF Factor
0	대비	2.13212
1	수익률	2.18485
2	시가	11278.34768
3	고가	10208.18086
4	저가	11049.07477
5	거래량	16.31442
6	거래대금	11.18805
7	시가총액	296.24356
8	상장주식수	14.18673
9	RSI	12.03592
10	UO	3.15488
11	SR	34.20326
12	WR	23.72687
13	ROC	3.40022
14	OBV	7.66410
15	BHB	9269.97186
16	BLB	7644.77796
17	EMA	257361.61159
18	WMA	115971.31721
19	MACD	7.73460
20	ADX	1.33861
21	CCI	9.11265

→ '시가','고가','저가','거래량','거래대금','시가총액','상장주식수','RSI','SR','WR','BHB','BLB','EMA','WMA' 제거

df2 : df1에서 VIF 10 이상의 변수들 제거한 데이터

```
In [36]: # df1 데이터의 VIF가 10이상인 변수들 제거
df2 = df1.drop(['시가','고가','저가','거래량','거래대금','시가총액','상장주식수','RSI','SR','WR','BHB','BLB','EMA','WMA'], axis=1)
```

```
In [37]: # 결측값 해결
df2.isnull().sum()
```

```
Out[37]: 일자      0
증가      0
대비      0
수익률    0
UO        0
ROC       0
OBV       0
MACD      0
ADX       0
CCI       0
dtype: int64
```

▲ 삼성전자

수익률 예측 - 전처리 적용

01. Data Reducing

(2) VIF를 활용한 다중공선성 제거

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

2000년부터로 데이터가 변경되었기에 VIF 재확인

```
In [34]: X_value = df1.drop(['일자','종가'],axis=1)
y_value = df1['종가']
```

```
In [35]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]
vif
```

Out[35]:

	features	VIF Factor
0	대비	2.03154
1	수익률	1.64309
2	시가	2542.94109
3	고가	2021.14116
4	저가	1964.18255
5	거래량	2.68518
6	거래대금	1.91510
7	시가총액	3.92852
8	상장주식수	2.32749
9	RSI	6.97557
10	UO	2.81028
11	SR	20.54279
12	WR	13.29114
13	ROC	2.52243
14	OBV	3.11317
15	BHB	2046.80066
16	BLB	1191.66577
17	EMA	47134.43127
18	WMA	20957.15051
19	MACD	6.53165
20	ADX	1.24248
21	CCI	7.80942

→ '시가','고가','저가','SR','WR','BHB','BLB','EMA','WMA' 제거

df2 : df1에서 VIF 10 이상의 변수들 제거한 데이터

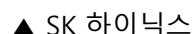
```
In [36]: # df1 데이터의 VIF가 10이상인 변수들 제거
df2 = df1.drop(['시가','고가','저가','SR','WR','BHB','BLB','EMA','WMA'], axis=1)
```

```
In [37]: # 결측값 해결
df2.isnull().sum()
```

Out[37]:

일자	0
종가	0
대비	0
수익률	0
거래량	0
거래대금	0
시가총액	0
상장주식수	0
RSI	0
UO	0
ROC	0
OBV	0
MACD	0
ADX	0
CCI	0

dtype: int64



수익률 예측 - 전처리 적용

01. Data Reducing

(2) VIF를 활용한 다중공선성 제거

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

2002년부터로 데이터가 변경되었기에 VIF 재확인

```
In [34]: X_value = df1.drop(['일자','종가'],axis=1)
y_value = df1['종가']
```

```
In [35]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])
```

Out[35]:

features	VIF Factor
0 대비	3.63183
1 수익률	1.79533
2 시가	8662.04732
3 고가	5787.47166
4 저가	8154.93270
5 거래량	2.13243
6 거래대금	6.35251
7 시가총액	4586.53160
8 상장주식수	3.48066
9 RSI	7.44352
10 UO	1.91245
11 SR	16.60205
12 WR	11.22394
13 ROC	3.15304
14 OBV	2.78582
15 BHB	9094.81921
16 BLB	7049.47646
17 EMA	244426.63169
18 WMA	108826.03155
19 MACD	5.88634
20 ADX	1.39212
21 CCI	6.15490

→ '시가','고가','저가','시가총액','SR','WR','BHB','BLB','EMA','WMA' 제거

df2 : df1에서 VIF 10 이상의 변수들 제거한 데이터

```
In [36]: # df1 데이터의 VIF가 10이상인 변수를 제거
df2 = df1.drop(['시가','고가','저가','시가총액','SR','WR','BHB','BLB','EMA','WMA'], axis=1)
```

```
In [37]: # 결측값 해결
df2.isnull().sum()
```

```
Out[37]: 일자      0
종가      0
대비      0
수익률    0
거래량    0
거래대금  0
시가총액  0
상장주식수 0
RSI       0
UO        0
ROC       0
OBV       0
MACD      0
ADX       0
CCI       0
dtype: int64
```

▲ 리노공업

수익률 예측 - 전처리 적용

01. Data Reducing

(2) VIF를 활용한 다중공선성 제거

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

2000년부터로 데이터가 변경되었기에 VIF 제작인

```
In [34]: X_value = df1.drop(['일자','종가'],axis=1)
y_value = df1['종가']
```

```
In [35]: vif = pd.DataFrame()
vif["features"] = X_value.columns
vif["VIF Factor"] = [variance_inflation_factor(X_value.values, i) for i in range(X_value.shape[1])]
```

```
Out[35]:
```

	features	VIF Factor
0	대비	6.06900
1	수익률	4.56537
2	시가	2660.76550
3	고가	1674.61606
4	저가	1481.67719
5	거래량	5.54936
6	거래대금	5.39865
7	시가총액	14.02977
8	상장주식수	13.22675
9	RSI	15.55192
10	UO	3.13136
11	SR	38.85777
12	WR	27.67196
13	ROC	4.33860
14	OBV	3.64830
15	BHB	1752.45577
16	BLB	1528.48074
17	EMA	51419.92248
18	WMA	24266.33403
19	MACD	10.24817
20	ADX	1.60641
21	CCI	7.18515

→ '시가','고가','저가','시가총액','상장주식수','RSI','SR','WR','BHB','BLB','EMA','WMA','MACD' 제거

df2 : df1에서 VIF 10 이상의 변수들 제거한 데이터

```
In [36]: # df1 데이터의 VIF가 10이상인 변수들 제거
df2 = df1.drop(['시가','고가','저가','시가총액','상장주식수','RSI','SR','WR','BHB','BLB','EMA','WMA','MACD'], axis=1)
```

```
In [37]: # 결측값 해결
df2.isnull().sum()
```

```
Out[37]: 일자      0
종가      0
대비      0
수익률    0
거래량    0
거래대금  0
UO       0
ROC      0
OBV      0
ADX      0
CCI      0
dtype: int64
```

수익률 예측 – 전처리 적용

01. Data Reducing

(3) OLS를 이용한 유의미한 변수 파악

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

```
In [38]: X_value = df.drop(['영화','종가'],axis=1)
y_value = df[['종가']]

In [39]: X_value_1 = X_value.iloc[:,1,:]

In [40]: VIFX = sm.add_constant(X_value_1, has_constant='add')
lin_model_VIF = sm.OLS(y_value, VIFX)
lin_model_VIF.fit()

In [41]: lin_model_VIF.summary()

Out[41]: OLS Regression Results
Dep. Variable: 종가 R-squared: 0.889
Model: OLS Adj. R-squared: 0.889
Method: Least Squares F-statistic: 6427
Date: Wed, 26 Oct 2022 Prob (F-statistic): 0.00
Time: 16:51:14 Log-Likelihood: -37076
No. Observations: 5632 AIC: 5.417e+04
Df Residuals: 5624 BIC: 5.422e+04
Df Model: 7
Covariance Type: nonrobust
            coef std err t P>|t| [0.025 0.975]
const 50.3997 3.223 15.640 0.000 44.082 56.717
대비 1.0126 0.302 3.352 0.001 0.420 1.605
수익률 -120.2561 56.468 -2.130 0.033 -230.956 -9.556
기준금리 -4.096e-05 3.46e-06 -12.068 0.000 -4.77e-05 -3.43e-05
거래량 8.899e-05 2.14e-07 41.674 0.000 8.45e-05 9.32e-05
UO -0.4563 0.003 -8.571 0.000 -0.561 -0.352
ROC 0.5235 0.107 4.998 0.000 0.314 0.733
OBV 3.049e-06 2.81e-06 108.424 0.000 2.99e-06 3.1e-06

Omnibus: 66.969 Durbin-Watson: 0.683
Prob(Omnibus): 0.000 Jarque-Bera (JB): 90.773
Skew: -0.160 Prob(JB): 1.94e-20
Kurtosis: 3.533 Cond. No. 8.34e+09
```

→ 제거할 변수 없음

최종 데이터 df3 : df2에서 히어볼설 결과를 트대로 읽으하지 않고 베스를 제거한 데이터

- $P-value$ 를 보면 모드 유의수준 0.05 이내에서 유의한 \rightarrow 제거할 변수 없음

• [42] 162 162 61

Out[43]:	일자	증가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX
0	2000-01-04	133.660000	3.640000	0.02800	121258.00000	3067398.00000	54.21263	6.36639	3522692.00000	26.50646
1	2000-01-05	123.860000	-9.80000	-0.07330	153265.00000	3921476.00000	45.75913	0.16983	3369427.00000	26.10429
2	2000-01-06	120.800000	-3.06000	-0.02470	116958.00000	2988051.00000	39.25693	1.77774	3252469.00000	24.80416
3	2000-01-07	119.100000	-1.70000	-0.01410	130572.00000	3094945.00000	40.28562	1.85581	3121997.00000	23.17014
4	2000-01-10	124.110000	5.01000	0.04100	144169.00000	3082830.00000	44.31999	1.57475	3266066.00000	22.01308

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified
[2] The condition number is large, 6.24e+09. This might indicate that there are strong multicollinearity or other numerical problems.

수익률 예측 - 전처리 적용

01. Data Reducing

(3) OLS를 이용한 유의미한 변수 파악

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

```
In [38]: X_value = df2.drop(['일자','종가'],axis=1)
y_value = df2['종가']

In [39]: X_value_1 = X_value.iloc[:,8]

In [40]: VIF = sm.add_constant(X_value_1, has_constant='add')
lin_model_VIF = sm.OLS(y_value, VIF)
lin_model_VIF = lin_model_VIF.fit()

In [41]: lin_model_VIF.summary()
```

```
OLS Regression Results
Dep. Variable: 종가 R-squared: 0.717
Model: OLS Adj. R-squared: 0.716
Method: Least Squares F-statistic: 1776.
Date: Wed, 26 Oct 2022 Prob (F-statistic): 0.00
Time: 17:46:09 Log-Likelihood: -6054
No. Observations: 5632 AIC: 1.207e+05
DF Residuals: 5623 BIC: 1.208e+05
DF Model: 8
Covariance Type: nonrobust

coef std err t P>|t| [0.025 0.975]
const 515.7730 1155.139 4.280 0.000 2772.840 7458.708
QDJ 0.2167 0.367 0.560 0.575 -0.541 0.975
수익률 -7368.7044 9368.119 -0.787 0.432 -2.57e+04 1.16e-04
UO 116.1611 21.626 5.372 0.000 73.766 158.554
ROC -117.8802 33.652 -3.502 0.000 -183.832 -51.889
OBV 5.579e-05 5.64e-07 116.587 0.000 6.47e-05 6.69e-05
MACD -0.1936 0.304 -0.637 0.524 -0.789 0.402
ADX 82.4912 18.149 4.543 0.000 46.876 118.036
CCI 1.3972 2.455 0.562 0.974 -3.475 6.269

Omnibus: 797.533 Durbin-Watson: 0.004
Prob(Omnibus): 0.000 Jarque-Bera (JB): 1172.856
Skew: 1.093 Prob(JB): 2.00e-255
Kurtosis: 3.471 Cond. No. 2.14e+10
```

→ '대비', '수익률', 'MACD', 'CCI' 제거

최종 데이터 df3 : df2에서 회귀분석 결과를 도대로 유의하지 않은 변수들 제거한 데이터

• P>|t|를 보면 대비, 수익률, MACD, CCI 제거

```
In [42]: df3 = df2.drop(['대비','수익률','MACD','CCI'],axis=1)
```

```
In [43]: df3.head()
```

```
Out[43]:
   일자    종가      UO      ROC      OBV      ADX
0  2000-01-04  6110  57.54896  18.41085  31399892  33.54690
1  2000-01-05  5580  53.27509  9.19765  29906288  33.34213
2  2000-01-06  5620  51.20576  14.92843  30994099  33.15198
3  2000-01-07  5540  50.05491  15.89958  30187903  31.98967
4  2000-01-10  5770  54.94309  17.75510  31125518  31.20221
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.14e+10. This might indicate that there are strong multicollinearity or other numerical problems.



수익률 예측 - 전처리 적용

01. Data Reducing

(3) OLS를 이용한 유의미한 변수 파악

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

```
In [38]: X_value = df2.drop(['날짜','종가'],axis=1)
y_value = df2['종가']

In [40]: X_value_1 = X_value.iloc[:,1:16]

In [41]: vpx = sm.add_constant(X_value_1,has_constant='add')
lin_model_VP = sm.OLS(y_value,Vpx)
lin_model_VP = lin_model_VP.fit()

In [42]: lin_model_VP.summary()
```

Out[42]: OLS Regression Results

Dep. Variable:	OLS	R-squared:	0.008			
Model:	OLS	Adj. R-squared:	0.007			
Method:	Least Squares	F-statistic:	528.5			
Date:	Wed, 28 Dec 2022	Prob (F-statistic):	0.00			
Time:	17:52:01	Log-Likelihood:	-0.9121			
No. Observations:	8532	AIC:	1.331e+05			
Df Residuals:	8520	BIC:	1.331e+05			
Df Model:	11					
Covariance Type:	oprobust					
coef	std err	t	P> t	[0.025	0.975]	
const	1.378e-05	7679.122	17.938	0.000	1.24e+05	1.53e+05
날짜	-0.008	0.159	-0.182	0.858	-0.338	0.278
종가	-7568.183	423.728	-17.933	0.000	3429.490	-4758.147
개별	-0.731e-09	8.98e-09	-0.082	0.000	-0.000	-0.8e-08
개별2	1.92e-09	8.98e-09	2.158	0.002	0.42e-09	2.74e-08
시가총액	1.86e-09	3.9e-11	48.214	0.000	1.8e-09	1.86e-09
날짜*종가	7.03e-06	1.16e-06	6.947	0.000	4.71e-06	9.35e-06
R2	888.5434	142.711	0.073	0.000	588.873	1148.412
UO	328.2472	106.282	3.088	0.002	116.808	538.592
ROC	372.4371	71.184	5.232	0.000	232.890	611.864
OBV	-0.474e-08	5.11e-07	-0.912	0.000	-0.32e-05	-0.32e-05
MACD	0.438	0.173	2.518	0.000	0.148	0.328
ADX	-0.005	73.132	-13.041	0.000	-107.067	-0.024
CCI	-0.10099	14.325	-0.418	0.000	-119.993	-0.827
Omnibus:	2668.473	Durbin-Watson:	0.028			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	34831.797			
Skew:	2.230	Prob(JB):	0.00			
Kurtosis:	14.322	Cond. No.	1.17e+15			

최종 데이터 df3 : df2에서 회귀분석 결과를 토대로 유의하지 않은 변수들 제거한 데이터
• P>|t|를 보면 대비 제거

```
In [43]: df3 = df2.drop(['대비'],axis=1)
```

```
In [44]: df3.head()
```

Out[44]:

일자	종가	수익률	거래량	거래대금	시가총액	상장주식수	RSI	UO	ROC	OBV	MACD	ADX	CCI
0 2000-01-04	505493	0.08550	9276920	235382580500	9885072442000	348934858	01.50889	48.04165	18.93779	109541490	1022.24001	15.70523	182.38787
1 2000-01-04	453444	-0.10290	7414370	177810167000	8080396173800	348934858	50.37414	45.03837	9.09995	182227120	1083.25405	15.57145	70.34012
2 2000-01-08	426889	-0.05190	6529140	147857558500	7641673348400	348934858	46.28862	42.42411	6.82940	158667980	-795.09394	14.84471	-0.75054
3 2000-01-07	425963	-0.00910	10492270	227868200000	10718388378200	493934858	45.62432	45.93732	5.85377	146205710	-2656.09293	13.83585	-52.45457
4 2000-01-10	422037	-0.00920	8041600	175098030000	10619590404000	493934858	44.93002	42.59997	-8.70494	137164020	-4219.44882	12.89847	-40.35734

Notes:
^[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
^[2] The condition number is large, 1.17e+15. This might indicate that there are strong multicollinearity or other numerical problems.

▲ SK 하이닉스

수익률 예측 - 전처리 적용

01. Data Reducing

(3) OLS를 이용한 유의미한 변수 파악

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

```
In [38]: X_value = df2.drop(['일자','종가'],axis=1)
y_value = df2['수익률']

In [39]: X_value_1 = X_value.iloc[:,1:16]

In [40]: VIFX = sm.add_constant(X_value_1, has_constant='add')
lin_model_VIF = sm.OLS(y_value, VIFX)
lin_model_VIF = lin_model_VIF.fit()

In [41]: lin_model_VIF.summary()

Out[41]:
```

OLS Regression Results

Dep. Variable:	종가	R-squared:	0.771			
Model:	OLS	Adj. R-squared:	0.770			
Method:	Least Squares	F-statistic:	1438.			
Date:	Wed, 20 Oct 2022	Prob (F-statistic):	0.00			
Time:	19:15:00	Log Likelihood:	-59945.			
No. Observations:	5145	AIC:	1.179e+05			
Df Residuals:	5132	BIC:	1.180e+05			
Df Model:	12					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-2.137e-04	372.472	-5.720	0.000	-0.74e-04	-1.41e-04
시가	-0.2040	0.287	-0.707	0.000	-0.25e-04	-1.482
수익률	4.191e-04	1.65e-04	2.534	0.011	9485.593	7434e-04
거래량	-0.1990	0.009	-20.020	0.000	-0.191	-0.192
거래대금	0.855e-08	1.3e-07	70.000	0.000	0.8e-08	1.04e-08
상장주식수	0.0051	0.000	38.224	0.000	0.005	0.005
RSI	105.185	71.443	1.354	0.019	28.398	308.214
UO	-66.8711	38.298	-2.282	0.023	-166.712	-12.830
ROC	1.3228	63.328	0.021	0.983	-122.821	125.496
OBV	-0.0001	4.256e-05	-2.530	0.011	-0.002	4.42e-05
MACD	-0.8200	0.280	-2.908	0.073	-1.08	0.048
ADX	-387.4708	32.198	-11.102	0.000	-420.992	-264.349
CCI	-0.5600	5.808	-0.095	0.025	-11.932	10.832

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large: 2.03e+11. This might indicate that there are strong multicollinearity or other numerical problems.

→ 'ROC', 'MACD', 'CCI' 제거

```
In [42]: df3 = df2.drop(['ROC','MACD','CCI'],axis=1)

In [43]: df3.head()

Out[43]:
```

일자	종가	대비	수익률	거래량	거래대금	상장주식수	RSI	UO	OBV	ADX	
0	2002-01-02	3256	-56	-0.01970	27530	208088870	68300000	27.11120	38.97427	-165930	0.00000
1	2002-01-03	3421	165	0.06090	47415	368108520	68300000	34.24160	41.42928	-116515	0.00000
2	2002-01-04	3491	70	0.02040	37407	298961590	68300000	37.05484	31.57977	-31108	0.00000
3	2002-01-07	3391	-100	-0.02870	18050	128772460	68300000	34.78557	37.43982	-97158	0.00000
4	2002-01-08	3487	96	0.02830	23288	181523100	68300000	38.88137	47.41133	-73890	0.00000

▲ 리노공업

수익률 예측 - 전처리 적용

01. Data Reducing

(3) OLS를 이용한 유의미한 변수 파악

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

```
In [38]: X_value = df2.drop(['일자','종가'],axis=1)
y_value = df2['종가']

In [39]: X_value_1 = X_value.iloc[:,1:6]

In [40]: VIF = sm.add_constant(X_value_1, has_constant='add')
lin_model_VIF = sm.OLS(y_value, VIF)
lin_model_VIF = lin_model_VIF.fit()

In [41]: lin_model_VIF.summary()
```

Out[41]: OLS Regression Results

Dep. Variable:	종가	R-squared:	0.437			
Model:	OLS	Adj. R-squared:	0.438			
Method:	Least Squares	F-statistic:	484.1			
Date:	Wed, 28 Oct 2022	Prob (F-statistic):	0.00			
Time:	18:37:16	Log-Likelihood:	-70020.			
No. Observations:	6532	AIC:	1.404e+05			
Df Residuals:	6522	BIC:	1.404e+05			
Df Model:	9					
Covariance Type:	nonrobust					
coef	std err	t	P> t	[0.025	0.975]	
const	1.557e+04	6555.405	2.339	0.019	2522.318	2.80e+04
대비	-0.7247	0.39	-1.859	0.003	-1.490	0.041
수익률	1.650e+05	5.84e+04	2.887	0.007	4.24e+04	2.71e+05
거래량	-0.4238	0.00	-26.357	0.00	-0.455	-0.392
거래대금	3.177e+08	1.49e-07	24.070	0.000	3.29e-08	3.87e-08
UO	1384.4718	97.022	14.094	0.000	1174.272	1554.872
ROC	158.0057	112.735	1.402	0.161	-02.999	379.011
OBV	0.0018	8.74e-05	20.885	0.000	0.002	0.002
ADX	05.6251	78.625	0.852	0.394	-87.013	220.804
CCI	-49.1940	12.032	-4.009	0.000	-72.701	-25.807
Omnibus:	038.055	Durbin-Watson:	0.968			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1580.340			
Skew:	1.069	Prob(JB):	0.00			
Kurtosis:	4.379	Cond. No.	1.01e+12			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.01e+12. This might indicate that there are strong multicollinearity or other numerical problems.

```
→ '대비', 'ROC', 'ADX' 제거
```

```
In [42]: df3 = df2.drop(['대비','ROC','ADX'],axis=1)
```

```
In [43]: df3.head()
```

```
Out[43]:
```

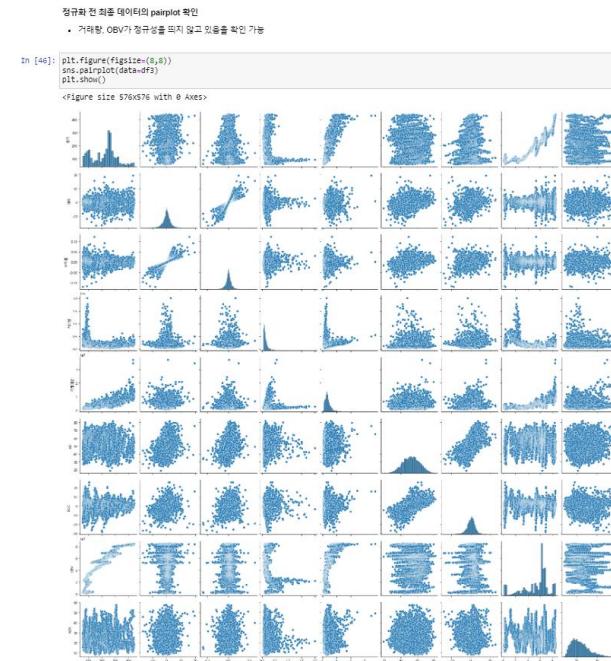
일자	종가	수익률	거래량	거래대금	UO	OBV	CCI	
0	2000-01-04	127816	-0.05480	213120	15471024100	38.86668	2108490	-72.91416
1	2000-01-05	113847	-0.10930	228249	14977732400	29.70141	1878241	-107.69913
2	2000-01-06	109133	-0.04140	219455	13745865300	28.50313	1658788	-117.35113
3	2000-01-07	98910	-0.11200	210916	11831309800	29.88013	1447870	-140.48208
4	2000-01-10	99005	0.02169	198791	11139399700	34.88571	1644861	-123.45120

수익률 예측 - 전처리 적용

02. 최종 데이터 생성 및 재시각화

(1) heatmap, pairplot

- heatmap과 pairplot을 통해 최종 데이터의 모습 재시각화



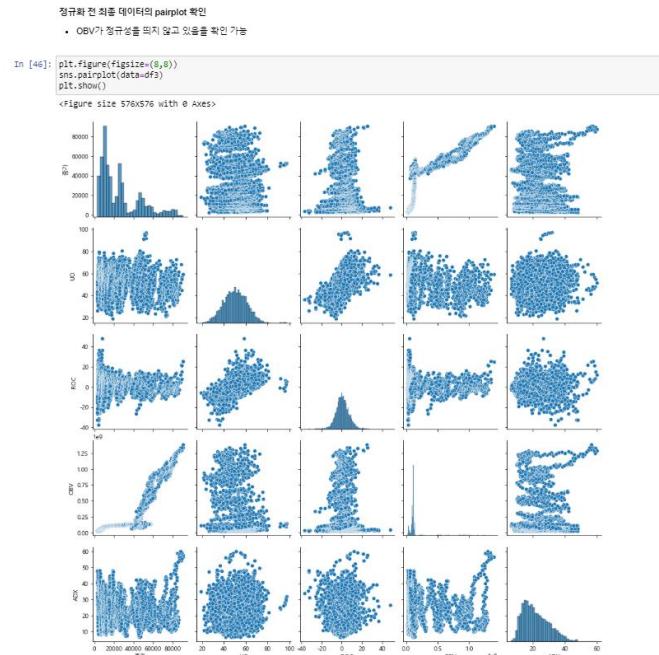
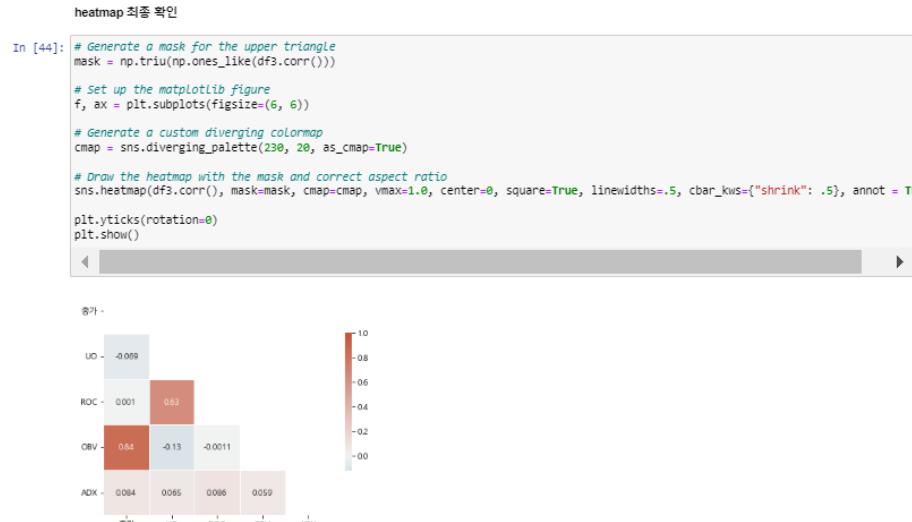
▲ 코스피 200

수익률 예측 - 전처리 적용

02. 최종 데이터 생성 및 재시각화

(1) heatmap, pairplot

- heatmap과 pairplot을 통해 최종 데이터의 모습 재시각화



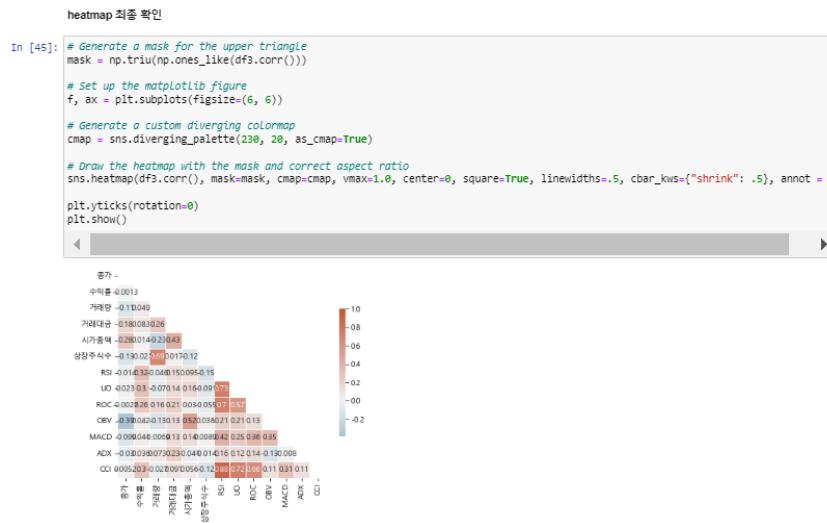
▲ 삼성전자

수익률 예측 - 전처리 적용

02. 최종 데이터 생성 및 재시각화

(1) heatmap, pairplot

- heatmap과 pairplot을 통해 최종 데이터의 모습 재시각화



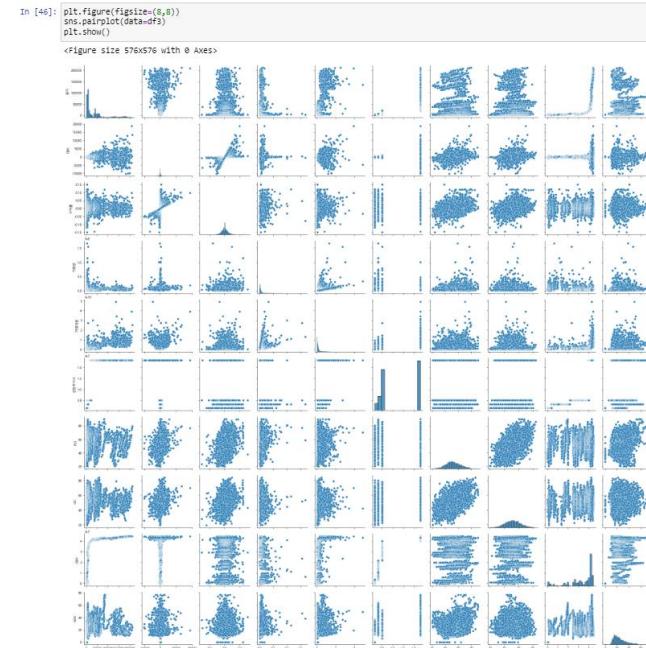
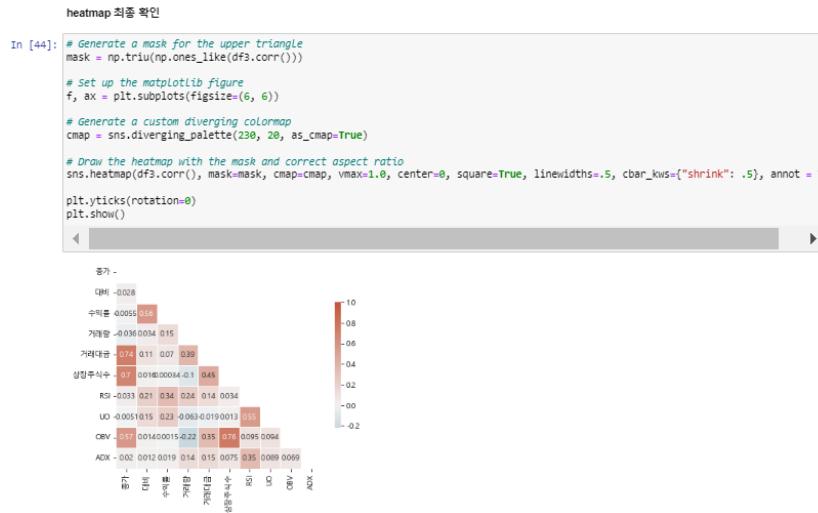
▲ SK 하이닉스

수익률 예측 - 전처리 적용

02. 최종 데이터 생성 및 재시각화

(1) heatmap, pairplot

- heatmap과 pairplot을 통해 최종 데이터의 모습 재시각화



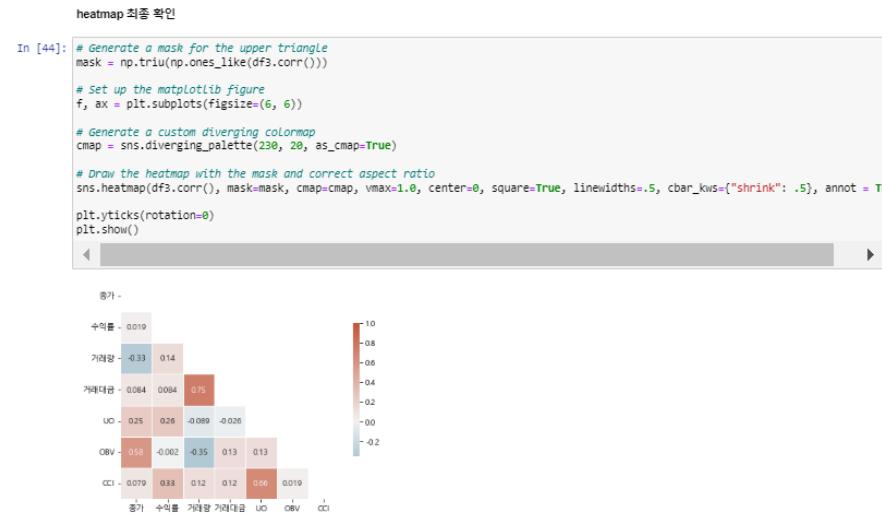
▲ 리노공업

수익률 예측 - 전처리 적용

02. 최종 데이터 생성 및 재시각화

(1) heatmap, pairplot

- heatmap과 pairplot을 통해 최종 데이터의 모습 재시각화



▲ CJ ENM

수익률 예측 - 전처리 적용

02. 최종 데이터 생성 및 재시각화

(2) ML 모델 사용을 위한 추가적 데이터 변경(DL은 그대로 유지)

- ML 회귀 분석을 위해서는 독립변수 값들이 존재해야 종속변수 값을 추론할 수 있음
- 기존 데이터로는 3일 치의 독립변수를 알 수 없다는 한계 인식
 - 종가를 3일치 위로 Shift 하는 작업을 수행
 - 종가_y 생성

종가를 위로 3칸 밀어올린 column 생성
 • 나머지 변수들로 하여금 3일 뒤 데이터를 예측하게 만들기 위해 y값을 위로 3칸 밀어올림

```
In [25]: df.head()
Out[25]:
   일자    종가    대비    수익률    시가    고가    저가    거래량    거래대금    상장
   시가    종액
0  1999-06-01  88.27000  2.59000  0.03020  85.64000  89.29000  84.78000  99007.00000  1644763.00000  NaN  100.00000  0.00000  77.38359  -22.61641  0.00000  99
1  1999-06-02  90.86000  2.59000  0.02930  88.40000  92.11000  88.40000  149224.00000  2472286.00000  NaN  100.00000  31.01796  82.94679  -17.05321  0.00000  248
2  1999-06-03  91.48000  2.62000  0.00680  91.13000  92.67000  89.07000  148286.00000  2272050.00000  NaN  100.00000  41.84100  84.91762  -15.08238  0.00000  396
3  1999-06-04  94.09000  2.61000  0.02850  91.64000  94.68000  91.49000  133969.00000  2366483.00000  NaN  100.00000  50.23102  94.04040  -5.95960  0.00000  530
4  1999-06-07  99.96000  5.87000  0.06240  95.14000  99.96000  95.14000  184467.00000  3246448.00000  NaN  100.00000  64.12940  100.00000  -0.00000  0.00000  714
```

```
In [66]: df['종가_y'] = df['종가'].shift(-3)
df.head()
Out[66]:
   거래대금    상장
   시가    종액    RSI    UO    SR    WR    ROC    OBV    BHB    BLB    EMA    WMA    MACD    ADX    CCI    종가_y
0       NaN  100.00000  0.00000  77.38359  -22.61641  0.00000  99007.00000  88.27000  88.27000  88.27000  0.00000  0.00000  0.00000  0.00000  94.09000
1       NaN  100.00000  31.01796  82.94679  -17.05321  0.00000  248231.00000  92.15500  86.97500  88.66848  0.00000  0.20661  0.00000  66.66667  99.96000
2       NaN  100.00000  41.84100  84.91762  -15.08238  0.00000  396517.00000  92.98395  87.42272  89.10101  0.00000  0.41559  0.00000  63.93772  101.18000
3       NaN  100.00000  50.23102  94.04040  -5.95960  0.00000  530486.00000  95.31366  87.03634  89.86854  0.00000  0.78279  0.00000  114.14601  94.54000
4       NaN  100.00000  64.12940  100.00000  -0.00000  0.00000  714953.00000  100.87527  84.98873  91.42108  0.00000  1.52982  0.00000  138.34374  101.61000
```

수익률 예측 - 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(1) ML 모델 훈련을 위한 준비

- ① 날짜를 index로 변경
- ② train, test set 분리 + 실제 예측할 값을 real_test로 분리
- ③ MSE 성능지표 함수 생성

ML

In [53]: df3.tail()

Out[53]:

일자	증가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX	증가_y	
5624	2022-10-17	289.57000	0.21000	0.00070	111597.00000	4981197.00000	52.86745	-0.21709	80366684.00000	32.20128	288.61000
5625	2022-10-18	293.59000	0.42000	0.01390	117118.00000	5739328.00000	57.88878	3.67245	80483802.00000	31.04973	288.85000
5626	2022-10-19	291.29000	-2.30000	-0.00780	110851.00000	5688842.00000	53.23347	3.05678	80372951.00000	29.66979	NaN
5627	2022-10-20	288.61000	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262	NaN
5628	2022-10-21	288.85000	0.24000	0.00080	72821.00000	3358988.00000	55.13946	0.20120	80308868.00000	28.37462	NaN

In [54]: all_data = df3.drop(['증가'], axis=1).set_index('일자')
all_data

Out[54]:

일자	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX	증가_y
2000-01-04	3.64000	0.02800	121258.00000	3067398.00000	54.21263	6.36639	3522692.00000	26.50646	119.10000
2000-01-05	-8.00000	-0.07330	153265.00000	3821476.00000	45.75913	0.16983	3369427.00000	26.10429	124.11000
2000-01-06	-3.06000	-0.02470	116958.00000	2988051.00000	39.25693	1.77774	3252469.00000	24.80416	123.11000
2000-01-07	-1.70000	-0.01410	130572.00000	3009494.00000	40.28562	1.85581	3121897.00000	23.17014	119.81000
2000-01-10	5.01000	0.04210	144169.00000	3028380.00000	44.31994	5.72451	3266066.00000	22.01308	119.22000
...
2022-10-17	0.21000	0.00070	111597.00000	4981197.00000	52.86745	-0.21709	80366684.00000	32.20128	288.61000
2022-10-18	4.02000	0.01390	117118.00000	5739328.00000	57.88878	3.67245	80483802.00000	31.04973	288.85000
2022-10-19	-2.30000	-0.00780	110851.00000	5688842.00000	53.23347	3.05678	80372951.00000	29.66979	NaN
2022-10-20	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262	NaN
2022-10-21	0.24000	0.00080	72821.00000	3358988.00000	55.13946	0.20120	80308868.00000	28.37462	NaN

5629 rows x 9 columns

In [55]: # 실제 예측해야 할 데이터 : 증가_y의 NaN 값
real_test = all_data.iloc[-3:]
real_test

Out[55]:

일자	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX	증가_y
2022-10-19	-2.30000	-0.00780	110851.00000	5688842.00000	53.23347	3.05678	80372951.00000	29.66979	NaN
2022-10-20	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262	NaN
2022-10-21	0.24000	0.00080	72821.00000	3358988.00000	55.13946	0.20120	80308868.00000	28.37462	NaN

In [56]: # all_data를 train과 test로 분할
train(2000-2022.10.16), test(2022.10.17~)train = all_data[['2022-10-16']]
X_train = train.loc[:, :-1]
y_train = train.loc[:, -1]In [57]: test = all_data['2022-10-17'].iloc[:-3]
X_test = test.loc[:, :-1]
y_test = test.loc[:, -1]In [58]: # 성능지표 생성
from sklearn.metrics import mean_squared_error

```
def confirm_result(y_test, y_pred):
    MSE = mean_squared_error(y_test, y_pred)
    pd.options.display.float_format = '{:.5f}'.format
    Result = pd.DataFrame(data=[MSE], index=['MSE'], columns=['Results'])
    return Result
```

수익률 예측 - 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(2) ML 모델 결과 비교

- ① 라이브러리에서부터 모델들을 import
- ② 모델 클래스에 X_train, y_train을 훈련
- ③ y_test값과 X_test를 통해 예측한 y_pred 값과 비교를 통해 MSE 측정
 - GradientBoost가 ML 모델들 중에는 성능이 가장 좋음
 - 20, 21일 예측에 대한 **ML의 최소 MSE = 652.52863**

RandomForest

```
In [59]: from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# 20,21일 예측에 대한 MSE
confirm_result(y_test, y_pred)
```

Out[59]:
Results
MSE 952.98297

LightGBM

```
In [60]: from lightgbm import LGBMRegressor
lgbm = LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred = lgbm.predict(X_test)

# 20,21일 예측에 대한 MSE
confirm_result(y_test, y_pred)
```

Out[60]:
Results
MSE 1094.53706

XGBoost

```
In [61]: from xgboost import XGBRegressor
xgb = XGBRegressor(random_state=42)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)

# 20,21일 예측에 대한 MSE
confirm_result(y_test, y_pred)
```

Out[61]:
Results
MSE 896.81190

GradientBoost

```
In [62]: from sklearn.ensemble import GradientBoostingRegressor
gb = GradientBoostingRegressor(random_state=42)
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)

# 20,21일 예측에 대한 MSE
confirm_result(y_test, y_pred)
```

Out[62]:
Results
MSE 652.52863

수익률 예측 – 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(3) DL 모델 훈련을 위한 준비 – 데이터 변환

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

```
In [48]: df4 = df3.copy()

In [49]: df4

Out[49]:
   일자    종가    대비    수익률    거래량    거래대금    UO    ROC    OBV    ADX
0  2000-01-04  133.66000  3.64000  0.02800  121258.00000  3067398.00000  54.21263  6.36639  3522692.00000  26.50646
1  2000-01-05  123.86000 -9.80000 -0.07330  153265.00000  3921476.00000  45.75913  0.16983  3369427.00000  26.10429
2  2000-01-06  120.80000 -3.06000 -0.02470  116958.00000  2988051.00000  39.25693  1.77774  3252469.00000  24.80416
3  2000-01-07  119.10000 -1.70000 -0.01410  130572.00000  3009494.00000  40.28562  1.85581  3121897.00000  23.17014
4  2000-01-10  124.11000  5.01000  0.04210  144169.00000  3028380.00000  44.31994  5.72451  3266066.00000  22.01308
...
...
5624 2022-10-17  289.57000  0.21000  0.00070  111597.00000  4981197.00000  52.86745 -0.21709  80366684.00000  32.20128
5625 2022-10-18  293.59000  4.02000  0.01390  117118.00000  5739328.00000  57.88878  3.67245  80483802.00000  31.04973
5626 2022-10-19  291.29000 -2.30000 -0.00780  110851.00000  568842.00000  53.23347  3.05678  80372951.00000  29.66979
5627 2022-10-20  288.61000 -2.68000 -0.00920  136904.00000  6043941.00000  56.07018  2.57677  80236047.00000  29.03262
5628 2022-10-21  288.85000  0.24000  0.00080  72821.00000  3358988.00000  55.13946  0.20120  80308668.00000  28.37462

5629 rows × 10 columns
```

```
In [50]: new_row1 = {'일자': '2022-10-24'}
new_row2 = {'일자': '2022-10-25'}
new_row3 = {'일자': '2022-10-26'}

df4 = df4.append(new_row1, ignore_index=True)
df4 = df4.append(new_row2, ignore_index=True)
df4 = df4.append(new_row3, ignore_index=True)

df4['일자'] = pd.to_datetime(df4['일자'], format = '%Y/%m/%d')

df4 = df4.fillna(0)
```

수익률 예측 - 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(3) DL 모델 훈련을 위한 준비 – 데이터 변환

② train과 test 셋 분리

③ MinMaxScaler를 활용한 데이터 scaling 진행

```
In [140]: def ts_train_test_normalize(all_data, time_steps, for_periods):
    """
    input:
        data: dataframe의 일자 & 종가
    output:
        X_train, y_train: data from 2000/1/4-2022-10-16
        X_test : data from 2022-10-17
        sc : MinMaxScaler fit to the training data
    """
    # train, test set 생성
    ts_train = all_data[:'2022-10-16'].iloc[:,0:1].values
    ts_test = all_data['2022-10-17':].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)

    # 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"][:'2022-10-16'], all_data["종가"]['2022-10-17':]), 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

In [141]: X_train, y_train, X_test, sc = ts_train_test_normalize(all_data, 5, 1)
          X_train.shape[0], X_train.shape[1]
Out[141]: (5618, 5)
```

X_train의 Tensor를 확인했을 때 점점
한 칸씩 위로 올라오는 것을 확인 가능

```
In [143]: X_train
Out[143]: array([
[[0.19779271],
[0.17216309],
[0.16416037],
[0.15971441],
[0.17281691]],

[[0.17216309],
[0.16416037],
[0.15971441],
[0.17281691],
[0.17020164]],

[[0.16416037],
[0.15971441],
[0.17281691],
[0.17020164],
[0.16157125]],

[[0.15971441],
[0.16416037],
[0.17281691],
[0.17020164],
[0.16157125]],

[[0.16416037],
[0.15971441],
[0.17281691],
[0.17020164],
[0.16157125]],

[[0.16157125],
[0.16020824],
[0.16002824],
[0.16002824],
[0.16002824]],

[[0.16002824],
[0.15984827],
[0.15973226],
[0.15973226],
[0.15973226]]])
```

```
In [144]: y_train
Out[144]: array([[0.17020164],
[0.16157125],
[0.16002824],
[0.16002824],
[0.16002824],

[[0.16002824],
[0.15984827],
[0.15973226],
[0.15973226],
[0.15973226]]])
```

수익률 예측 - 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(3) DL 모델 훈련을 위한 준비 – 데이터 변환

- 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

```
In [140]: def ts_train_test_normalize(all_data, time_steps, for_periods):
    """
    input:
        data: dataframe의 일자 & 종가
    output:
        X_train, y_train: data from 2000/1/4-2022-10-16
        X_test : data from 2022-10-17
        sc : MinMaxScaler fit to the training data
    """
    # train, test set 생성
    ts_train = all_data[:'2022-10-16'].iloc[:,0:1].values
    ts_test = all_data['2022-10-17':].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)

    # 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"][:'2022-10-16'], all_data["종가"]['2022-10-17':]), 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
```

In [141]: X_train, y_train, X_test, sc = ts_train_test_normalize(all_data, 5, 1)
X_train.shape[0], X_train.shape[1]
Out[141]: (5618, 5)

X_train의 Tensor를 확인했을 때 점점
한 칸씩 위로 올라오는 것을 확인 가능

```
In [143]: X_train
Out[143]: array([[0.19779271],
   [0.17216309],
   [0.16416037],
   [0.15971441],
   [0.17281691]],

[[0.17216309],
   [0.16416037],
   [0.15971441],
   [0.17281691],
   [0.17020164]],

[[0.16416037],
   [0.15971441],
   [0.17281691],
   [0.17020164],
   [0.16157125]],

...,
[[0.58406779],
   [0.60213929],
   [0.6041792 ],
   [0.60995894],
   [0.60773596]],

[[0.60213929],
   [0.6041792 ],
   [0.60995894],
   [0.60773596],
   [0.59348275]],

[[0.6041792 ,
   [0.60995894],
   [0.60773596],
   [0.59348275],
   [0.59905327]]])
```

In [144]: y_train
Out[144]: array([[0.17020164],
 [0.16157125],
 [0.16002824],
 ...,
 [0.59348275],
 [0.59905327],
 [0.58723226]])

수익률 예측 - 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(3) DL 모델 훈련을 위한 준비 – GRU 모델 생성

- keras.Sequential, keras.layers, keras.optimizers 활용

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
- 수행 결과를 plot으로 그리는 함수 또한 생성

```
In [148]: def GRU_model(X_train, y_train, X_test, sc):
    # 모델 생성
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, 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))

    # compile
    my_GRU_model.compile(optimizer = SGD(lr = 0.01, decay = 1e-7,
                                         momentum = 0.9, nesterov = False),
                          loss = 'mean_squared_error')

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

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

    return my_GRU_model, GRU_prediction
```

```
In [149]: def actual_pred_plot(preds):
    """
    Plot the actual vs prediction
    """
    actual_pred = pd.DataFrame(columns = ['종가', 'prediction'])
    actual_pred['종가'] = all_data.loc['2022-10-17':,'종가'][0:len(preds)]
    actual_pred['prediction'] = preds[:,0]

    from keras.metrics import MeanSquaredError
    m = MeanSquaredError()
    m.update_state(np.array(actual_pred['종가']), np.array(actual_pred['prediction']))

    return (m.result().numpy(), actual_pred.plot())
```

수익률 예측 - 모델 학습

01. 모델 비교 및 선택(코스피 200 데이터 기준)

(3) DL 모델 훈련을 위한 준비 – LSTM 모델 생성

- keras.Sequential, keras.layers, keras.optimizers 활용

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

LSTM 모델

```
In [100]: def LSTM_model(X_train, y_train, X_test, sc):
    # 모델 생성
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, GRU, LSTM
    from keras.optimizers import SGD

    # LSTM층 구조
    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))

    # Compile
    my_LSTM_model.compile(optimizer = SGD(lr = 0.01, decay = 1e-7, momentum = 0.9, nesterov = False),loss = 'mean_squared_error')

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

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

    return my_LSTM_model, LSTM_prediction
```

수익률 예측 - 모델 학습

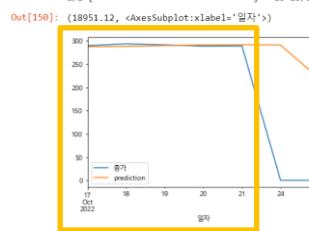
01. 모델 비교 및 선택(코스피 200 데이터 기준)

(4) DL 모델 결과 비교

- 0으로 추가로 지정한 값으로 인해 그래프 상에서 급락하는 형태 생성 → 노란색 박스 부분만 확인
- 20, 21일 예측에 대한 **DL의 최소 MSE = 5.13567**
- 추가적으로 저번 주 5일 동안의 예측 결과 비교 결과 GRU 모델이 더 우수

```
In [150]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
actual_pred_plot(GRU_prediction)

WARNING:tensorflow:6 out of the last 6 cells to `function Model.make_p
B88030> triggered tf.function retracing. Tracing is expensive and the
@tf.function repeatedly in a loop, (2) passing tensors with different
r (1), please define your @tf.function outside of the loop. For (2), @
id unnecessary retracing. For (3), please refer to https://www.tensorf
www.tensorflow.org/api_docs/python/tf/function for more details.
1/1 [*****] - 1s/step
```



```
In [151]: y_pred_gru = pd.DataFrame(GRU_prediction[:, 0])
y_test_gru = all_data.loc['2022-10-17':,'종가'][0:len(GRU_prediction)]
y_test_gru.reset_index(drop=True, inplace=True)
```

```
In [152]: # 20,21일 예측에 대한 MSE
confirm_result(y_test_gru[3:5], y_pred_gru[3:5])
```

```
Out[152]:
```

Results	
MSE	6.43257

```
In [153]: # 17-21일 예측에 대한 MSE
confirm_result(y_test_gru[5:], y_pred_gru[5:])
```

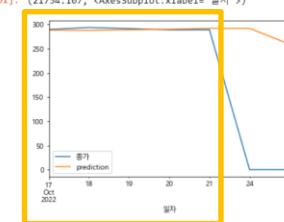
```
Out[153]:
```

Results	
MSE	10.36801

```
In [101]: my_LSTM_model, LSTM_prediction = LSTM_model(X_train, y_train, X_test, sc)
LSTM_prediction[1:10]
actual_pred_plot(LSTM_prediction)

1/1 [*****] - 2s/step
```

```
Out[101]: (21754.107, <AxesSubplot:xlabel='일자'>)
```



```
In [102]: y_pred_lstm = pd.DataFrame(LSTM_prediction[:, 0])
y_test_lstm = all_data.loc['2022-10-17':,'종가'][0:len(LSTM_prediction)]
y_test_lstm.reset_index(drop=True, inplace=True)
```

```
In [103]: X_test.shape
```

```
Out[103]: (7, 5, 1)
```

```
In [114]: # 20,21일 예측에 대한 MSE
confirm_result(y_test_lstm[3:5], y_pred_lstm[3:5])
```

```
Out[114]:
```

Results	
MSE	5.13567

```
In [123]: # 17-21일 예측에 대한 MSE
confirm_result(y_test_lstm[5:], y_pred_lstm[5:])
```

```
Out[123]:
```

Results	
MSE	11.07327

최종적으로 GRU 모델을 선택!

수익률 예측 - 모델 학습

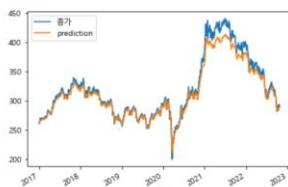
01. 모델 비교 및 선택(코스피 200 데이터 기준)

(5) GRU 모델의 예측 시각화

- 추가적으로 GRU 모델의 성능을 확인하고자 했음
- train을 2016년까지, test를 그 이후부터 현재까지로 지정하여 훈련 및 테스트 후 시각화

```
In [56]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
actual_pred_plot(GRU_prediction)

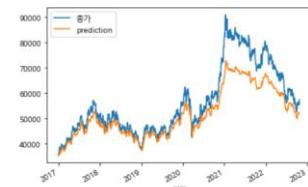
45/45 [=====] - 2s 4ms/step
Out[56]: (113, 44188, <AxesSubplot:xlabel='일자'>)
```



▲ 코스피 200

```
In [56]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
actual_pred_plot(GRU_prediction)

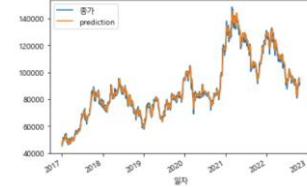
45/45 [=====] - 2s 3ms/step
Out[56]: (40173104.0, <AxesSubplot:xlabel='일자'>)
```



▲ 삼성전자

```
In [57]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
actual_pred_plot(GRU_prediction)

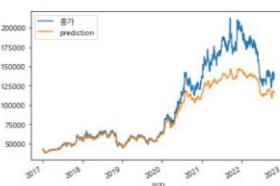
45/45 [=====] - 2s 4ms/step
Out[57]: (8165094.5, <AxesSubplot:xlabel='일자'>)
```



▲ SK 하이닉스

```
In [56]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
actual_pred_plot(GRU_prediction)

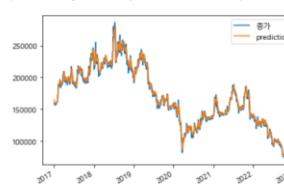
45/45 [=====] - 1s 2ms/step
Out[56]: (452787870.0, <AxesSubplot:xlabel='일자'>)
```



▲ 리노공업

```
In [56]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
actual_pred_plot(GRU_prediction)

45/45 [=====] - 2s 4ms/step
Out[56]: (33611720.0, <AxesSubplot:xlabel='일자'>)
```



▲ CJ ENM

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(1) 10.27 종가 예측

- 3일치 예측을 위해 0을 담고 있는 column들 생성

	In [60]:	df4
	Out[60]:	
0		일자 증가 대비 수익률 거래량 거래대금 UO ROC OBV ADX
0		2000-01-04 133.66000 3.64000 0.02800 121258.00000 3067398.00000 54.21263 6.36639 3522692.00000 26.50646
1		2000-01-05 123.86000 -9.80000 -0.07330 153265.00000 3921476.00000 45.75913 0.16983 3369427.00000 26.10429
2		2000-01-06 120.80000 -3.06000 -0.02470 116958.00000 2988051.00000 39.25693 1.77774 3252469.00000 24.80416
3		2000-01-07 119.10000 -1.70000 -0.01410 130572.00000 3009494.00000 40.28562 1.85581 3121897.00000 23.17014
4		2000-01-10 124.11000 5.01000 0.04210 144169.00000 3028380.00000 44.31994 5.72451 3266066.00000 22.01308
...		...
5627		2022-10-20 288.61000 -2.68000 -0.00920 136904.00000 6043941.00000 56.07018 2.57677 80236047.00000 29.03262
5628		2022-10-21 288.57000 -0.04000 -0.00010 102369.00000 4677515.00000 54.53237 0.10407 80133678.00000 28.37462
5629		2022-10-24 291.47000 2.90000 0.01000 116176.00000 5372981.00000 58.28677 0.83723 80249854.00000 26.96330
5630		2022-10-25 291.58000 0.11000 0.00040 116351.00000 5506859.00000 50.95743 0.10987 80366205.00000 25.65280
5631		2022-10-26 293.85000 2.27000 0.00780 120348.00000 5804827.00000 49.08677 1.18453 80486553.00000 24.11615

5632 rows × 10 columns

```
In [61]: new_row1 = {'일자' : '2022-10-27'}
new_row2 = {'일자' : '2022-10-28'}
new_row3 = {'일자' : '2022-10-31'}
new_row4 = {'일자' : '2022-11-01'}
new_row5 = {'일자' : '2022-11-02'}

df4 = df4.append(new_row1, ignore_index=True)
df4 = df4.append(new_row2, ignore_index=True)
df4 = df4.append(new_row3, ignore_index=True)
df4 = df4.append(new_row4, ignore_index=True)
df4 = df4.append(new_row5, ignore_index=True)

df4['일자'] = pd.to_datetime(df4['일자'], format = '%Y/%m/%d')

df4 = df4.fillna(0)
```

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(1) 10.27 종가 예측

- 과거 5일의 데이터를 이용해 1일을 예측하도록 하는 데이터 변환(27일을 예측하도록 train, test 지정)

```
In [65]: def ts_train_test_normalize(all_data, time_steps, for_periods):
    """
    input:
        data: dataframe의 일자 & 종가
    output:
        X_train, y_train: data from 2000/1/4-2022-10-26
        X_test : data from 2022-10-27
        sc : MinMaxScaler fit to the training data
    """
    # train, test set 설정
    ts_train = all_data['2022-10-26'].iloc[:,0:1].values
    ts_test = all_data['2022-10-27'].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 steps의 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])
        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"][:'2022-10-26'], all_data["종가"]['2022-10-27':]), 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

In [66]: X_train, y_train, X_test, sc = ts_train_test_normalize(all_data, 5, 1)
X_train.shape[0], X_train.shape[1]
Out[66]: (5626, 5)
```

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(1) 10.27 종가 예측

- GRU 모델 생성

```
In [70]: def GRU_model(X_train, y_train, X_test, sc):
    # 모델 생성
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, 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))

    # compile
    my_GRU_model.compile(optimizer = SGD(lr = 0.01, decay = 1e-7,
                                         momentum = 0.9, nesterov = False),
                          loss = 'mean_squared_error')

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

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

    return my_GRU_model, GRU_prediction
```

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(1) 10.27 종가 예측

- GRU 모델 훈련 이후 예측 진행
 - y_pred_gru의 0번째 값이 27일에 대한 예측값
- 코스피 200 27일 종가 : 291.6902770996094**

```
In [71]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
1/1 [=====] - 1s 1s/step
Out[71]: array([[222.36119, 225.6311,
   144.43262, 148.5742],
   [ 79.23406,  82.20282]], dtype=float32)

In [72]: y_pred_gru = pd.DataFrame(GRU_prediction[:, 0])
y_test_gru = all_data.loc['2022-10-27', '종가'][0:len(GRU_prediction)]
y_test_gru.reset_index(drop=True, inplace=True)

In [73]: y_test_gru
Out[73]: 0    0.00000
1    0.00000
2    0.00000
3    0.00000
Name: 종가, dtype: float64

In [74]: y_pred_gru
Out[74]: 0
          0    291.69028
1    222.36119
2    144.43262
3    79.23406

In [75]: np.float_(np.float_(y_pred_gru.values.tolist()[0]))
Out[75]: 291.6902770996094

10월 27일 예측값

In [76]: print(np.float_(np.float_(y_pred_gru.values.tolist()[0])))
291.6902770996094
```

수익률 예측 – 최종 예측

01. 최종 모델 학습(GRU 모델)

(1) 10.27 종가 예측

- 해당 종가를 0으로 지정된 27일 종가에 입력

In [77]: `all_data.tail()`

Out[77]:

일자	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX
2022-10-27	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-28	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-31	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-01	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-02	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

In [78]: `all_data.loc['2022-10-27']['종가'] = np.float_(np.float_(y_pred_gru.values.tolist()[0]))`

In [79]: `all_data.tail(10)`

Out[79]:

일자	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX
2022-10-20	288.61000	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262
2022-10-21	288.57000	-0.04000	-0.00010	102369.00000	4677515.00000	54.53237	0.10407	80133678.00000	28.37462
2022-10-24	291.47000	2.90000	0.01000	116176.00000	5372981.00000	58.28677	0.83723	80249854.00000	26.96330
2022-10-25	291.58000	0.11000	0.00040	116351.00000	5506859.00000	50.95743	0.10987	80366205.00000	25.65280
2022-10-26	293.85000	2.27000	0.00780	120348.00000	5804827.00000	49.08677	1.18453	80486553.00000	24.11615
2022-10-27	291.69028	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-28	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-31	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-01	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-02	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(2) 10.28 종가 예측

- 과거 5일의 데이터를 이용해 1일을 예측하도록 하는 데이터 변환(28일을 예측하도록 train, test 지정)
- 예측된 27일의 종가가 입력된 데이터셋을 사용

```
10월 28일 예측

In [80]: def ts_train_test_normalize(all_data, time_steps, for_periods):
    """
    input:
        data: dataframe의 일자 & 종가
    output:
        X_train, y_train: data from 2000/1/4-2022-10-27
        X_test : data from 2022-10-28
        sc : MinMaxScaler fit to the training data
    """
    # train, test set 생성
    ts_train = all_data[:'2022-10-27'].iloc[:,0:1].values
    ts_test = all_data['2022-10-28'].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])
        y_train.append(ts_train_scaled[i+1: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"][:'2022-10-27'], all_data["종가"]['2022-10-28':]), 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
```

In [81]: X_train, y_train, X_test, sc = ts_train_test_normalize(all_data, 5, 1)
X_train.shape[0], X_train.shape[1]

Out[81]: (5627, 5)

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(2) 10.28 종가 예측

- GRU 모델 생성

```
In [83]: def GRU_model(X_train, y_train, X_test, sc):
    # 모델 생성
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, 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))

    # compile
    my_GRU_model.compile(optimizer = SGD(lr = 0.01, decay = 1e-7,
                                         momentum = 0.9, nesterov = False),
                          loss = 'mean_squared_error')

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

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

    return my_GRU_model, GRU_prediction
```

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(2) 10.28 종가 예측

- GRU 모델 훈련 이후 예측 진행
- y_pred_gru의 0번째 값이 28일에 대한 예측값

코스피 200 28일 종가 : 292.4195861816406

```
In [84]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
```

1/1 [=====] - 1s 1s/step

```
Out[84]: array([223.2838, 219.13303,
   [145.45253, 139.02568]], dtype=float32)
```

```
In [85]: y_pred_gru = pd.DataFrame(GRU_prediction[:, 0])
y_test_gru = all_data.loc['2022-10-28', '종가'][0:len(GRU_prediction)]
y_test_gru.reset_index(drop=True, inplace=True)
```

```
In [86]: y_test_gru
```

```
Out[86]: 0    0.00000
1    0.00000
2    0.00000
Name: 종가, dtype: float64
```

```
In [87]: y_pred_gru
```

```
Out[87]: 0
         0
0  292.41959
1  223.28380
2  145.45253
```

```
In [88]: np.float_(np.float_(y_pred_gru.values.tolist()[0]))
```

```
Out[88]: 292.4195861816406
```

10월 28일 예측값

```
In [89]: print(np.float_(np.float_(y_pred_gru.values.tolist()[0])))
```

292.4195861816406

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(2) 10.28 종가 예측

- 해당 종가를 0으로 지정된 28일 종가에 입력

```
In [90]: all_data.loc['2022-10-28']['종가'] = np.float_(np.float_(y_pred_gru.values.tolist())[0])
```

```
In [91]: all_data.tail(10)
```

```
Out[91]:
```

	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX
일자									
2022-10-20	288.61000	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262
2022-10-21	288.57000	-0.04000	-0.00010	102369.00000	4677515.00000	54.53237	0.10407	80133678.00000	28.37462
2022-10-24	291.47000	2.90000	0.01000	116176.00000	5372981.00000	58.28677	0.83723	80249854.00000	26.96330
2022-10-25	291.58000	0.11000	0.00040	116351.00000	5506859.00000	50.95743	0.10987	80366205.00000	25.65280
2022-10-26	293.85000	2.27000	0.00780	120348.00000	5804827.00000	49.08677	1.18453	80486553.00000	24.11615
2022-10-27	291.69028	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-28	292.41959	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-31	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-01	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-02	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(3) 10.31 종가 예측

- 과거 5일의 데이터를 이용해 1일을 예측하도록 하는 데이터 변환(31일을 예측하도록 train, test 지정)
- 예측된 28일의 종가가 입력된 데이터셋을 사용

```
10월 31일 예측

In [92]: def ts_train_test_normalize(all_data, time_steps, for_periods):
    """
    input:
        data: dataframe의 일자 & 종가
    output:
        X_train, y_train: data from 2000/1/4-2022-10-28
        X_test: data from 2022-10-31
    """
    sc = MinMaxScaler()
    sc.fit(all_data)

    # train, test set 생성
    ts_train = all_data['2022-10-28'].iloc[:,0:1].values
    ts_test = all_data['2022-10-31'].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])
        y_train.append(ts_train_scaled[i:i+for_periods])
    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"][:'2022-10-28'], all_data["종가"]['2022-10-31':]), 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])

    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

In [93]: X_train, y_train, X_test, sc = ts_train_test_normalize(all_data, 5, 1)
X_train.shape[0], X_train.shape[1]

Out[93]: (5628, 5)
```

* 코스피200 데이터 기준이며 다른 데이터들도 같은 방법을 사용해 예측

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(3) 10.31 종가 예측

- GRU 모델 생성

```
In [95]: def GRU_model(X_train, y_train, X_test, sc):
    # 모델 생성
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, 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))

    # compile
    my_GRU_model.compile(optimizer = SGD(lr = 0.01, decay = 1e-7,
                                         momentum = 0.9, nesterov = False),
                          loss = 'mean_squared_error')

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

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

    return my_GRU_model, GRU_prediction
```

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(3) 10.31 종가 예측

- GRU 모델 훈련 이후 예측 진행
 - y_pred_gru의 0번째 값이 31일에 대한 예측값
- 코스피 200 31일 종가 : 293.0155029296875**

```
In [96]: my_GRU_model, GRU_prediction = GRU_model(X_train, y_train, X_test, sc)
GRU_prediction[1:10]
```

1/1 [=====] - 2s 2s/step

```
Out[96]: array([[226.11697, 230.61778]], dtype=float32)
```

```
In [97]: y_pred_gru = pd.DataFrame(GRU_prediction[:, 0])
y_test_gru = all_data.loc['2022-10-31', '종가'][0:len(GRU_prediction)]
y_test_gru.reset_index(drop=True, inplace=True)
```

```
In [98]: y_test_gru
```

```
Out[98]: 0    0.00000
          1    0.00000
Name: 종가, dtype: float64
```

```
In [99]: y_pred_gru
```

```
Out[99]:
```

0
0 293.01550
1 226.11697

```
In [100]: np.float_(np.float_(y_pred_gru.values.tolist()[0]))
```

```
Out[100]: 293.0155029296875
```

10월 31일 예측값

```
In [101]: print(np.float_(np.float_(y_pred_gru.values.tolist()[0])))
```

```
293.0155029296875
```

수익률 예측 - 최종 예측

01. 최종 모델 학습(GRU 모델)

(3) 10.31 종가 예측

- 해당 종가를 0으로 지정된 31일 종가에 입력

```
In [102]: all_data.loc['2022-10-31']['종가'] = np.float_(np.float_(y_pred_gru.values.tolist()[0]))
```

```
In [103]: all_data.tail(10)
```

Out[103]:

일자	종가	대비	수익률	거래량	거래대금	UO	ROC	OBV	ADX
2022-10-20	288.61000	-2.68000	-0.00920	136904.00000	6043941.00000	56.07018	2.57677	80236047.00000	29.03262
2022-10-21	288.57000	-0.04000	-0.00010	102369.00000	4677515.00000	54.53237	0.10407	80133678.00000	28.37462
2022-10-24	291.47000	2.90000	0.01000	116176.00000	5372981.00000	58.28677	0.83723	80249854.00000	26.96330
2022-10-25	291.58000	0.11000	0.00040	116351.00000	5506859.00000	50.95743	0.10987	80366205.00000	25.65280
2022-10-26	293.85000	2.27000	0.00780	120348.00000	5804827.00000	49.08677	1.18453	80486553.00000	24.11615
2022-10-27	291.69028	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-28	292.41959	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-10-31	293.01550	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-01	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2022-11-02	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

수익률 예측 - 최종 예측

02. 종목별 종가 예측 결과

(1) 코스피 200

```
In [105]: result = all_data['종가'].copy()  
result.to_frame()
```

Out[105]:

일자	종가
2000-01-04	133.66000
2000-01-05	123.86000
2000-01-06	120.80000
2000-01-07	119.10000
2000-01-10	124.11000
...	...
2022-10-27	291.69028
2022-10-28	292.41959
2022-10-31	293.01550
2022-11-01	0.00000
2022-11-02	0.00000

5637 rows × 1 columns

(2) 삼성전자

```
In [105]: result = all_data['종가'].copy()  
result.to_frame()
```

Out[105]:

일자	종가
2000-01-04	6110.00000
2000-01-05	5580.00000
2000-01-06	5620.00000
2000-01-07	5540.00000
2000-01-10	5770.00000
...	...
2022-10-27	57652.03125
2022-10-28	58058.46875
2022-10-31	58220.65625
2022-11-01	0.00000
2022-11-02	0.00000

5637 rows × 1 columns

수익률 예측 - 최종 예측

02. 종목별 종가 예측 결과

(3) SK 하이닉스

```
In [106]: result = all_data['종가'].copy()  
result.to_frame()
```

Out[106]:

일자	종가
2000-01-04	505463.00000
2000-01-05	453444.00000
2000-01-06	429889.00000
2000-01-07	425963.00000
2000-01-10	422037.00000
...	...
2022-10-27	92437.64844
2022-10-28	93197.28125
2022-10-31	93669.67969
2022-11-01	0.00000
2022-11-02	0.00000

5637 rows × 1 columns

```
In [105]: result = all_data['종가'].copy()  
result.to_frame()
```

Out[105]:

일자	종가
2002-01-02	3256.00000
2002-01-03	3421.00000
2002-01-04	3491.00000
2002-01-07	3391.00000
2002-01-08	3487.00000
...	...
2022-10-27	138772.37500
2022-10-28	139478.42188
2022-10-31	140127.15625
2022-11-01	0.00000
2022-11-02	0.00000

5150 rows × 1 columns

수익률 예측 – 최종 예측

02. 종목별 종가 예측 결과

(5) CJ ENM

```
In [105]: result = all_data['종가'].copy()  
result.to_frame()
```

Out[105]:

종가

일자

2000-01-04	127816.00000
2000-01-05	113847.00000
2000-01-06	109133.00000
2000-01-07	96910.00000
2000-01-10	99005.00000
...	...
2022-10-27	77284.70312
2022-10-28	76833.28906
2022-10-31	76224.39844
2022-11-01	0.00000
2022-11-02	0.00000

5637 rows × 1 columns

수익률 예측 - 최종 예측

03. 종목별 수익률 계산 과정

(1) 코스피 200

- 종가를 아래로 한 칸 shift한 '전일 종가' column 생성
- 이후 '(당일 종가 – 전일 종가) / 전일 종가'를 통해 '수익률' 생성

```
In [110]: result_df = result_df.set_index(result.index)
```

```
In [113]: result_df['수익률'] = (result_df['종가']-result_df['전일 종가']) / result_df['전일 종가']
result_df.tail(10)
```

```
Out[113]:
```

일자	종가	전일 종가	수익률
2022-10-20	288.61000	291.29000	-0.00920
2022-10-21	288.57000	288.61000	-0.00014
2022-10-24	291.47000	288.57000	0.01005
2022-10-25	291.58000	291.47000	0.00038
2022-10-26	293.85000	291.58000	0.00779
2022-10-27	291.69028	293.85000	-0.00735
2022-10-28	292.41959	291.69028	0.00250
2022-10-31	293.01550	292.41959	0.00204
2022-11-01	0.00000	293.01550	-1.00000
2022-11-02	0.00000	0.00000	Nan

```
In [114]: result_df.loc['2022-10-27':'2022-10-31']
```

```
Out[114]:
```

일자	종가	전일 종가	수익률
2022-10-27	291.69028	293.85000	-0.00735
2022-10-28	292.41959	291.69028	0.00250
2022-10-31	293.01550	292.41959	0.00204

```
In [107]: result_df = pd.DataFrame(result.values.tolist(), columns = ['종가'])
```

```
In [108]: result_df['전일 종가'] = result_df['종가'].shift(1)
result_df.set_index(result.index)
```

```
Out[108]:
```

일자	종가	전일 종가
2000-01-04	133.66000	Nan
2000-01-05	123.86000	133.66000
2000-01-06	120.80000	123.86000
2000-01-07	119.10000	120.80000
2000-01-10	124.11000	119.10000
...
2022-10-27	291.69028	293.85000
2022-10-28	292.41959	291.69028
2022-10-31	293.01550	292.41959
2022-11-01	0.00000	293.01550
2022-11-02	0.00000	0.00000

5637 rows × 2 columns

```
In [109]: result_df
```

```
Out[109]:
```

일자	종가	전일 종가
0	133.66000	Nan
1	123.86000	133.66000
2	120.80000	123.86000
3	119.10000	120.80000
4	124.11000	119.10000
...
5632	291.69028	293.85000
5633	292.41959	291.69028
5634	293.01550	292.41959
5635	0.00000	293.01550
5636	0.00000	0.00000

5637 rows × 2 columns

수익률 예측 - 최종 예측

03. 종목별 수익률 계산 과정

(2) 삼성전자

```
In [110]: result_df = result_df.set_index(result.index)

In [113]: result_df['수익률'] = (result_df['종가']-result_df['전일 종가']) / result_df['전일 종가']
result_df.tail(10)
```

Out[113]:

일자	종가	전일 종가	수익률
2022-10-20	55500.00000	55800.00000	-0.00538
2022-10-21	55900.00000	55500.00000	0.00721
2022-10-24	57600.00000	55900.00000	0.02862
2022-10-25	57700.00000	57500.00000	0.00348
2022-10-26	59400.00000	57700.00000	0.02946
2022-10-27	57652.03125	59400.00000	-0.02943
2022-10-28	58058.46875	57652.03125	0.00705
2022-10-31	58220.65625	58058.46875	0.00279
2022-11-01	0.00000	58220.65625	-1.00000
2022-11-02	0.00000	0.00000	NaN

```
In [114]: result_df.loc['2022-10-27':'2022-10-31']
```

Out[114]:

일자	종가	전일 종가	수익률
2022-10-27	57652.03125	59400.00000	-0.02943
2022-10-28	58058.46875	57652.03125	0.00705
2022-10-31	58220.65625	58058.46875	0.00279

(3) SK 하이닉스

```
In [111]: result_df = result_df.set_index(result.index)

In [114]: result_df['수익률'] = (result_df['종가']-result_df['전일 종가']) / result_df['전일 종가']
result_df.tail(10)
```

Out[114]:

일자	종가	전일 종가	수익률
2022-10-20	90200.00000	92900.00000	-0.02906
2022-10-21	90500.00000	90200.00000	0.00333
2022-10-24	91800.00000	90500.00000	0.01436
2022-10-25	93500.00000	91800.00000	0.01852
2022-10-26	93900.00000	93500.00000	0.00428
2022-10-27	92437.64844	93900.00000	-0.01557
2022-10-28	93197.28125	92437.64844	0.00822
2022-10-31	93669.67969	93197.28125	0.00507
2022-11-01	0.00000	93669.67969	-1.00000
2022-11-02	0.00000	0.00000	NaN

```
In [115]: result_df.loc['2022-10-27':'2022-10-31']
```

Out[115]:

일자	종가	전일 종가	수익률
2022-10-27	92437.64844	93900.00000	-0.01557
2022-10-28	93197.28125	92437.64844	0.00822
2022-10-31	93669.67969	93197.28125	0.00507

수익률 예측 - 최종 예측

03. 종목별 수익률 계산 과정

(4) 리노공업

```
In [110]: result_df = result_df.set_index(result.index)

In [113]: result_df['수익률'] = (result_df['종가']-result_df['전일 종가']) / result_df['전일 종가']
result_df.tail(10)
```

Out[113]:

일자	종가	전일 종가	수익률
2022-10-20	133000.00000	135800.00000	-0.02062
2022-10-21	135100.00000	133000.00000	0.01579
2022-10-24	140000.00000	135100.00000	0.03627
2022-10-25	139700.00000	140000.00000	-0.00214
2022-10-26	139000.00000	139700.00000	-0.00501
2022-10-27	138772.37500	139000.00000	-0.00164
2022-10-28	139478.42188	138772.37500	0.00509
2022-10-31	140127.15625	139478.42188	0.00465
2022-11-01	0.00000	140127.15625	-1.00000
2022-11-02	0.00000	0.00000	NaN

```
In [114]: result_df.loc['2022-10-27':'2022-10-31']
```

Out[114]:

일자	종가	전일 종가	수익률
2022-10-27	138772.37500	139000.00000	-0.00164
2022-10-28	139478.42188	138772.37500	0.00509
2022-10-31	140127.15625	139478.42188	0.00465

```
In [110]: result_df = result_df.set_index(result.index)
```

```
In [113]: result_df['수익률'] = (result_df['종가']-result_df['전일 종가']) / result_df['전일 종가']
result_df.tail(10)
```

Out[113]:

일자	종가	전일 종가	수익률
2022-10-20	79700.00000	78800.00000	0.01142
2022-10-21	78800.00000	79700.00000	-0.01129
2022-10-24	80400.00000	78800.00000	0.02030
2022-10-25	76700.00000	80400.00000	-0.04602
2022-10-26	74000.00000	76700.00000	-0.03520
2022-10-27	77284.70312	74000.00000	0.04439
2022-10-28	76833.28906	77284.70312	-0.00584
2022-10-31	76224.39844	76833.28906	-0.00792
2022-11-01	0.00000	76224.39844	-1.00000
2022-11-02	0.00000	0.00000	NaN

```
In [114]: result_df.loc['2022-10-27':'2022-10-31']
```

Out[114]:

일자	종가	전일 종가	수익률
2022-10-27	77284.70312	74000.00000	0.04439
2022-10-28	76833.28906	77284.70312	-0.00584
2022-10-31	76224.39844	76833.28906	-0.00792

수익률 예측 - 최종 예측

04. 최종 결과물

구분	종목명	종목코드	10월 27일 (목) 수익률	10월 28일 (금) 수익률	10월 31일 (월) 수익률
지수	코스피200	KS200	-0.00735	0.0025	0.00204
코스피 종목 (1)	삼성전자	005930	-0.02943	0.00705	0.00279
코스피 종목 (2)	SK하이닉스	000660	-0.01557	0.00822	0.00507
코스닥 종목 (1)	리노공업	058470	-0.00164	0.00509	0.00465
코스닥 종목 (2)	CJ ENM	035760	0.04439	-0.00584	-0.00792

다만, '수익률 예측 제출 양식.csv' 종목코드에서 앞에 0이 사라지는 문제 해결하지 못함

모델을 돌릴 때마다 정말 근소한 차이가 있을 수 있음

부록

01. data

(1) 코스피200(990601_221021).csv

- ML, DL 모델 비교를 위해 사용했던 데이터

(2) '_221026.csv'로 끝나는 코스피200, 삼성전자, SK하이닉스, 리노공업, CJENM ~

- 최종 예측 모델에 사용된 데이터

02. code

(1) 코스피200_ML 비교.ipynb, 코스피200_LSTM,GRU 비교.ipynb

- ML, DL 모델 비교를 위해 사용했던 코드

(2) '_221021'로 끝나는 코스피200, 삼성전자, SK하이닉스, 리노공업, CJENM

- 최종 예측 모델을 담은 코드

감사합니다