응용통계학과 분석 공모전 보고서

팀명	통테이토
조원	김가영, 남화승, 이가영, 이나현
주제	Linear Regression 을 통한 투자심리지수 제작

목차

1. 프로젝트 목적 및 개요

1) 프로젝트 목적

투자심리지수란, 투자심리의 변화를 일정기간 동안 파악하여 장세의 과열도를 측정하기 위한 기술적 분석의 한 지표이다. '투자심리지수의 대용변수와 유용성-김가람, 류두진(2018)' 등에 의하면, 투자자 심리란 투자자의 공통된 판단오류로 인해 가격을 잘못 측정하는 현상을 의미한다. 또한 같은 논문에 따르면 개인투자자의 매수- 매도 거래량이 투자심리지수의 대용변수로써 활용될 수 있으므로, 종속변수(y)로써 일별 개인 매수-매도 거래량 불균형을 사용하여 투자심리지수를 구축하고자 한다. 한편, 'Investor Sentiment and the Cross-Section of Stock Returns-Baker and Wurgler(2006)'에 의하면 다변수 투자심리지수의 설명력이 높다는 점에 착안하여, 매수매도 불균형에 대해 여러가지 경제및 금융 변수들로 회귀 분석을 진행함으로써 투자심리지수를 구축해보고자 한다.

2) 프로젝트 개요

(1) 종속변수 선택

KOSPI 를 기준으로, 2018 년부터 2022 년 4 분기까지 총 5 년의 데이터를 사용했다. 그리고 종속변수로 개인투자자의 매수-매도 거래량 불균형(IBSI)을 채택했다. 'The Behavior of Individual Investors-Barber et al.(2009)'에 의하면, 개인투자자의 거래행태는 체계적이며 군집 행동을 보인다. 이러한 거래가 정보를 기반으로 발생하는 합리적이고 이성적인 움직임이 아닌, 개인투자자들의 심리적 편의와 비이성성으로 인해 발생하기 때문에 투자자 심리의 대용변수로 개인투자자의 매수-매도 거래량 불균형을 제안했다. 또한, '개인투자자의 투자심리와 주식수익률-강장구. 권경윤. 심명화(2013)'에 따르면, 국내 주식시장에서 투자자 심리로 인해 과소평가되거나 과대평가되었던 주식 이 시간이 지남에 따라 원래 가치로 돌아오는 회귀(reversal) 현상이 발생한다고 주장하였다. 이에 투자심리지수의 대용변수로써, 분석의 종속변수로써 개인투자자의 매수-매도 거래량 불균형이 적합하다고 여겨 채택하였다.

$$\mathrm{IBSIi,t} = \frac{\mathrm{BVi,t} - \mathrm{SVi,t}}{\mathrm{BVi,t} + \mathrm{SVi,t}}$$

위 식은 t 시점에서의 i 기업의 개인투자자 매수-매도 거래량 불균형의 산출십이다. BVi,t 는 개인투자자의 매수 거래량, IVi,t 는 개인 투자자의 매도 거래량을 의미한다. IBSI 가 0 보다 크다면 주식의 수요가 과잉되었고, 즉 이는 투자자의 심리가 긍정적인 것으로 판단한다. 반대로 IBSI 가 0 보다 작다면 주식의 공급이 과잉되었고, 투자자 심리가 부정적임을 가정하였다.

(2) 독립변수 선택

이어서 독립변수로 아래와 같은 지표들을 채택하였다. 모든 변수는 분기별로 시기를 통일해 진행하였다.

a. CPI

: 소비자 물가지수이다. 거시 경제 지표로, 경기 전망을 판단할 수 있다.

b. CD(Certificate of Deposit)

: 주식투자 예치금이다. 투자자 예탁금(장내파생상품 거래 예수금 제외)을 주식투자 예치금으로 사용한다. 추세보다 많은 금액이 예치되어 있을수록 투자자들은 낙관적인 심리를 보인다.

c. 기준금리

: 거시 경제 지표로 시장의 유동성을 결정하는 기준이다. 한국은행 기준금리만을 적용하여 분석에 활용하였다.

d. 소비자심리지수

: 현재생활형편, 생활형편전망 등 6 개 영역을 지수화한 지표이다. 경제에 대한 소비자들의 전반전인 인식을 파악할 수 있다.

e. VKOSPI

: 코스피 변동성 지수이다. 시장의 변동성 수준에 따라 시장 참여자들의 기대 심리가 달라진다.

f. 거래량

: 한국 주식시장의 거래량을 파악하기 위해 코스피 시장 내 일별 전체 거래량 데이터를 활용하였다. 투자자들의 과열 수준을 판단할 수 있다.

g. 상장주식 거래회전율

: 시장 내 유동성을 판단할 수 있다. 회전율이 높을수록 거래가 활발하며 투기적 성향이 높음을 의미한다.

h. KOSPI MA120

: 코스피 지수에 대한 120 일 이동평균선(장기 추세선)을 산출해 변수에 추가하였다. 시장의 거시적인 추세를 파악할 수 있다.

i. 환율

: 원/달러 기준 환율을 채택하였다. 환율에 따라 해외투자자의 시장 참여가 달라진다. 또한 경기상황과 상관관계가 있어 투자자 심리와 연관성이 존재할 것으로 기대한다.

2. EDA 및 데이터 전처리

1) EDA

```
df.set_index("Datetime", inplace=True)
df = df.bfill()
df
```

Datetime	cpi	cd	cons_sent	exchange	base	rt_rate	vkospi	volume	kospi_ma120	target
2018-01-02	98.106	27321579	111.0	1061.2	1.50	19.34	12.66	262205	2438.634831	-0.012518
2018-01-03	98.106	29106445	111.0	1064.5	1.50	19.34	12.64	331095	2439.400415	0.004646
2018-01-04	98.106	26859408	111.0	1062.2	1.50	19.34	12.22	333836	2440.116581	-0.000972
2018-01-05	98.106	27121167	111.0	1062.7	1.50	19.34	11.92	308770	2441.026331	-0.025804
2018-01-08	98.106	26730598	111.0	1066.0	1.50	19.34	12.31	311429	2442.071914	-0.013039
...	...	...	...	...	...	...	...	...	...	...
2022-12-23	109.280	43902495	90.2	1280.8	3.25	13.31	17.43	366989	2371.032672	-0.003814
2022-12-26	109.280	47542604	90.2	1274.8	3.25	13.31	17.46	427845	2371.172671	-0.005889
2022-12-27	109.280	45718241	90.2	1271.4	3.25	13.31	17.09	448499	2371.097754	-0.057144
2022-12-28	109.280	46984411	90.2	1267.0	3.25	13.31	17.53	405894	2371.001420	0.039391
2022-12-29	109.280	47046503	90.2	1264.5	3.25	13.31	18.40	361194	2370.185836	0.053954

1265 rows × 10 columns

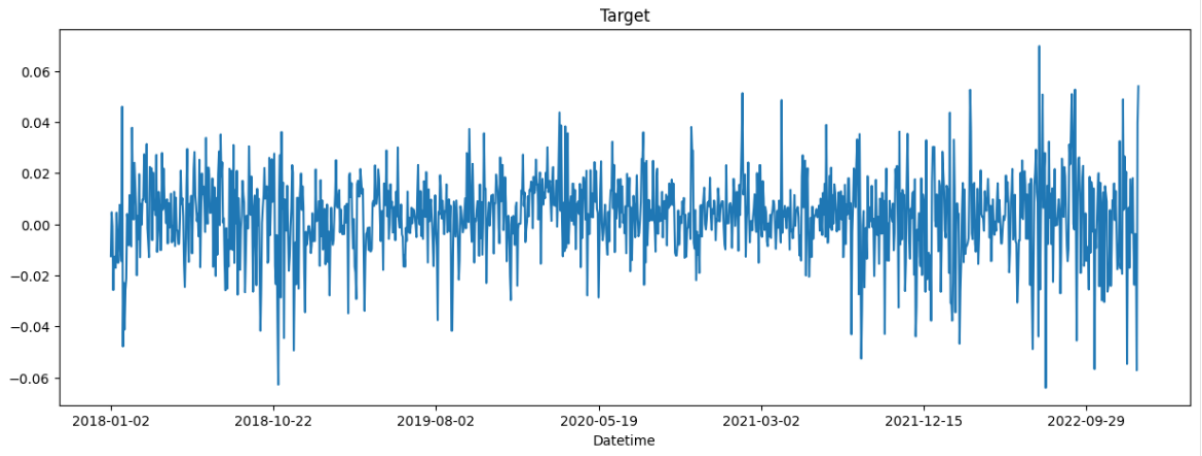
각 변수들 간의 다중공선성을 파악하는 코드와 결과는 다음과 같다.

```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
# VIF 계산
def calculate_vif(dataframe, dependent_column):
    X = dataframe.drop(columns=[dependent_column])
    columns = X.columns
    vif_data = pd.DataFrame()
    vif_data["Variable"] = columns
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif_data

# 다중공선성 확인
vif_df = calculate_vif(df, dependent_column='target')
print(vif_df)
```

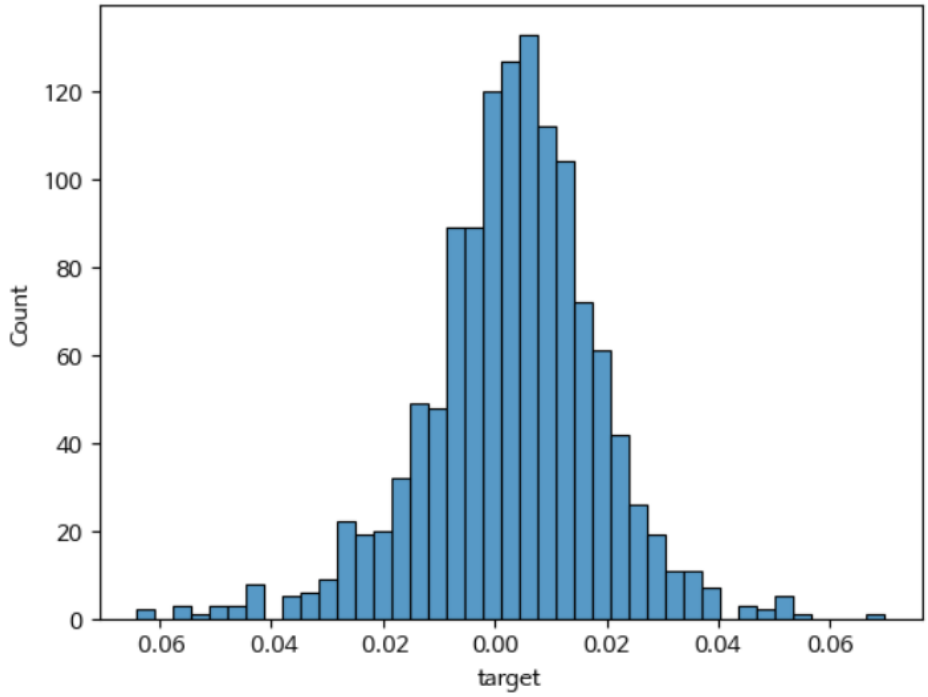
	Variable	VIF
0	cpi	3189.853711
1	cd	66.226706
2	cons_sent	640.415340
3	exchange	1508.489884
4	base	14.766145
5	rt_rate	28.922478
6	vkospi	18.347755
7	volume	17.592667
8	kospi_ma120	338.940758

경제지표 특성상 변수들 간의 다중공선성이 매우 높게 나타난 모습을 볼 수 있다. 다중공선성 문제를 해결하기 위해 Step Wise selection, AIC, BIC 의 변수 선택법을 진행하였다.

```
_ = df.target.plot(style = '-', figsize = (15, 5), title = "Target")
plt.rc('font', family='NanumBarunGothic')
```



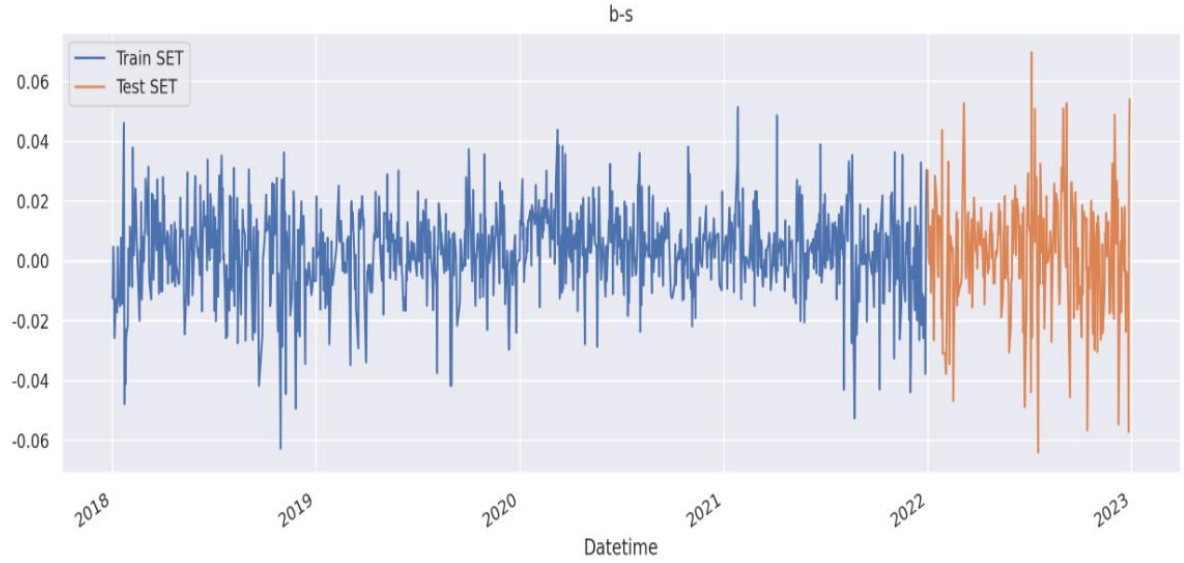
```
sns.histplot(df.target) # y ~ Normal
plt.rc('font', family='NanumBarunGothic')
```



2) PREPROCESSING DATA

2022-01-01 을 기준으로 train-test split 을 진행하였다.(80% train data & 20% test data)

```
split_date = '01-Jan-2022'
df_train = df.loc[df.index < split_date].copy()
df_test = df.loc[df.index >= split_date].copy()

df_train.iloc[:,-1:].rename(columns = {'target': 'Train SET'})₩
    .join(df_test.iloc[:,-1:].rename(columns={'target': 'Test SET'}),
         how = 'outer').plot(figsize = (15,5), title = 'b-s', style = '-')

X_train, y_train = df_train.iloc[:, :-1], df_train.iloc[:, -1:]
X_test, y_test = df_test.iloc[:, :-1], df_test.iloc[:, -1:]
plt.rc('font', family='NanumBarunGothic')
```

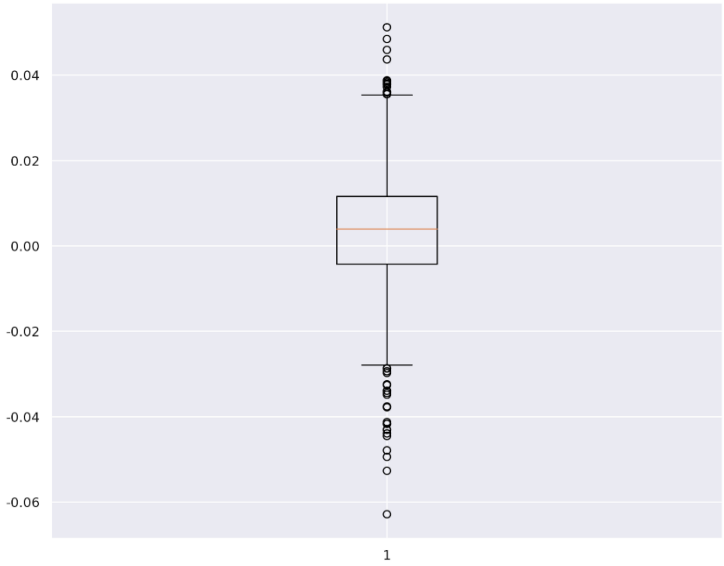


스케일링 방법 중, standard scaling 을 채택해 진행하였다.

```
ss= StandardScaler()

df_train_scaled = pd.DataFrame(ss.fit_transform(df_train), index = df_train.index, columns = df_train.columns)
df_test_scaled = pd.DataFrame(ss.transform(df_test), index = df_test.index, columns = df_test.columns)

X_train, y_train = df_train.iloc[:, :-1], df_train.iloc[:, -1:]
X_test, y_test = df_test.iloc[:, :-1], df_test.iloc[:, -1:]

X_train_scaled, y_train_scaled = df_train_scaled.iloc[:, :-1], df_train_scaled.iloc[:, -1:]
X_test_scaled, y_test_scaled = df_test_scaled.iloc[:, :-1], df_test_scaled.iloc[:, -1:]
```

이상치 제거과정은 아래와 같다.

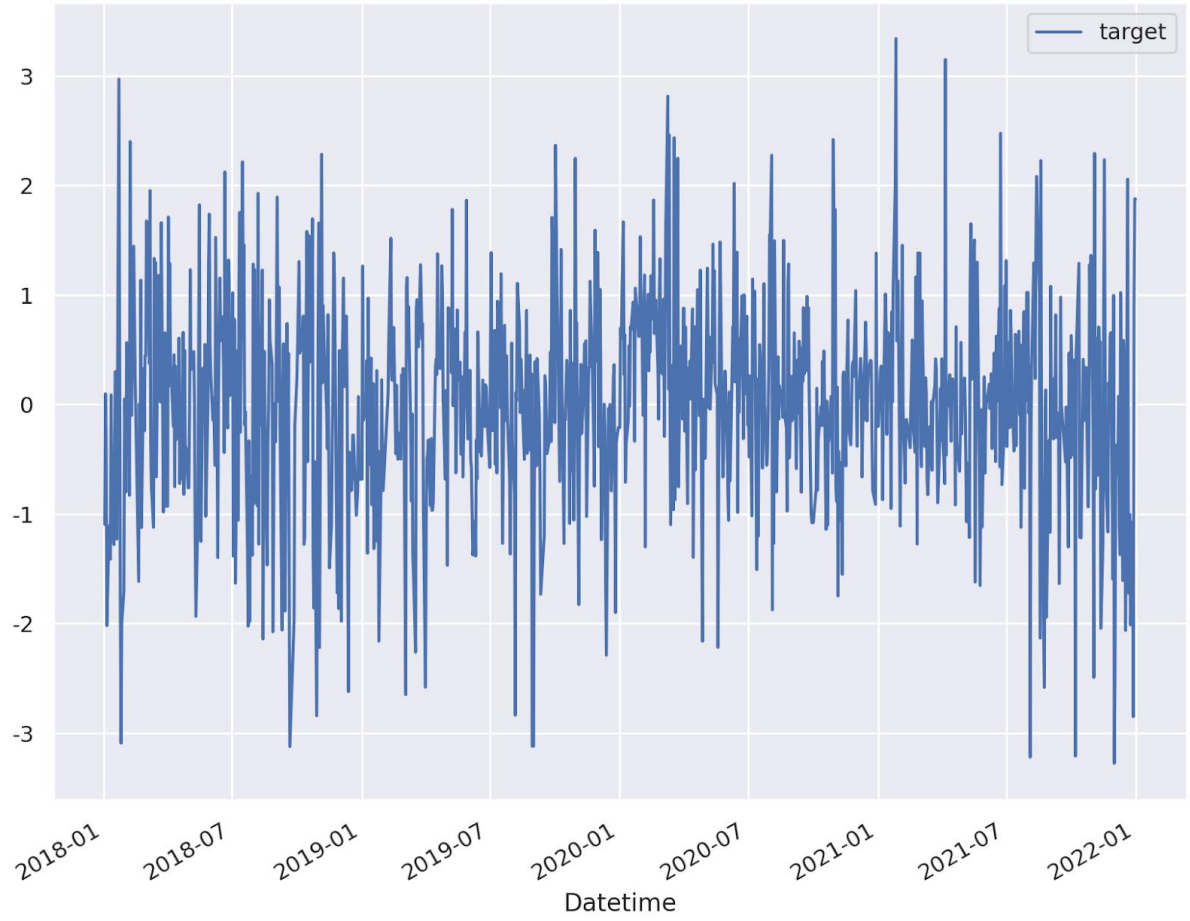
```
plt.boxplot(y_train)
plt.show()
```




```
q1 = y_train.quantile(0.25)[0]
q3 = y_train.quantile(0.75)[0]

IQR = q3 - q1


rev_range = 2.5
remove_index = (y_train <= q3 + rev_range* IQR) & (y_train >= q1 - rev_range * IQR)

X_train = X_train.loc[remove_index.target]
y_train = y_train.loc[remove_index.target]
```

```
q1 = y_train_scaled.quantile(0.25)[0]
q3 = y_train_scaled.quantile(0.75)[0]

IQR = q3 - q1


rev_range = 2.5
remove_index = (y_train_scaled <= q3 + rev_range* IQR) & (y_train_scaled >= q1 - rev_range * IQR)

X_train_scaled = X_train_scaled.loc[remove_index.target]
y_train_scaled = y_train_scaled.loc[remove_index.target]

y_train_scaled.plot()
```

데이터의 IQR 을 계산해, IQR 의 2.5 배를 이상치 제거 범위로 설정하였다. y train 데이터를 정규화해 q3-q1 로 이상치를 제거했다. 아래는 이상치가 제거된 정규화된 y_train_scaled 데이터의 시각화이다.



3. 변수 선택

회귀분석 과목에서 배웠듯이, 회귀분석의 목적은 회귀방정식을 이용하여 독립변수와 종속변수 간에 존재하는 복잡한 상호작용을 잘 모형화하는 것이다. 즉, 되도록 많은 독립변수를 모형에 포함하여 회귀방정식을 정교하게 만들어야한다. 이를 위해서, goodness of fit 의 관점과 principle of parsimony(simplicity of model)의 관점을 잘 조화시켜 한다. 설명변수가 많아질 수록 적합도는 올라가지만, 모형
의 간단성은 저하된다. 반대로, 설명변수가 적을수록 적합도는 떨어지지만, 모형은 더욱 간단해진다. 우리는 서로 상충되는 두 관점을 잘 조화시켜 적절한 변수선택을 함으로써 회귀계수 및 반응변수에 대한 추정치의 정확도를 높이고자 한다.

1) Best Selection

변수 선택을 위해 사용되는 통계량은 크게 5 가지로 분류되지만, 우리는 그 중 AIC 와 F-statistic 을 통해 모든 경우의 수를 판단한 뒤 Best Model 을 설정해보았다.

a. AIC

AIC 를 구하기 위해 다음과 같이 함수를 설정한다. 앞서 설명한 것과 같이, 모든 경우의 수를 판단하기 위해 for 문을 통해 변수들의 조합을 계산하였다. 데이터 컬럼, 즉 변수들의 가능한 조합을 *combo* 로 설정했고, 이를 *variables* 리스트에 추가하였다. *model* 에는 ols 모델을, *aic* 에는 OLS 모델의 AIC 를 appending 하였다. AIC 값은 작을수록 좋은 변수선택을 뜻하는 것이기에 *argmin* 을 통해 AIC 값이 작은 것을 골라냈다. *variables* 중 AIC 가 작은 변수 조합을 채택하고, 해당 변수 조합으로 구성된 *model* 을 return 한다.

```
def best_lm_aic(df):
    result = {'variables':[], 'model':[], 'aic':[]}
    for k in range(1, len(df.columns)-1):

        for combo in itertools.combinations(df.columns[:-1], k):
            lm = ols('target~' + '+'.join(combo), data = df).fit()
            result['variables'].append(combo)
            result['model'].append(lm)
            result['aic'].append(lm.aic)

    min_arg = np.argmin(result['aic'])
    print(f"Best variable selection : {result['variables'][min_arg]}")
    return result['model'][min_arg]
```

이렇게 정의한 함수에 앞서 전처리한 데이터를 넣으면 다음과 같은 결과가 나온다.

```
best_lm = best_lm_aic(df_train_scaled)
best_lm.summary()
```

```
Best variable selection : ('cons_sent', 'exchange', 'rt_rate', 'vkospi', 'kospi_ma120')
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.117e-18	0.031	-3.59e-17	1.000	-0.061	0.061
cons_sent	0.1484	0.064	2.315	0.021	0.023	0.274
exchange	0.0743	0.042	1.788	0.074	-0.007	0.156
rt_rate	0.1115	0.040	2.780	0.006	0.033	0.190
vkospi	0.1222	0.048	2.559	0.011	0.029	0.216
kospi_ma120	-0.1220	0.045	-2.741	0.006	-0.209	-0.035

Omnibus:	50.819	**Durbin-Watson:**	1.776
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	89.770
Skew:	-0.370	**Prob(JB):**	3.21e-20
Kurtosis:	4.257	**Cond. No.**	4.01

OLS Regression Results

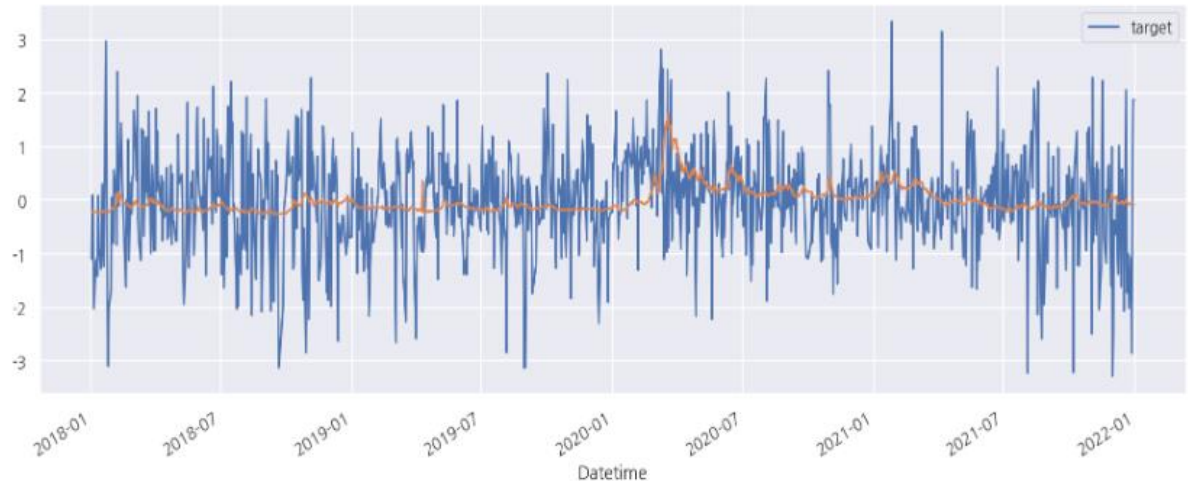
Dep. Variable:	target	**R-squared:**	0.025
Model:	OLS	**Adj. R-squared:**	0.021
Method:	Least Squares	**F-statistic:**	5.257
Date:	Mon, 17 Jul 2023	**Prob (F-statistic):**	8.98e-05
Time:	21:05:02	**Log-Likelihood:**	-1424.3
No. Observations:	1013	**AIC:**	2861.
Df Residuals:	1007	**BIC:**	2890.
Df Model:	5		
Covariance Type:	nonrobust		

환율을 제외하고 모든 변수들이 5% 유의수준 하에서 통계적으로 유의하다. 우리는 변수선택의 기준을 결정계수가 아닌 AIC 로 설정하였으므로 결정계수 값이 낮게 나온 것은 일정 부분 감안하기로 하였다.

이후 전처리한 train 데이터로 값을 예측해 plot 을 그려보았다.

```
pred = best_lm.predict(X_train_scaled)
```

```
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = y_train_scaled.plot(ax = ax, style = '-')
_ = (2*pred).plot(ax = ax, style = '-')
```



b. F-statistic

F 값 역시 AIC 와 과정이 동일하다. 변수들의 모든 가능한 조합을 for 문을 통해 계산함으로써 F 값이 가장 낮은 모델을 채택하는 함수를 정의하였다.

```
def best_lm_F(df):
    result = {'variables':[], 'model':[], 'F-value':[]}
    for k in range(1, len(df.columns)-1):

        for combo in itertools.combinations(df.columns[:-1], k):
            lm = ols('target~' + '+'.join(combo), data = df).fit()
            result['variables'].append(combo)
            result['model'].append(lm)
            result['F-value'].append(lm.fvalue)

    max_arg = np.argmax(result['F-value'])
    print(f"Best variable selection : {result['variables'][max_arg]}")
    return result['model'][max_arg]
```

전처리한 데이터를 함수에 넣은 후 나온 결과는 다음과 같다.

```
best_lm = best_lm_F(df_train_scaled)
best_lm.summary()
```

```
Best variable selection : ('vkospi',)
```

OLS Regression Results

Dep. Variable:	target	**R-squared:**	0.014
Model:	OLS	**Adj. R-squared:**	0.013
Method:	Least Squares	**F-statistic:**	14.83
Date:	Mon, 17 Jul 2023	**Prob (F-statistic):**	0.000125
Time:	21:05:08	**Log-Likelihood:**	-1430.0
No. Observations:	1013	**AIC:**	2864.
Df Residuals:	1011	**BIC:**	2874.
Df Model:	1		
Covariance Type:	nonrobust		

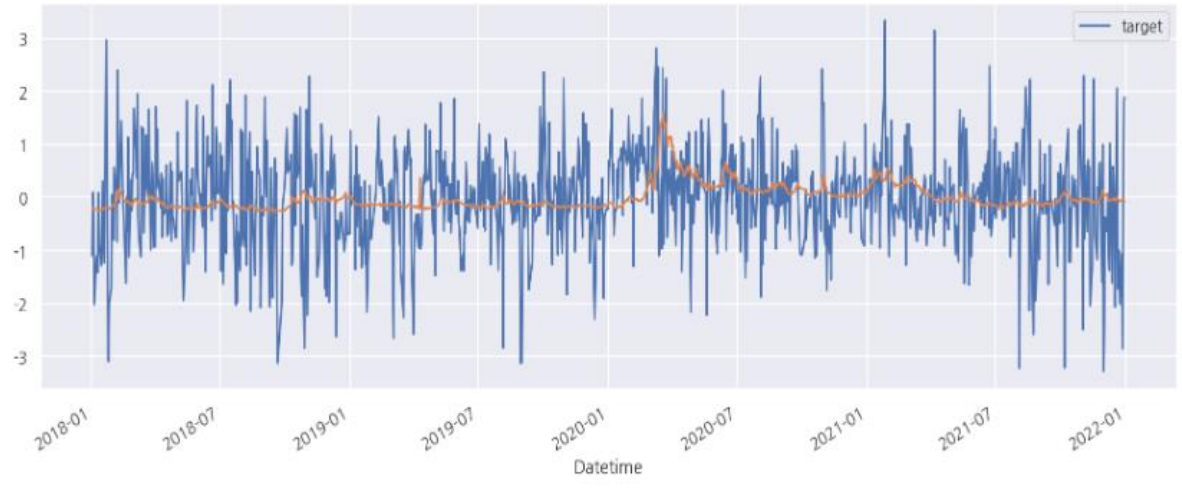
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.117e-18	0.031	-3.58e-17	1.000	-0.061	0.061
vkospi	0.1202	0.031	3.851	0.000	0.059	0.182

Omnibus:	65.631	**Durbin-Watson:**	1.761
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	117.756
Skew:	-0.457	**Prob(JB):**	2.69e-26
Kurtosis:	4.398	**Cond. No.**	1.00

vkospi 만 변수로 선택하며 p-value 는 0 이다. 해당 모델에 train data 를 적합 시킨 후 plot 를 그려보았다.

```
pred = best_lm.predict(X_train_scaled)
```

```
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = y_train_scaled.plot(ax = ax, style = '-')
_ = (2*pred).plot(ax = ax, style = '-')
```



2) Stepwise Selection

변수 선택의 3 가지 방법- 전진선택법(FS), 후진소거법(BE), 단계적 소거법(SS) - 중 전진과 후진을 결합한 모양의 단계적 소거법을 채택하여 활용했다. 이 역시 함수를 정의하여 결과를 도출했다. 이미 선택된 변수는 더 이상 검토하지 않는 FS 와 달리, SS 는 새로운 변수가 들어올 때마다 기존의 변수가 계속 유의하게 남아 있을 수 있는 지를 검토한다. 즉, 이미 선택된 변수가 새로운 변수에 의해 중요성을 상실하게 되는 지를 매단계에서 검토한다.

```
#Stepwise Selection을 위한 함수 생성

def stepwise_feature_selection(X_train, y_train, variables=X_train.columns.tolist(), sl_enter = 0.05, sl_remove = 0.05 ):
    import statsmodels.api as sm
    import matplotlib.pyplot as plt
    import warnings
    warnings.filterwarnings("ignore")

    y = y_train ## 반응 변수

    selected_variables = [] ## 선택된 변수들


    sv_per_step = [] ## 각 스텝별로 선택된 변수들
    adjusted_r_squared = [] ## 각 스텝별 수정된 결정계수
    steps = [] ## 스텝
    step = 0
    while len(variables) > 0:
        print('-'*30)
        print(f'[step : {step+1}]')
        print('-'*30)
        remainder = list(set(variables) - set(selected_variables))
        pval = pd.Series(index=remainder) ## 변수의 p-value
        ## 기존에 포함된 변수와 새로운 변수 하나씩 돌아가면서
        ## 선형 모형을 적합한다.
        for col in remainder:
            X = X_train[selected_variables+[col]]
            X = sm.add_constant(X)
            model = sm.OLS(y,X).fit(disp=0)
            pval[col] = model.pvalues[col]
        print(X)

        min_pval = pval.min()
        if min_pval < sl_enter: ## 최소 p-value 값이 기준 값보다 작으면 포함
            selected_variables.append(pval.idxmin())
            ## 선택된 변수들에대해서
            ## 어떤 변수를 제거할지 고른다.
            while len(selected_variables) > 0:
                selected_X = X_train[selected_variables]
                selected_X = sm.add_constant(selected_X)
                selected_pval = sm.OLS(y,selected_X).fit(disp=0).pvalues[1:] ## 절편항의 p-value는 뺀다
                max_pval = selected_pval.max()
                if max_pval >= sl_remove: ## 최대 p-value값이 기준값보다 크거나 같으면 제외
                    remove_variable = selected_pval.idxmax()
                    selected_variables.remove(remove_variable)

                else:
                    break

            step += 1
            steps.append(step)
            adj_r_squared = sm.OLS(y,sm.add_constant(X_train[selected_variables])).fit(disp=0).rsquared_adj
            adjusted_r_squared.append(adj_r_squared)
            sv_per_step.append(selected_variables.copy())
        else:
            break
```

```
    print(f'Selected Vairables : {selected_variables}')
    fig = plt.figure(figsize=(100,10))
    fig.set_facecolor('white')

    font_size = 15
    plt.xticks(steps,[f'step {s}₩n'+'₩n'.join(sv_per_step[i]) for i,s in enumerate(steps)], fontsize=12)
    plt.plot(steps,adjusted_r_squared, marker='o')

    plt.ylabel('Adjusted R Squared',fontsize=font_size)
    plt.grid(True)
    plt.show()

    return selected_variables
```

```
selected_variables = stepwise_feature_selection(X_train_scaled, y_train_scaled)
```

```
------------------------------
[step : 1]
------------------------------
            const  kospi_ma120
Datetime
2018-01-02    1.0     0.085292
2018-01-03    1.0     0.087302
2018-01-04    1.0     0.089182
2018-01-05    1.0     0.091570
2018-01-08    1.0     0.094316
...           ...          ...
2021-12-27    1.0     1.785336
2021-12-28    1.0     1.779364
2021-12-29    1.0     1.772540
2021-12-30    1.0     1.765808
2021-12-31    1.0     1.760044

[1008 rows x 2 columns]
------------------------------
[step : 2]
------------------------------
            const    vkospi  kospi_ma120
Datetime
2018-01-02    1.0 -0.848614     0.085292
2018-01-03    1.0 -0.851285     0.087302
2018-01-04    1.0 -0.907376     0.089182
2018-01-05    1.0 -0.947441     0.091570
2018-01-08    1.0 -0.895357     0.094316
...           ...       ...          ...
2021-12-27    1.0 -0.207571     1.785336
2021-12-28    1.0 -0.259655     1.779364
2021-12-29    1.0 -0.329102     1.772540
2021-12-30    1.0 -0.335779     1.765808
2021-12-31    1.0 -0.335779     1.760044
```

선택된 변수들로 식을 만들어 ols 모델에 적합시키면 다음과 같은 결과가 나온다.

```
fomula = 'target~'+'+'.join(selected_variables)
fomula

lm = ols(fomula, data = df_train_scaled).fit()
lm.summary()
```

OLS Regression Results

Dep. Variable:	target	**R-squared:**	0.014
Model:	OLS	**Adj. R-squared:**	0.013
Method:	Least Squares	**F-statistic:**	14.83
Date:	Mon, 17 Jul 2023	**Prob (F-statistic):**	0.000125
Time:	21:09:48	**Log-Likelihood:**	-1430.0
No. Observations:	1013	**AIC:**	2864.
Df Residuals:	1011	**BIC:**	2874.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.117e-18	0.031	-3.58e-17	1.000	-0.061	0.061
vkospi	0.1202	0.031	3.851	0.000	0.059	0.182

Omnibus:	65.631	**Durbin-Watson:**	1.761
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	117.756
Skew:	-0.457	**Prob(JB):**	2.69e-26
Kurtosis:	4.398	**Cond. No.**	1.00

F-statistic 과 마찬가지로 vkospi 만 변수로 채택하며 p-value 는 0 이다.

이렇게 3 가지 방법을 사용하여 최적의 변수선택을 찾아본 결과, AIC 를 기준으로 평가했을 때의 best model 이 가장 적절하다고 판단하였다. F-statistic 과 단계적 선택법(SS)은 vkospi 만을 변수로 채택하기 때문에 우리가 지향하는 다변수 심리지수 구축과 방향성이 맞지 않을 뿐더러, p-value 값이 0 으로 나온다는 문제점이 있어 해당 모델은 제외하였다. 따라서 AIC 에 근거한 5 개의 설명변수 - 변동성(vkospi), 상장주식 회전율(rt_rate), KOSPI MA120(kospi_ma120), 소비자 심리지수(cons_sent), 환율(exchange) - 를 채택하여 분석을 진행하였다.

4. 모델 적용

1) OLS 모델 (ordinary least squares)

선택된 5 개의 독립변수를 활용해 잔차의 제곱을 최소화하여 모델을 적합시키는 OLS 모델을 적용하였다.

a. 모델 적합

```
ols_model = ols('target~vkospi+rt_rate+kospi_ma120+cons_sent+exchange', data = df_train_scaled).fit()
```

- 결과:

OLS Regression Results

Dep. Variable:	target	R-squared:	0.025
Model:	OLS	Adj. R-squared:	0.021
Method:	Least Squares	F-statistic:	5.257
Date:	Mon, 17 Jul 2023	Prob (F-statistic):	8.98e-05
Time:	22:48:43	Log-Likelihood:	-1424.3
No. Observations:	1013	AIC:	2861.
Df Residuals:	1007	BIC:	2890.
Df Model:	5		
Covariance Type:	nonrobust		

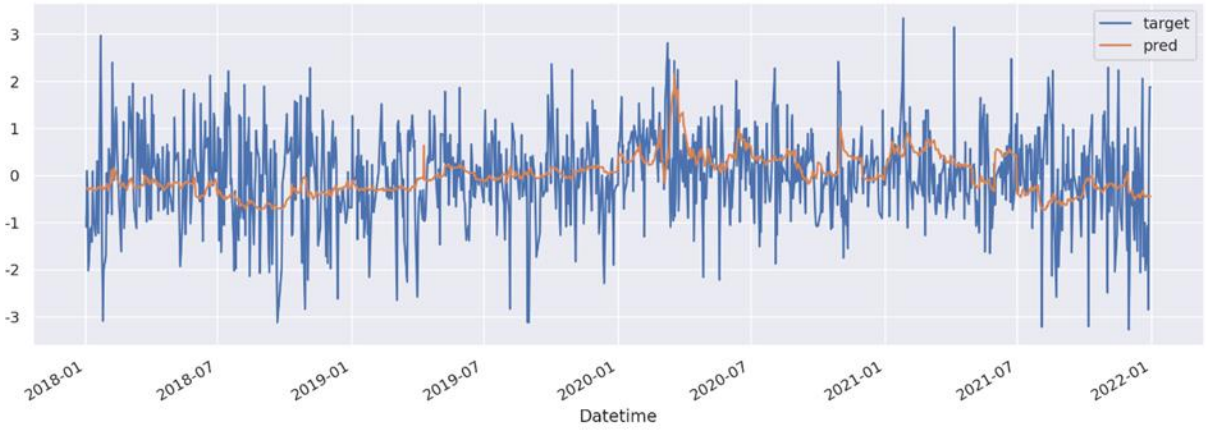
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.117e-18	0.031	-3.59e-17	1.000	-0.061	0.061
vkospi	0.1222	0.048	2.559	0.011	0.029	0.216
rt_rate	0.1115	0.040	2.780	0.006	0.033	0.190
kospi_ma120	-0.1220	0.045	-2.741	0.006	-0.209	-0.035
cons_sent	0.1484	0.064	2.315	0.021	0.023	0.274
exchange	0.0743	0.042	1.788	0.074	-0.007	0.156

Omnibus:	50.819	Durbin-Watson:	1.776
Prob(Omnibus):	0.000	Jarque-Bera (JB):	89.770
Skew:	-0.370	Prob(JB):	3.21e-20
Kurtosis:	4.257	Cond. No.	4.01

결과 해석: 해당 OLS 모델은 R-squared 값이 0.025로, 종속 변수에 대한 설명력이 낮아서 모델이 데이터를 상대적으로 잘 설명하지 못한다고 해석할 수 있다. P-value는 8.98e-05로, 모델이 통계적으로 유의미함을 알 수 있다. Omnibus와 Jarque-Bera는 0.000으로 정규성을 충족하지 않음을 알 수 있다.

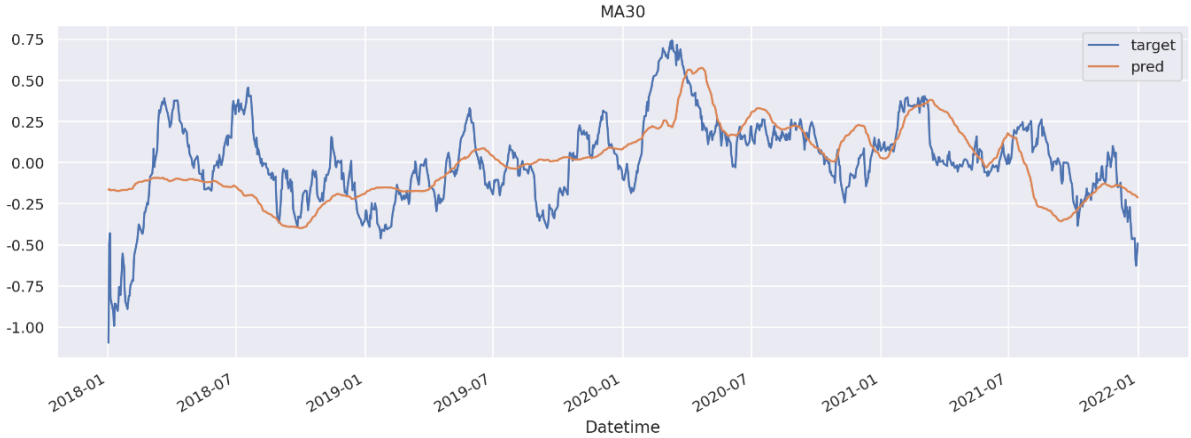
b. 모델 예측 및 시각화

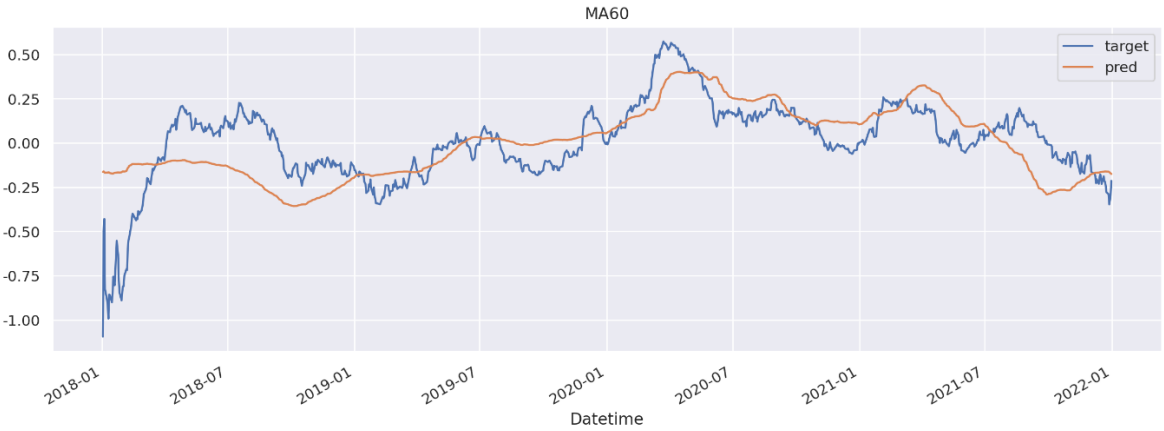
OLS 모델을 통해 예측 값을 생성하여 실제 값과 비교하는 시각화를 진행하였다. 플롯에서 보이는 주황색 선이 예측 값이며, 실제 값은 파란색 선으로 표현되어 있다.



이후, vs_plot_ma 함수를 정의하여 이동평균의 창 크기를 7, 30, 60, 120 으로 조절하며 각 플롯의 추세를 확인하였다.

```
def vs_plot_ma(data1, data2, n, name1 = '실제', name2 = '예측') :
    f, ax = plt.subplots(1)
    f.set_figheight(5)
    f.set_figwidth(15)

    _ = data1.rolling(window = n, min_periods = 1).mean().dropna().plot(ax = ax)
    _ = data2.rolling(window = n, min_periods = 1).mean().dropna().plot(ax = ax)
    plt.legend([name1, name2])
    plt.title(f"MA{n} 비교")
```





2) Ridge 모델

fit_regularized 함수를 사용하여 잔차의 제곱합과 함께 변수의 제곱합을 최소화하는 Ridge Regression 을 적용하였다.

a. 모델 적합

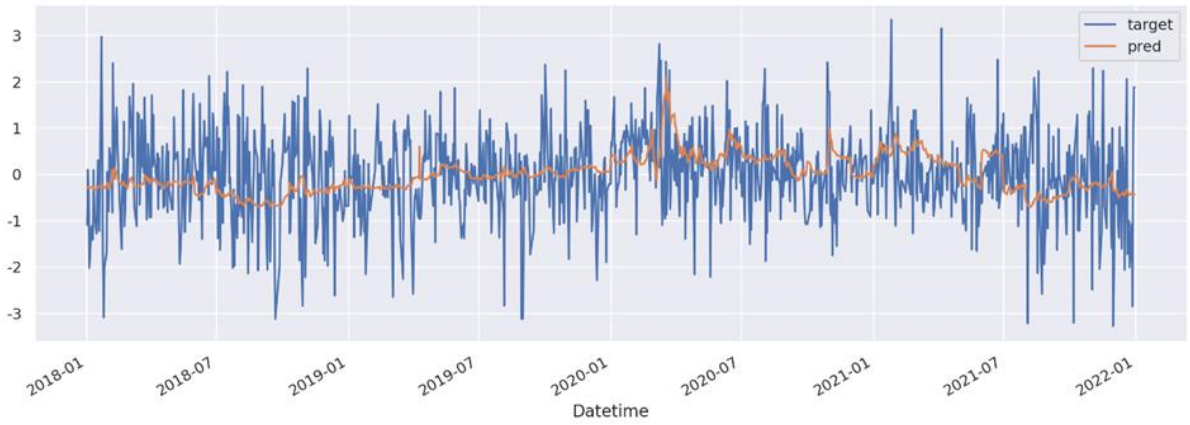
```
ridge = ols('target~vkospi+rt_rate+kospi_ma120+cons_sent+exchange', data = df_train_scaled).fit_regularized(alpha = 0.01, L1_wt = 0)
coef['Ridge'] = ridge.params
```

해당 모델에서는 가중치의 제곱합을 제한하는 L2 정규화를 사용하므로 L1_wt 는 0 으로 설정하였다. 규제의 정도를 나타내는 alpha 값은 0.01 로 약한 규제를 적용하였다. 회귀 계수 결과는 다음과 같다. 모든 계수가 0 에 수렴하는 경향을 가지고 있다.

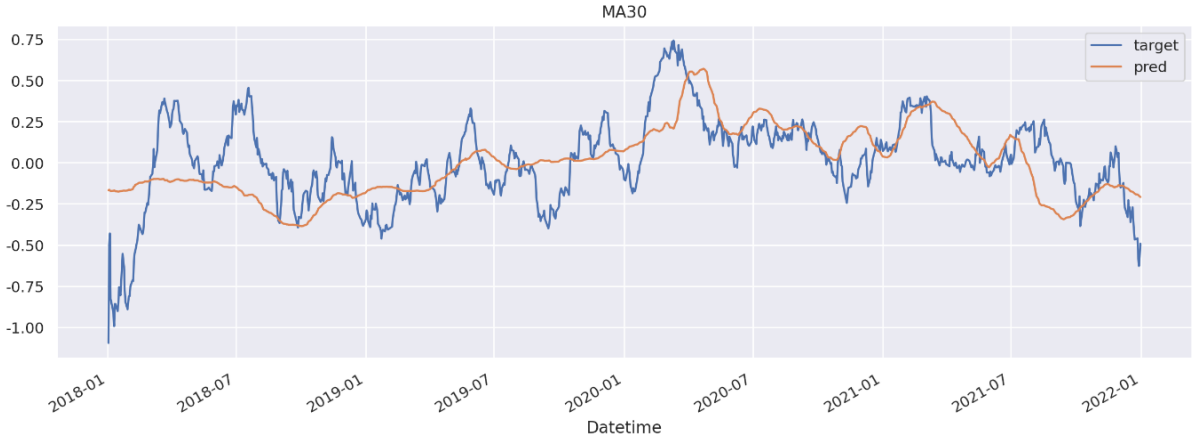
Intercept	8.202528e-18
vkospi	1.173501e-01
rt_rate	1.080444e-01
kospi_ma120	-1.155182e-01
cons_sent	1.366694e-01
exchange	6.995974e-02

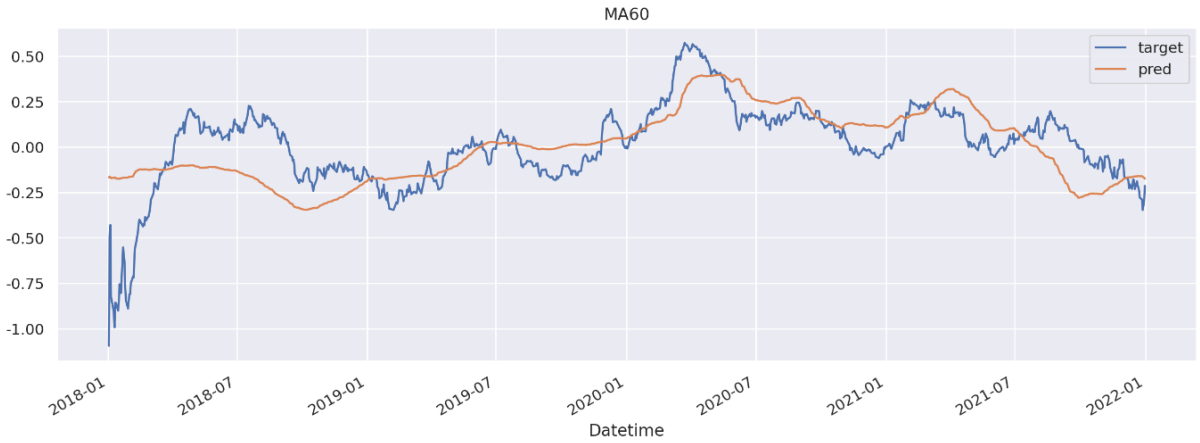
b. 모델 예측 및 시각화

Ridge 모델을 통해 예측 값을 생성하여 실제 값과 비교하는 시각화를 진행하였다. 플롯에서 보이는 주황색 선이 예측 값이며, 실제 값은 파란색 선으로 표현되어 있다.



이후, 앞서 OLS 모델에서 정의했던 vs_plot_ma 함수를 이용하여 이동평균의 창 크기를 7, 30, 60, 120 으로 조절하며 각 플롯의 추세를 확인하였다.





c. 하이퍼파라미터 튜닝

Ridge 모델의 하이퍼파라미터인 alpha 와 fit_intercept 을 튜닝하기 위해 그리드 서치(grid search)를 수행하였다.

```
param_grid = {
    'alpha' : [0.0001, 0.001, 0.01, 0.1, 1, 10],
    'fit_intercept' : [True, False],
}

ridge_est = Ridge()

kf = KFold(random_state=2023,
           n_splits=10,
           shuffle=True,
          )

grid_search = GridSearchCV(estimator=ridge_est,
                           param_grid=param_grid,
                           cv=kf,
                           n_jobs=-1,
                           verbose=2
                          )

# fit with (x_train, y_train)
grid_search.fit(X_train_scaled, y_train_scaled)

grid_search.best_params_
```

```
Fitting 10 folds for each of 12 candidates, totalling 120 fits
{'alpha': 10, 'fit_intercept': False}
```

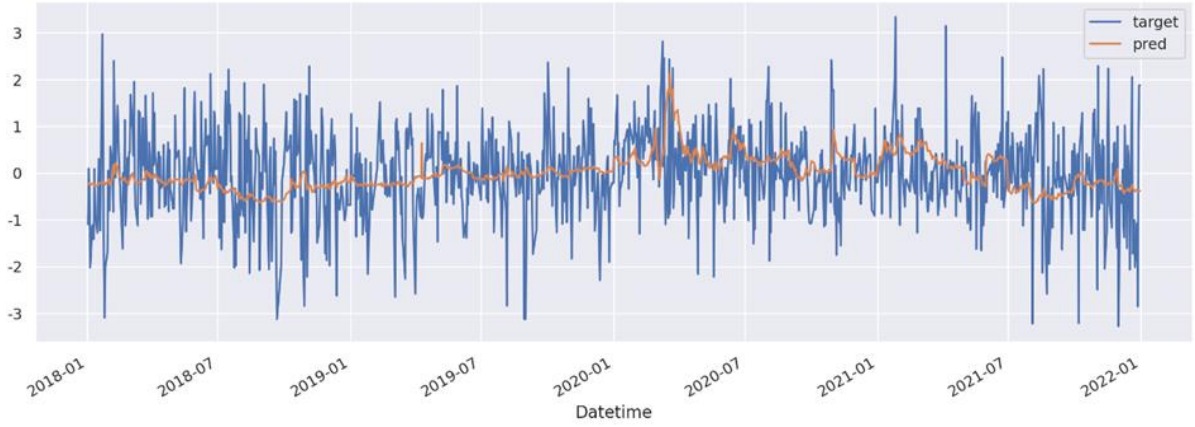
그리드 서치 결과, 규제 강도를 나타내는 alpha 값이 10 이고, Fit Intercept 가 False 일 때 최적의 모델이라는 것을 알 수 있었다.

d. 최적화된 모델 시각화

앞서 그리드 서치로 찾은 최적의 하이퍼파라미터를 적용해 최적화된 Ridge 모델을 시각화하였다.

```
# 사용할 변수 선택
selected_variables = ['vkospi', 'rt_rate', 'kospi_ma120', 'cons_sent', 'exchange']

# 선택된 변수만을 포함한 X_train_selected 생성
X_train_selected = X_train_scaled[selected_variables]
X_test_selected = X_test_scaled[selected_variables]
```

```
ridge = Ridge(alpha = 10, fit_intercept=False)
ridge.fit(X_train_selected, y_train_scaled)
ridge_pred = pd.DataFrame(ridge.predict(X_train_selected), index = y_train.index, columns = ['pred'])
```



3) LASSO 모델

fit_regularized 메서드를 사용하여 잔차의 제곱합과 함께 변수의 절댓값 합을 최소화하는 Lasso Regression 을 적용하였다.

a. 모델 적합

```
lasso = ols('target~vkospi+rt_rate+kospi_ma120+cons_sent+exchange', data = df_train_scaled).fit_regularized(alpha = 0.01, L1_wt = 1)
coef['LASSO'] = lasso.params
```

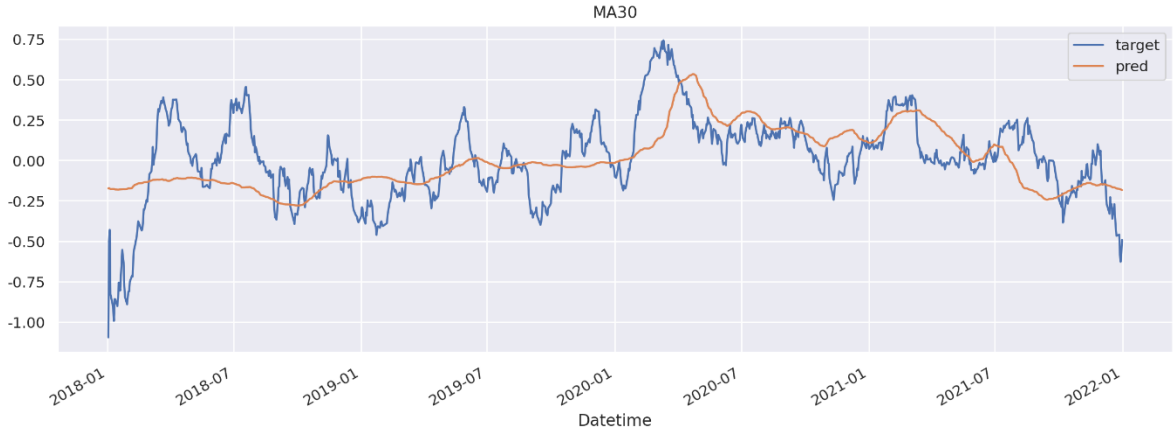
해당 모델에서는 가중치의 절댓값 합을 제한하는 L1 정규화를 사용하기 때문에 L1_wt 는 0 으로 설정하였다. 규제의 정도를 나타내는 alpha 값은 0.01 로, 약간의 규제를 적용하였다. 모델의 회귀 계수는 다음과 같다.

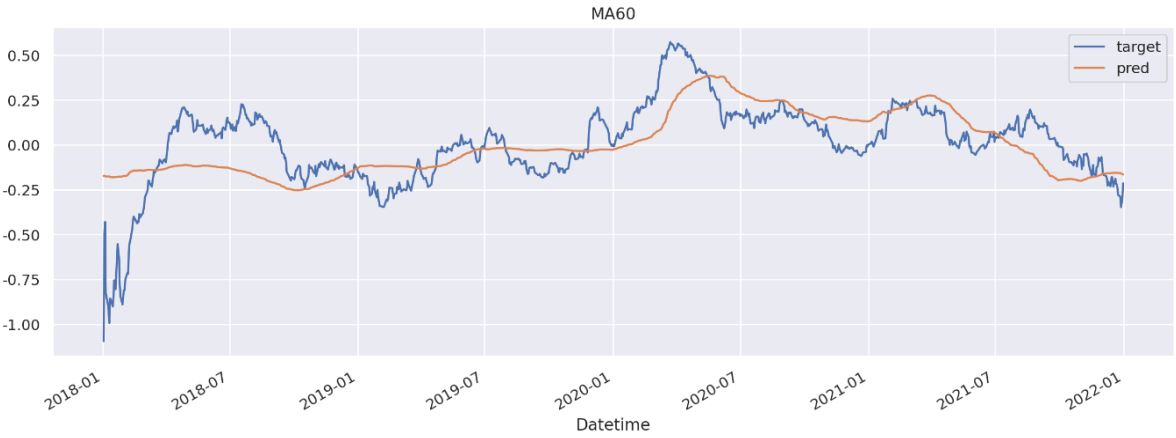
```
Intercept      0.000000
vkospi         0.083727
rt_rate        0.079882
kospi_ma120   -0.066424
cons_sent      0.048393
exchange       0.031993
```

b. 모델 예측 및 시각화:

Lasso 모델을 통해 예측 값을 생성하여 실제 값과 비교하는 시각화를 진행하였다. 플롯에서 보이는 주황색 선이 예측 값이며, 실제 값은 파란색 선으로 표현되어 있다.

이후, 앞서 OLS 모델에서 정의했던 vs_plot_ma 함수를 이용하여 이동평균의 창 크기를 7, 30, 60, 120 으로 조절하며 각 플롯의 추세를 확인하였다.





c. 하이퍼파라미터 튜닝

하이퍼파라미터 튜닝을 위한 그리드 서치를 진행하였다. x_train, y_train 으로 그리드 서치를 적합하였다.

```
param_grid = {
    'alpha' : [0.0001, 0.001, 0.01, 0.1, 1, 10],
    'fit_intercept' : [True, False],
}

lasso_est = Lasso()

kf = KFold(random_state=2023,
           n_splits=10,
           shuffle=True,
          )

grid_search = GridSearchCV(estimator=lasso_est,
                           param_grid=param_grid,
                           cv=kf,
                           n_jobs=-1,
                           verbose=2
                          )

# fit with (x_train, y_train)
grid_search.fit(X_train_scaled, y_train_scaled)
```

result:

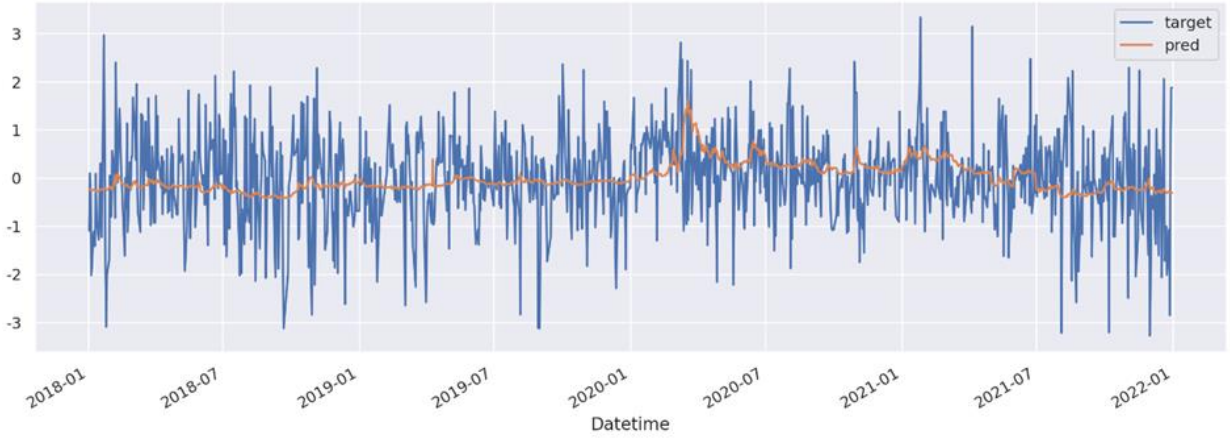
```
Fitting 10 folds for each of 12 candidates, totalling 120 fits
{'alpha': 0.01, 'fit_intercept': False}
```

그리드 서치 결과, 규제 강도를 나타내는 alpha 값이 0.01 이고, fit_Intercept 가 False 일 때 최적의 모델이라는 것을 알 수 있었다.

d. ~~d.~~최적화된 모델 시각화

최적화된 하이퍼파라미터 조합의 모델로 예측을 생성한 후 시각화를 진행하였다.

```
lasso = Lasso(alpha = 0.01, fit_intercept=False)
lasso.fit(X_train_selected, y_train_scaled)
lasso_pred = pd.DataFrame(lasso.predict(X_train_selected), index = y_train.index, columns = ['pred'])
```



4) Elastic Net 모델

Elastic Net은 Ridge와 LASSO를 합쳐 놓은 형태이며, 목적식은 다음과 같다.

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \quad s.t. \quad \alpha|\boldsymbol{\beta}\| + (1-\alpha)\|\boldsymbol{\beta}\|^2 < t$$

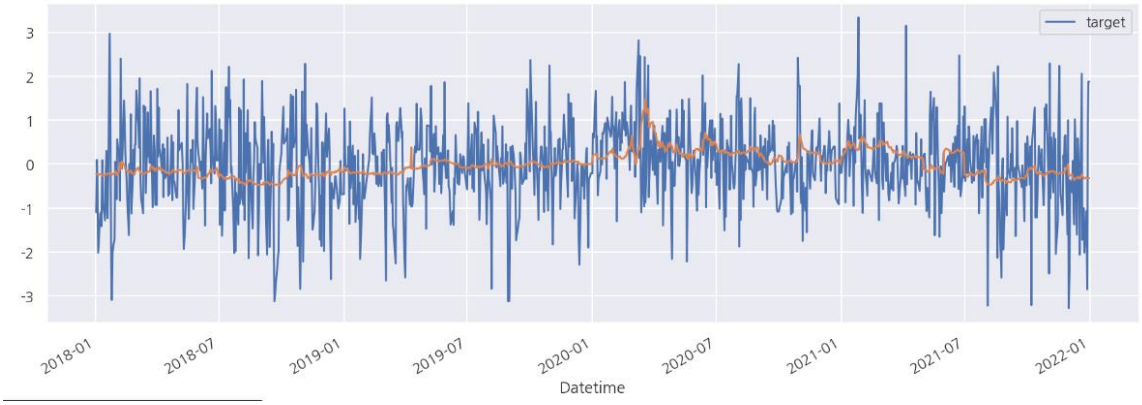
Elastic Net은 α가 0에 가까울수록 Ridge regression과 그 특성이 유사해지며, 반대로 α가 1에 가까울수록 LASSO와 유사한 특성을 갖는다. Elastic Net은 표면적으로는 단순히 L1 norm과 L2 norm을 혼합한 형태에 불과해보이지만 사실 이 둘을 동시에 사용함으로써 변수 간 grouping effect를 제공하는 엄청난 이점을 갖게 된다. 즉, 변수간 상관관계가 큰 변수들에 유사한 가중치를 줌으로써 중요한 변수들은 똑같이 중요하게 취급하고 중요하지 않은 변수들은 모두 공평하게 중요하지 않게 취급한다. 즉, 변수간 상관관계가 크더라도 중요하지 않은 변수들을 모두 버리면서 중요한 변수들만 잘 골라내어 중요도와 상관관계에 따라 적합한 가중치를 적용할 수 있다.

a. 모델 적합

```
elastic_net = ols('target~vkospi+rt_rate+kospi_ma120+cons_sent+exchange', data = df_train_scaled).fit_regularized(alpha = 0.01, L1_wt = 0.5)
coef['Elastic Net'] = elastic_net.params
```

b. 모델 예측 및 시각화

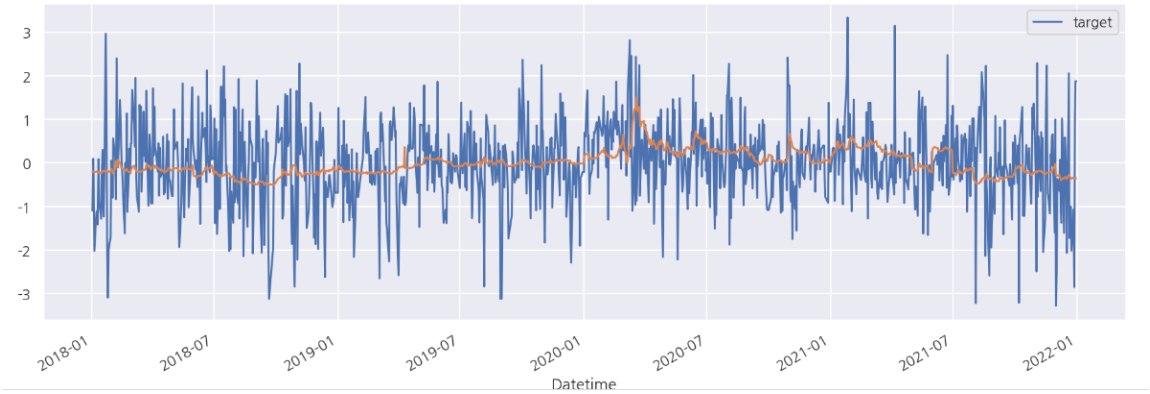
```
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)

_ = y_train_scaled.plot(ax = ax)
_ = (2 * elastic_net.predict(X_train_scaled)).plot(ax = ax)
```



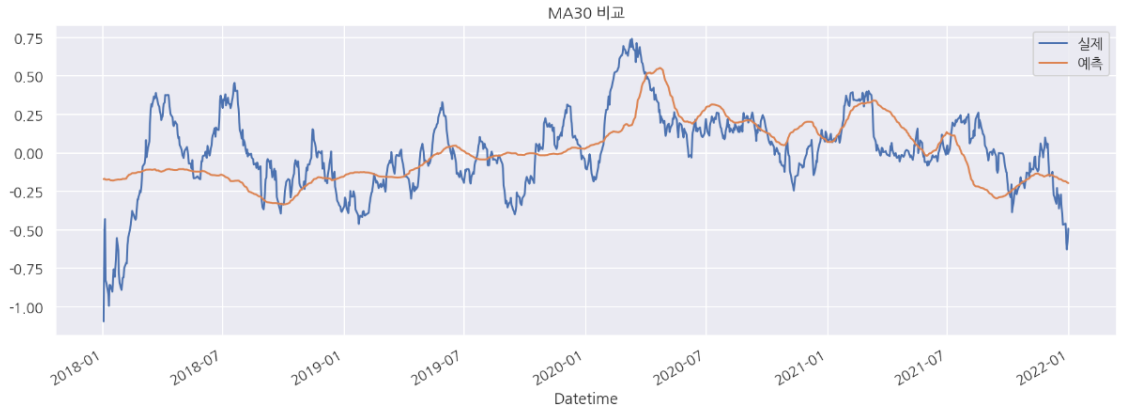
'elastic_net'는 우리가 선정한 5가지 변수를 OLS를 통해 target식을 완성하고 이를 Elastic Net을 적용하였다. 위는 y_train 값과 'elastic_net'의 2배를 비교한 plot이다.

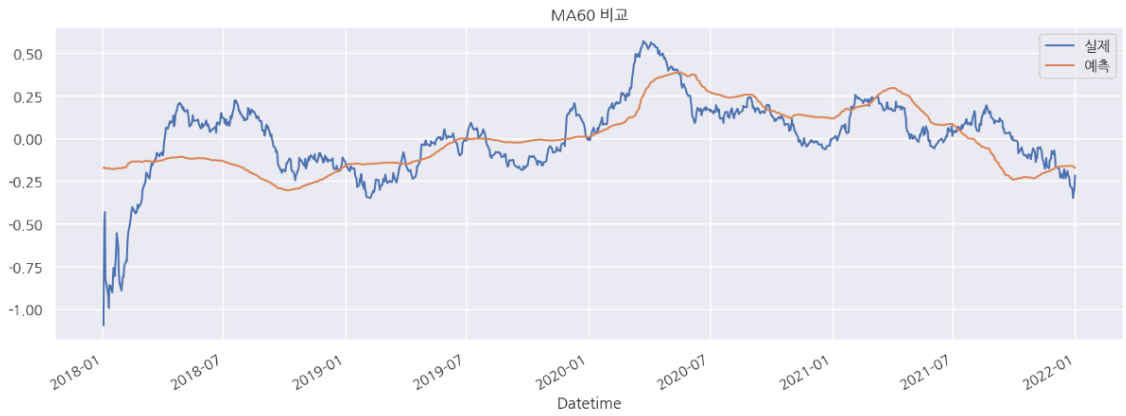
아래의 'elastic_net_origin' 변수는 모든 독립변수에 대하여 OLS를 통해 Elastic Net을 적용하였다. 마찬가지로 y_train 값과 'elastic_net_origin'의 2배를 비교한 plot이다.

```
fomula = 'target ~ ' + '+'.join(X_train_scaled.columns)
elastic_net_origin = ols(fomula, data = df_train_scaled).fit_regularized(alpha = 0.01, L1_wt = 0.5)

f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)

_ = y_train_scaled.plot(ax = ax)
_ = (2 * elastic_net_origin.predict(X_train_scaled)).plot(ax = ax)
```



결과적으로 모든 변수에 대해 OLS를 적용한 것(elastic_net_origin)보다 우리가 채택한 변수 5가지를 사용한 elastic_net을 적용한 것이 y_train과 더 비슷한 추이를 보여준다.



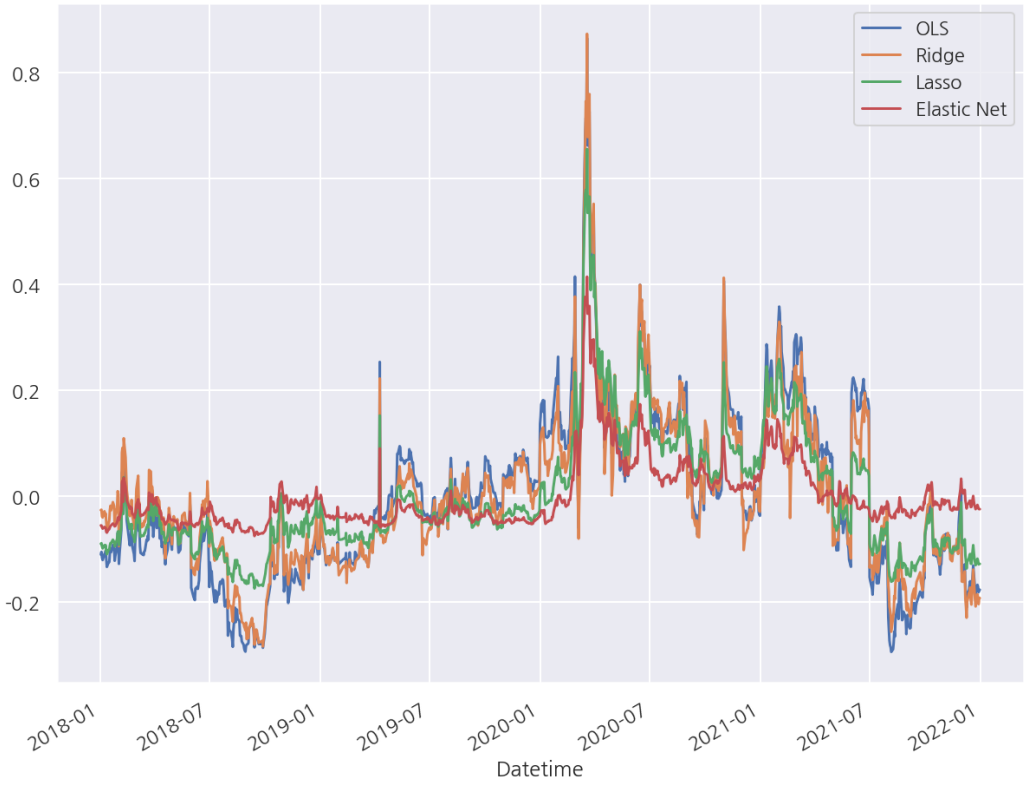


Elastic Net의 알파에 대해 grid search를 진행한 결과 알파가 0.1, fit_intercept = False일 때 CV에서의 성능이 가장 좋았다.

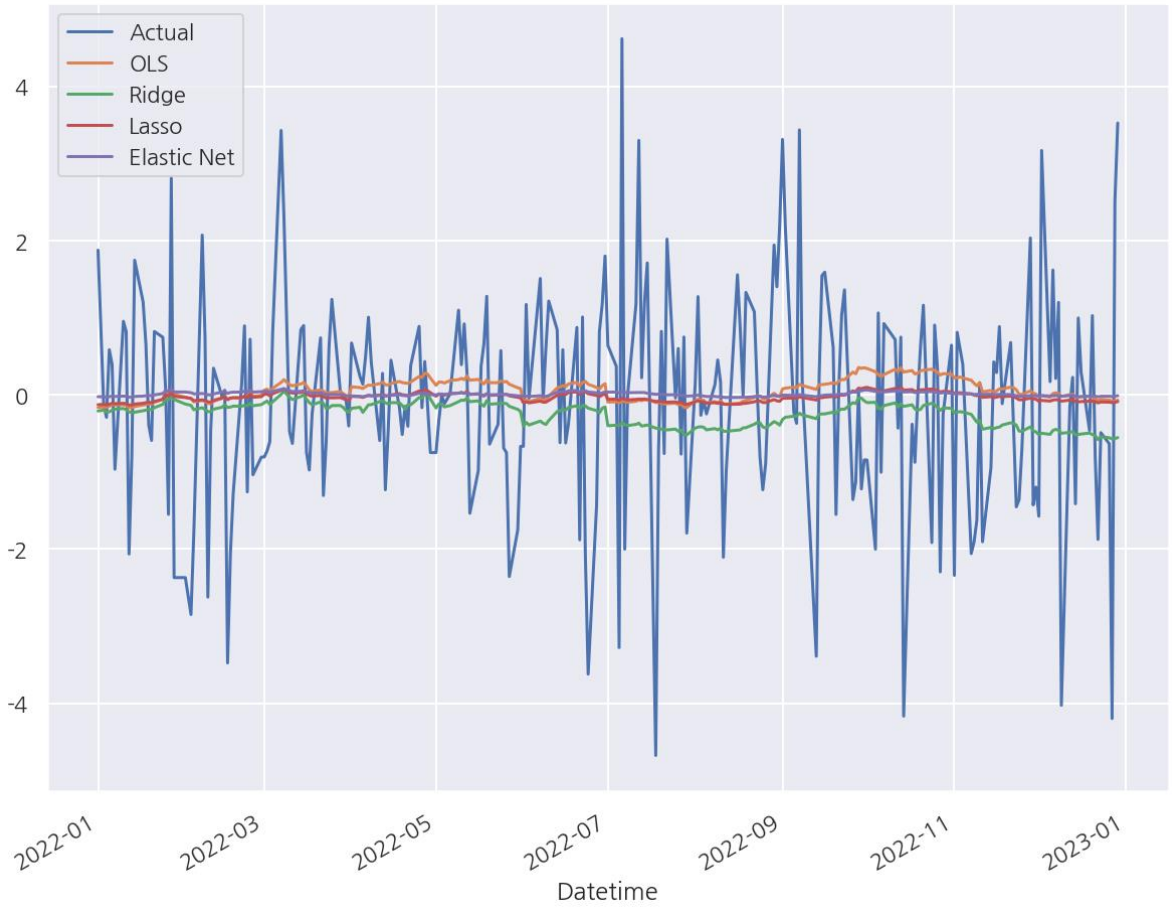
5. 결과 해석 및 결론

1) 결과 해석

4가지 모델에 대해 결과를 비교해보았다. Train set에 대한 plot을 아래와 같이 작성하였다.



Test set에 대한 plot을 아래와 같이 작성하였다.



```
[Train MSE]
------------------------------
OLS : 0.9116736105691042
Ridge : 0.9101030132591473
Lasso : 0.9137761466995219
Elastic Net : 0.9212568968119216
```

```
[Test MSE]
------------------------------
OLS : 2.017776088723826
Ridge : 2.038050508157004
Lasso : 1.9825249139713939
Elastic Net : 1.9807949851764564
```

Elastic Net 의 경우 학습데이터에 대해서는 MSE 가 높지만, 상대적으로 과적합되지 않아 일반화성능이 좋다.

2) 결론

우리가 도출한 투자심리지수가 코스피지수와 강력한 음의 상관관계를 보이기 때문에, 투자자들의 비합리성을 보이는 지수의 소기의 목적은 달성하였다. 그러나 여러 지표를 통해 통계적 유의성이 부족하여 다른 모델 구축을 통해 후속연구가 필요하다.