G-Research Crypto Forecasting

2018108247 공지혁

개요

1 개요

2 필요성

3 라이브러리

4 데이터셋

5 시각화(visualization)

6 데이터 전처리

7 모델 구현

8 모델 평가

9 소감

필요성

암호화폐는 굉장히 인기가 많고 격한 시장으로, 투자자들에게 엄청난 수익(또는 그 반대)을 가져다 주기도 합니다. 많은 분들이 아실 비트코인이나 이더리움을 포함해 지금까지 수 천개의 암호화폐들이 만들어졌습니다.

CryptoCompare에 따르면(2021년 7월 25일 기준) 암호화폐는 지난 한 해 동안 매일 평균 거래량이 410억 달러로, 거래소에서 광범위하게 거래되었습니다.

암호 화폐들 간에 주요한 특징이 있다면, 가격 변동이 서로 연결되어 있다는 점입니다. 예를 들어 비트코인은 암호 화폐 시장의 가격 변동을 주로 이끌어왔지만, 다른 수 많은 코인도 이 추세와 함께 움직여왔습니다. 모든 코인이 서로 유기적으로 시장에 영향을 주고 있는 것입니다.

암호화폐 수익 예측은 여전히 풀리지 않은 문제입니다. 과한 변동성, 시계열 데이터의 비정상성, 시장과 밈 조작, 매우 빠르게 변화하는 시황과 자산간의 관계를 고려할 때 이는 매우 어려운 예측 작업입니다. 그러나 이렇게 어려운 문제인만큼 상당히 매력적인 머신러닝 영역이기도 합니다.

라이브러리

데이터셋

In [3]:
```
# pandas 라이브러리를 사용해 csv파일을 읽어옵니다.
# 읽어온 데이터는 데이터프레임으로 변환되어 crypto_df에 저장됩니다.
crypto_df = pd.read_csv(data_folder + 'train.csv')
```

In [4]:
```
# 데이터프레임에서 가장 상단 10개만 살펴봅니다.
crypto_df.head(10)
```

Out[4]:

	timestamp	Asset_ID	Count	Open	High	Low	Close	Volume	VWAP
0	1514764860	2	40.0	2376.580000	2399.5000	2357.1400	2374.590000	19.233005	2373.11639
1	1514764860	0	5.0	8.530000	8.5300	8.5300	8.530000	78.380000	8.530000
2	1514764860	1	229.0	13835.194000	14013.8000	13666.1100	13850.176000	31.550062	13827.0620
3	1514764860	5	32.0	7.659600	7.6596	7.6567	7.657600	6626.713370	7.657713
4	1514764860	7	5.0	25.920000	25.9200	25.8740	25.877000	121.087310	25.891363
5	1514764860	6	173.0	738.302500	746.0000	732.5100	738.507500	335.987856	738.839291
6	1514764860	9	167.0	225.330000	227.7800	222.9800	225.206667	411.896642	225.197944
7	1514764860	11	7.0	329.090000	329.8800	329.0900	329.460000	6.635710	329.454118
8	1514764920	2	53.0	2374.553333	2400.9000	2354.2000	2372.286667	24.050259	2371.43449
9	1514764920	0	7.0	8.530000	8.5300	8.5145	8.514500	71.390000	8.520215

데이터셋

- **timestamp** : 모든 타임스탬프는 초 단위 Unix 타임스탬프(1970-01-01 00:00:00.000 UTC 이후 경과된 '초')로 반환됩니다. 이 데이터 세트의 타임스탬프는 60의 배수로 표현되어 있습니다. 즉, '분' 단위 데이터를 나타냅니다.
- **Asset_ID** : Asset ID는 각 암호화폐와 매핑되어 있습니다(예를 들어 Asset_ID 1은 비트코인). `asset_details.csv` 파일에서 매핑 정보를 상세히 볼 수 있습니다.
- **Count** : 각 타임스탬프별 거래 수를 의미합니다(분 단위 총 거래 수).
- **Open** : USD(달러) 단위 시가.
- **High** : 매 분마다 도달한 USD(달러) 단위 고가.
- **Low** : 매 분마다 도달한 USD(달러) 단위 저가.
- **Close** : USD(달러) 단위 종가.
- **Volume** : 매수, 매도 수량.
- **VWAP** : VWAP는 거래량 가중 평균 가격(Volume Weighted Average Price)을 의미합니다. 즉, 시간 간격 동안 자산의 평균 가격으로, 거래량으로 가중된 수치를 보여줍니다.
- **Target** : 15분 단위의 log변환된 잔여이익(residual returns)을 보여줍니다. *(여기서 잔여이익이란 수수료 등을 제외하고 얻은 순이익을 뜻합니다.*

데이터 분석

In [5]:
```
asset_details = pd.read_csv(data_folder + 'asset_details.csv')
asset_details
```

In [6]:
```
# ignore_index를 True로 주지 않으면 현재의 인덱스가 컬럼으로 들어오게 됩니다.
# 인덱스는 0부터 새롭게 만들고 기존 인덱스는 버릴 것이니 True 옵션을 주겠습니다.
asset_details = asset_details.sort_values('Asset_ID', ignore_index=True)
asset_details
```

Out[6]:

	Asset_ID	Weight	Asset_Name
0	0	4.304065	Binance Coin
1	1	6.779922	Bitcoin
2	2	2.397895	Bitcoin Cash
3	3	4.406719	Cardano
4	4	3.555348	Dogecoin
5	5	1.386294	EOS.IO
6	6	5.894403	Ethereum
7	7	2.079442	Ethereum Classic
8	8	1.098612	IOTA
9	9	2.397895	Litecoin
10	10	1.098612	Maker
11	11	1.609438	Monero
12	12	2.079442	Stellar
13	13	1.791759	TRON

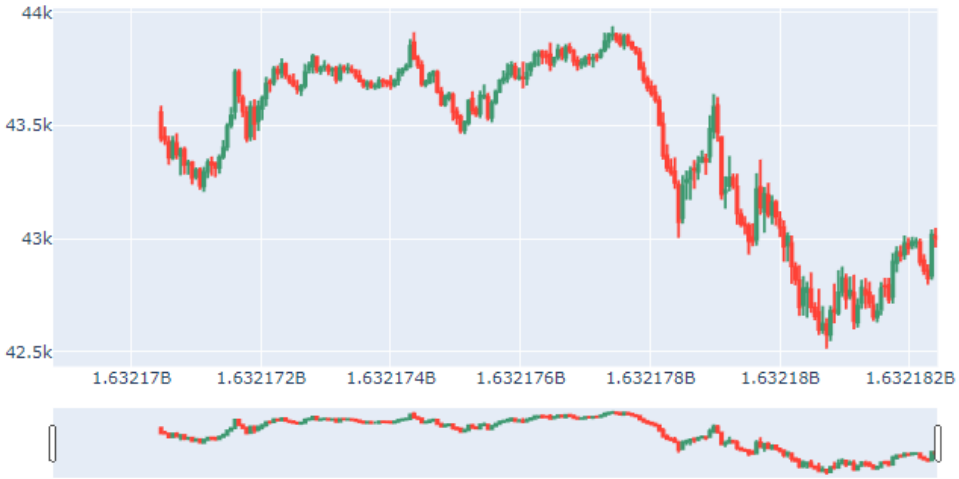
데이터 분석(EDA)

In [10]:

```
import plotly.graph_objects as go

# index(timestamp)를 x축으로 놓고 시가/고가/저가/종가 데이터를 입력해줍니다.
fig = go.Figure(
    data=[go.Candlestick(
            x=btc_mini.index,
            open=btc_mini['Open'],
            high=btc_mini['High'],
            low=btc_mini['Low'],
            close=btc_mini['Close']
        )]
    )

fig.show()
```



시각화 (visualization)

In [21]:

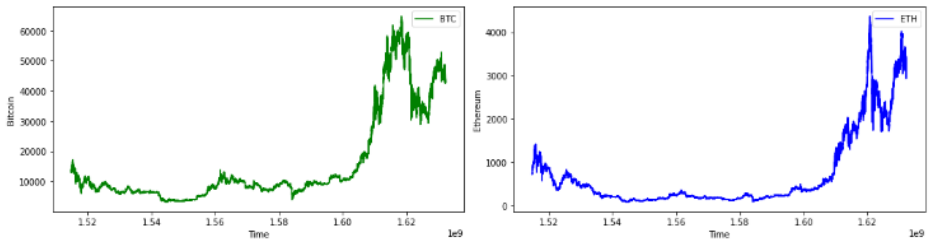
```
import matplotlib.pyplot as plt

# 시각화 공간을 15 x 4 크기로 만든다.
f = plt.figure(figsize=(15,4))

# 전체 시각화 공간을 1행 2열로 분할하고 그 중 1번째 공간을 변수 'ax1'에 할당하겠습니다.
ax1 = f.add_subplot(121)
# 공간 할당된 ax1에 비트코인 종가 그래프를 그립니다.
plt.plot(btc['Close'], label='BTC', color='green')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Bitcoin')

# 앞에서 전체 시각화 공간을 1행 2열로 분할했었습니다. 그 중 2번째 공간을 변수 'ax2'에 할당하겠습니다.
ax2 = f.add_subplot(122)
# 공간 할당된 ax2에 비트코인 종가 그래프를 그립니다.
plt.plot(eth['Close'], label='ETH', color='blue')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Ethereum')

# 두 그래프 사이 간격을 좁혀줍니다.
plt.tight_layout()
# 그래프를 출력합니다.
plt.show()
```



시각화 (visualization)

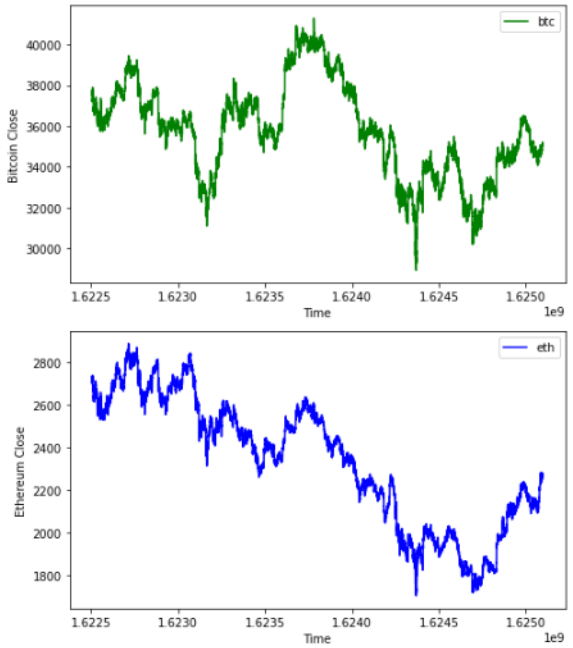
In [29]:
```
btc_mini_2021 = btc.loc[totimestamp('01/06/2021'):totimestamp('01/07/2021')]
eth_mini_2021 = eth.loc[totimestamp('01/06/2021'):totimestamp('01/07/2021')]
```

In [30]:
```
# 7x8 크기의 시각화 공간을 할당받습니다.
f = plt.figure(figsize=(7,8))

# 2행 1열로 시각화 공간을 나누고 그 중 1번째 공간을 ax1 변수로 제어합니다.
ax1 = f.add_subplot(211)
# ax1 공간에 2021년 6월 1일 비트코인 가격 차트를 그립니다.
plt.plot(btc_mini_2021['Close'], label='btc', color='green')
plt.legend() # label을 표시합니다
plt.xlabel('Time')
plt.ylabel('Bitcoin Close')

# 앞에서 2행 1열로 시각화 공간을 나눴습니다. 그 중 2번째 공간을 ax2 변수로 제어합니다.
ax2 = f.add_subplot(212)
plt.plot(eth_mini_2021['Close'], label='eth', color='blue')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Ethereum Close')

plt.tight_layout() # 두 그래프 사이 간격을 좁힙니다.
plt.show()
```



데이터 전처리

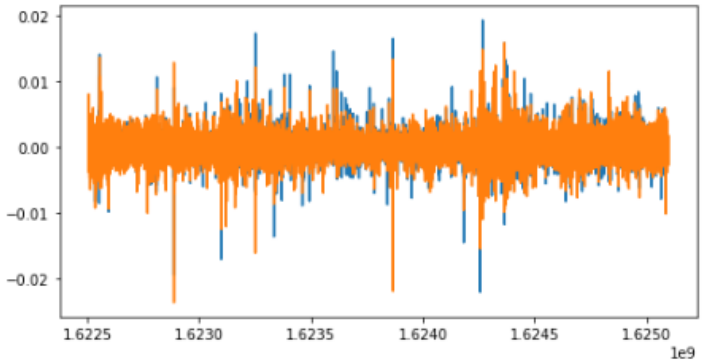
In [31]:
```
# 로그 변환 처리를 위해 함수를 만들어줍니다.
def log_return(series, periods=1):
    return np.log(series).diff(periods=periods)
```

In [32]:
```
import scipy.stats as stats

# log return btc --> lret_btc 로 표현했습니다.
# log_return 함수는 우리가 위에서 미리 만들어둔 함수로, 내부적으로 diff 함수를 사용하기 때문에 첫번째 row는 버립니다.
# 첫번째 row(행)를 버리는 이유는 해당 데이터 포인터는 null이 들어가 있기 때문입니다.(이전 데이터가 없으므로 null이 채워짐)
lret_btc = log_return(btc_mini_2021['Close'])[1:]
lret_eth = log_return(eth_mini_2021['Close'])[1:]
```

In [33]:
```
# lret_btc(Series 타입)의 이름은 'Close'인 상태입니다. 이름을 변경해줍니다.
lret_btc.rename('lret_btc', inplace=True)
# lret_eth(Series 타입)의 이름도 'Close'인 상태입니다. 이름을 변경해줍니다.
lret_eth.rename('lret_eth', inplace=True)

# 8x4 크기로 시각화 합니다.
plt.figure(figsize=(8,4))
# 한 공간에 2개 그래프를 겹쳐 그립니다.
plt.plot(lret_btc)
plt.plot(lret_eth)
plt.show()
```



데이터 전처리

In [34]:

```
# 앞에서 했던 것과 같은 방식으로 로그 수익을 구해줄 것입니다.
# 단, 이번에는 하루가 아닌 전체 기간을 처리해줍니다.
lret_btc_long = log_return(btc['Close'])[1:]
lret_eth_long = log_return(eth['Close'])[1:]

# 시리즈 데이터 이름을 Close에서 각 데이터에 맞게 바꿔줍니다.
lret_btc_long.rename('lret_btc', inplace=True)
lret_eth_long.rename('lret_eth', inplace=True)

# 두 데이터를 병합합니다.
two_assets = pd.concat([lret_btc_long, lret_eth_long], axis=1)
two_assets
```

Out[34]:

	lret_btc	lret_eth
timestamp		
1514764920	-0.001595	-0.000335
1514764980	-0.001939	-0.001027
1514765040	-0.002414	-0.000543
1514765100	-0.003137	-0.001898
1514765160	-0.000569	-0.000687
...	...	...
1632182160	-0.002007	-0.003137
1632182220	-0.000910	-0.001305
1632182280	-0.000770	-0.001165
1632182340	0.004433	0.006670
1632182400	-0.000343	-0.001542

1956959 rows × 2 columns

데이터 전처리

In [35]:
```
two_assets.corr()
```

Out[35]:

	lret_btc	lret_eth
lret_btc	1.000000	0.568814
lret_eth	0.568814	1.000000

In [36]:
```
# 타임스탬프는 60초 간격입니다.
# 60의 배수(여기서는 600,000)로 나눈 나머지로 묶게 되면 시간순으로 구획별 상관도를 확인할 수 있습니다.
# ex. 유닉스 시간이 60만보다 작은 1번째 구간, 60만보다 크고 120만보다 작은 2번째 구간 ...
two_assets.groupby(two_assets.index//(60*10000)).corr()
```

Out[36]:

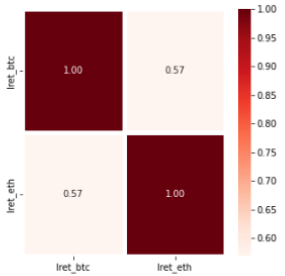
		lret_btc	lret_eth
timestamp			
2524	lret_btc	1.000000	0.212106
	lret_eth	0.212106	1.000000
2525	lret_btc	1.000000	0.369729
	lret_eth	0.369729	1.000000
2526	lret_btc	1.000000	0.766609
...	...	...	...
2718	lret_eth	0.893232	1.000000
2719	lret_btc	1.000000	0.847503
	lret_eth	0.847503	1.000000
2720	lret_btc	1.000000	0.871091
	lret_eth	0.871091	1.000000

394 rows × 2 columns

[39]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(5,5))
sns.heatmap(data = two_assets.corr(), annot=True,
fmt = '.2f', linewidths=5, cmap='Reds')
```

[39]: <AxesSubplot:>

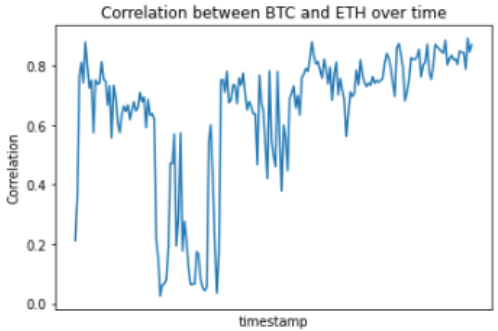


데이터 전처리

In [38]:

```
corr_time = two_assets.groupby(two_assets.index//(10000*60)).corr().loc[:,"lret_btc"].loc[:,"lret_eth"]

corr_time.plot()
plt.xticks([])
plt.ylabel("Correlation")
plt.title("Correlation between BTC and ETH over time")
plt.show()
```



모델 구현

In [42]:
```
# 거래데이터로부터 모델에 넣어줄 입력 피처들을 선정합니다.
# 입력 피처는 4개입니다 : 5분 단위 로그 수익률, 1분 단위 로그 수익률의 절대값, 캔들 윗꼬리와 아랫꼬리

# 먼저 윗꼬리, 아랫꼬리 피처를 생성해줄 함수를 만들겠습니다.
# 윗꼬리는 고가에서 몸통 윗부분까지입니다. 따라서 시가와 종가 중 큰 값과 고가를 비교해주면 됩니다.
upper_shadow = lambda asset: asset['High'] - np.maximum(asset['Close'], asset['Open'])
# 아랫꼬리는 몸통 아랫부분부터 저가까지입니다. 따라서 시가와 종가 중 낮은 값과 저가를 비교해주면 됩니다.
lower_shadow = lambda asset: np.minimum(asset['Close'], asset['Open']) - asset['Low']

# 비트코인 : 선정한 4개 입력 피처를 데이터프레임으로 묶어줍니다.
X_btc = pd.concat([
    log_return(btc['VWAP'], periods=5),
    log_return(btc['VWAP'], periods=1).abs(),
    upper_shadow(btc),
    lower_shadow(btc)
], axis=1)

# 이더리움 : 동일합니다.
X_eth = pd.concat([
    log_return(eth['VWAP'], periods=5),
    log_return(eth['VWAP'], periods=1).abs(),
    upper_shadow(eth),
    lower_shadow(eth)
], axis=1)

# 마지막으로 비트코인과 이더리움 각각에 대해 예측할 y값을 변수에 담아주겠습니다.
y_btc = btc['Target']
y_eth = eth['Target']
```

In [43]:
```
X_btc
```

Out[43]:

	VWAP	VWAP	0	1
timestamp				
1514764860	NaN	NaN	163.624000	169.084000
1514764920	NaN	0.000961	217.264000	148.102000
1514764980	NaN	0.002481	176.500000	200.314000
1514765040	NaN	0.001629	196.488000	191.760000
1514765100	NaN	0.003489	189.900000	170.474000
...	...	...	...	...
1632182160	0.000074	0.001123	18.070849	20.752857
1632182220	-0.001954	0.001303	27.802857	19.845714
1632182280	-0.002773	0.000828	28.114286	29.820000
1632182340	-0.001061	0.002137	24.882857	20.912802
1632182400	0.000643	0.001760	38.548750	40.865000

1956960 rows × 4 columns

In [44]:
```
y_btc
```

Out[44]:
```
timestamp
1514764860   -0.014643
1514764920   -0.015037
1514764980   -0.010309
1514765040   -0.008999
1514765100   -0.008079
                ...
1632182160         NaN
1632182220         NaN
1632182280         NaN
1632182340         NaN
1632182400         NaN
Name: Target, Length: 1956960, dtype: float64
```

모델 구현

In [45]:

```
# 먼저 분리할 지점, 즉 시간 범위를 지정해줍니다.
# 여기서는 50:50으로 나눠줍니다. 학습데이터는 5월, 평가데이터는 6월로 지정합니다.
train_window = [totimestamp("01/05/2021"), totimestamp("30/05/2021")]
test_window = [totimestamp("01/06/2021"), totimestamp("30/06/2021")]

# 이제 지정한 시간 범위대로 비트코인, 이더리움 각각 학습/평가 데이터를 생성해줍니다.
# 단, 여기서 우리의 목표는 아주 단순한 회귀 모델을 만드는 것이므로 window size는 1로 합니다.
# (시계열 데이터에 window size는 매우 중요합니다. window size란 몇 개의 연결된 x 데이터(행)로 y값을 예측할 것인가에 대한 정보입니다.)
X_btc_train = X_btc.loc[train_window[0]:train_window[1]].fillna(0).to_numpy() # 빈 값은 0으로 채워주고 numpy 배열로 변환해줍니다.
y_btc_train = y_btc.loc[train_window[0]:train_window[1]].fillna(0).to_numpy()

X_eth_train = X_eth.loc[train_window[0]:train_window[1]].fillna(0).to_numpy()
y_eth_train = y_eth.loc[train_window[0]:train_window[1]].fillna(0).to_numpy()

X_btc_test = X_btc.loc[test_window[0]:test_window[1]].fillna(0).to_numpy()
y_btc_test = y_btc.loc[test_window[0]:test_window[1]].fillna(0).to_numpy()

X_eth_test = X_eth.loc[test_window[0]:test_window[1]].fillna(0).to_numpy()
y_eth_test = y_eth.loc[test_window[0]:test_window[1]].fillna(0).to_numpy()
```

In [47]:

```
print(X_btc_train.shape, X_btc_test.shape)
print(X_eth_train.shape, X_eth_test.shape)

print(y_btc_train.shape, y_btc_test.shape)
print(y_eth_train.shape, y_eth_test.shape)
```

```
(41761, 4) (41761, 4)
(41761, 4) (41761, 4)
(41761,) (41761,)
(41761,) (41761,)
```

모델 구현

In [48]:

```
from sklearn.preprocessing import StandardScaler

# 스케일링 인스턴스를 생성합니다.
scaler = StandardScaler()

# train 데이터에는 fit, trainsform 과정을 모두 수행해주고, test 데이터에는 transform만 적용합니다.
# - fit은 스케일링을 위해 평균, 데이터 범위 등을 찾는 과정입니다.
# - transform은 fit으로 찾은 통계정보를 가지고 실제 스케일링(정규화)을 수행하는 부분입니다.
# - 만약, test 데이터에 fit을 적용하게 되면 이전에 train 데이터에서 찾은 통계정보가 test 데이터 기준
으로 리셋되버립니다.
X_btc_train_scaled = scaler.fit_transform(X_btc_train)
X_btc_test_scaled = scaler.transform(X_btc_test)

X_eth_train_scaled = scaler.fit_transform(X_eth_train)
X_eth_test_scaled = scaler.transform(X_eth_train)
```

모델 구현

In [49]:
```
from sklearn.linear_model import LinearRegression

# 먼저, 하나의 자산만 예측하는 모형입니다.
# 비트코인, 이더리움 각각에 대해 예측 모형을 만들어보겠습니다.

# 1. 회귀 모형 인스턴스를 생성합니다.
lr = LinearRegression()
# 2. X(입력)과 y(출력)을 쌍으로 학습시킵니다.(지도학습)
lr.fit(X_btc_train_scaled, y_btc_train)
# 3. 학습시킨 회귀 모형으로 새로운 데이터(test셋)를 받아 예측합니다.
y_pred_lr_btc = lr.predict(X_btc_test_scaled)

# 예측 결과를 출력해보겠습니다.
print(y_pred_lr_btc)
```

```
[ 3.99725659e-05 -3.55222618e-04 -4.20519813e-04 ...  2.54616439e-05
  5.43563589e-06 -3.09767883e-05]
```

비트코인

In [50]:
```
# 동일하게 이더리움 수익 예측도 진행합니다.

lr.fit(X_eth_train_scaled, y_eth_train)
y_pred_lr_eth = lr.predict(X_eth_test_scaled)

# 예측 결과를 출력해보겠습니다.
print(y_pred_lr_eth)
```

```
[6.93201503e-05 5.12309687e-07 2.48939606e-05 ... 6.59544220e-05
 2.48855722e-05 2.74144832e-05]
```

이더리움

모델 구현

In [52]:
```
np.concatenate((X_btc_train_scaled, X_eth_train_scaled), axis=1).shape
```

Out[52]:
```
(41761, 8)
```

In [55]:
```
np.column_stack((y_btc_train, y_eth_train))
```

Out[55]:
```
array([[ 0.00306574, -0.0025832 ],
       [ 0.00415549, -0.00108809],
       [ 0.00444132, -0.00196447],
       ...,
       [ 0.00194192,  0.0019096 ],
       [ 0.002041  ,  0.00018122],
       [ 0.00101983,  0.00073888]])
```

In [56]:
```
print(y_btc_train.shape)
print(y_eth_train.shape)
print(np.column_stack((y_btc_train, y_eth_train)).shape)
```

```
(41761,)
(41761,)
(41761, 2)
```

모델 구현

In [57]:
```
from sklearn.multioutput import MultiOutputRegressor

# 2차원 행렬 병합 방식입니다. axis=1은 컬럼 기준(가로) 병합을 의미합니다. 기본값은 열(세로) 병합입니다.
X_both_train = np.concatenate((X_btc_train_scaled, X_eth_train_scaled), axis=1)
X_both_test = np.concatenate((X_btc_test_scaled, X_eth_test_scaled), axis=1)

# 1차원 벡터 병합 방식입니다. axis 옵션 없이 기본값으로 컬럼 기준 병합을 수행합니다.
y_both_train = np.column_stack((y_btc_train, y_eth_train))
y_both_test = np.column_stack((y_btc_test, y_eth_test))

# MultiOutputRegressor는 내부적으로 어떤 회귀모형을 사용할 것인지 입력받습니다.
# 여기서는 선형회귀를 사용합니다.
mlr = MultiOutputRegressor(LinearRegression())
mlr.fit(X_both_train, y_both_train)
y_pred_mlr_both = mlr.predict(X_both_test)
```

In [58]:
```
y_pred_mlr_both
```

Out[58]:
```
array([[-5.65032954e-05, -2.49664839e-05],
       [-3.82119184e-04, -9.30576399e-05],
       [-2.35028274e-04,  2.89037115e-04],
       ...,
       [-5.33463746e-05,  1.27324212e-04],
       [-2.31585278e-04,  1.34103503e-04],
       [-2.03453917e-04,  2.39642853e-04]])
```

In [59]:
```
print(y_pred_mlr_both.shape)
```

```
(41761, 2)
```

모델 평가

$$r_{XY} = \frac{\sum_i^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i^n (X_i - \bar{X})^2}\sqrt{\sum_i^n (Y_i - \bar{Y})^2}}$$

In [60]:
```
np.corrcoef(y_pred_lr_btc, y_btc_test)
```

Out[60]:
```
array([[ 1.        , -0.00983528],
       [-0.00983528,  1.        ]])
```

모델 평가

In [61]:

```
print('Test score for LR baseline: BTC', f"{np.corrcoef(y_pred_lr_btc, y_btc_test)[0,1]:.3f}",
                                  ', ETH', f"{np.corrcoef(y_pred_lr_eth, y_eth_test)[0,1]:.3f}")
print('Test score for multiple output LR baseline: BTC', f"{np.corrcoef(y_pred_mlr_both[:,0], y_
btc_test)[0,1]:.3f}",
                                                      ', ETH', f"{np.corrcoef(y_pred_mlr_both[:,1], y_
eth_test)[0,1]:.3f}")
```

```
Test score for LR baseline: BTC -0.010 , ETH 0.002
Test score for multiple output LR baseline: BTC -0.011 , ETH 0.010
```

소감

주가, 암호화폐 가격 예측은 수익적인 부분에서 많은 매력을 가지고 있기 때문에 회사에서도 연구 개발하고 있는 분야입니다. 그래서 제가 적용했던 선형회귀(Linear Regression)같은 간단한 머신러닝 모델로는 다양한 상황에 맞추어 예측을 해야하는 주가모델에서 점수가 낮게 나온것 같습니다.

최근 주가&암호화폐 예측에서는 딥러닝 모델을 다양하게 쓰이고 있습니다. LSTM와 같은 시계열 예측모델을 이용하여 더욱 정확성을 높이고 있습니다. 이번 과제를 진행하면서 모델을 배우고 적용하는 과정 속에서 데이터 마다 다양한 모델을 찾고 적용을 해야했고, 전처리에서도 많은 지식이 필요하다고 느꼈습니다.

Reference

Crypto-portfolio Optimization: Training-test split

(https://www.kaggle.com/code/federicominutoli/crypto-portfolio-optimization-training-test-split)

scikit-learn document

(https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html)