Project Report:

In this project we have implemented the Statistical arbitrage strategy proposed in Avellaneda and Lee 2010 on the universe of 40 crypto currencies sourced from FTX and analyzed the performance of the strategy

In this project we have used multiple classes and functions detailed below to implement the strategy:

- Initially we have calculated the factors that are responsible for the returns based on two principal components of PCA.

  returns into a systematic component $\beta_i F$ and an (uncorrelated) idiosyncratic component $\tilde{R}_i$. Alternatively, we consider multi-factor models of the form

$$R_i = \sum_{j=1}^{m} \beta_{ij} F_j + \tilde{R}_i. \qquad (5)$$

- After the factors are determined we have individually regressed it with stock returns and calculated the residual returns unexplained by the factors. The unexplained risk by factors is considered as the idiosyncratic component and assumed to be responsible for Alpha generation
- Also, we have auto regressed the cumulative residuals to find out the parameters that are used in the S score.

$$X_{n+1} = a + bX_n + \zeta_{n+1}, \quad n = 1, \ldots, 59.$$

According to (13), we have

$$a = m(1 - e^{-\kappa \Delta t})$$
$$b = e^{-\kappa \Delta t}$$
$$\text{Variance}(\zeta) = \sigma^2 \frac{1 - e^{-2\kappa \Delta t}}{2\kappa}$$

whence

$$\kappa = -\log(b) \times 252$$
$$m = \frac{a}{1-b}$$
$$\sigma = \sqrt{\frac{\text{Variance}(\zeta) \times 2\kappa}{1 - b^2}}$$
$$\sigma_{\text{eq}} = \sqrt{\frac{\text{Variance}(\zeta)}{1 - b^2}}.$$

$$s = \frac{X(t) - m}{\sigma_{\text{eq}}},$$

- Once the S scores are generated we have applied the signal function to check if a particular stock needs to be long or short or close (+1,-1,0) respectively.

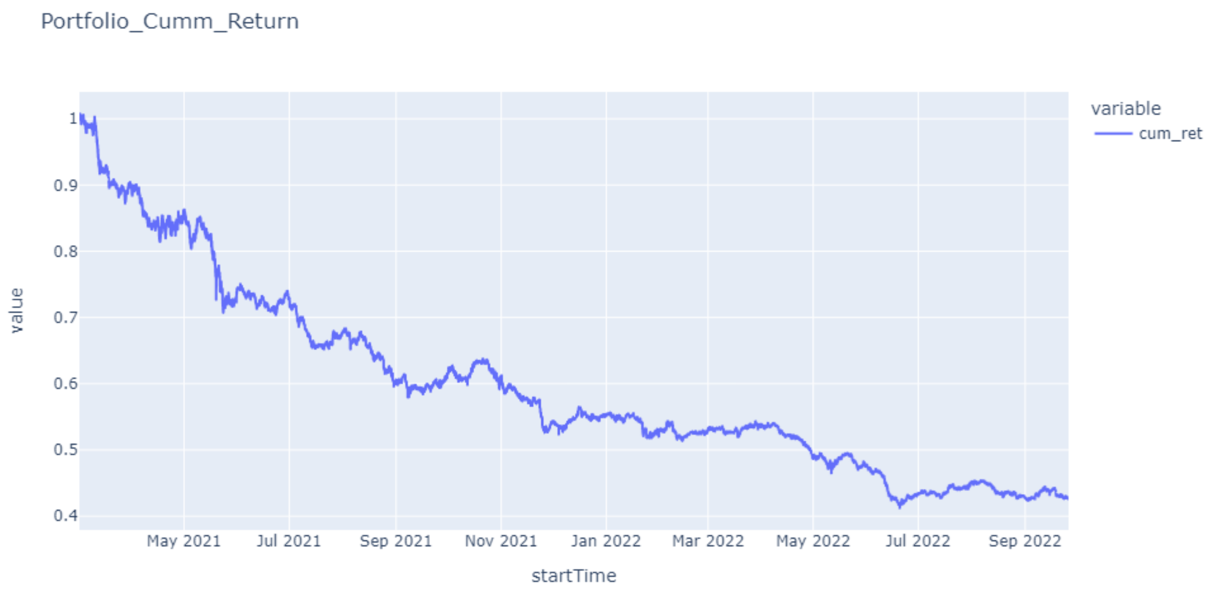
$$\bar{s}_{bo} = \bar{s}_{so} = 1.25$$
$$\bar{s}_{bc} = 0.75 \quad \text{and} \quad \bar{s}_{sc} = 0.50.$$

Our basic trading signal based on mean-reversion is

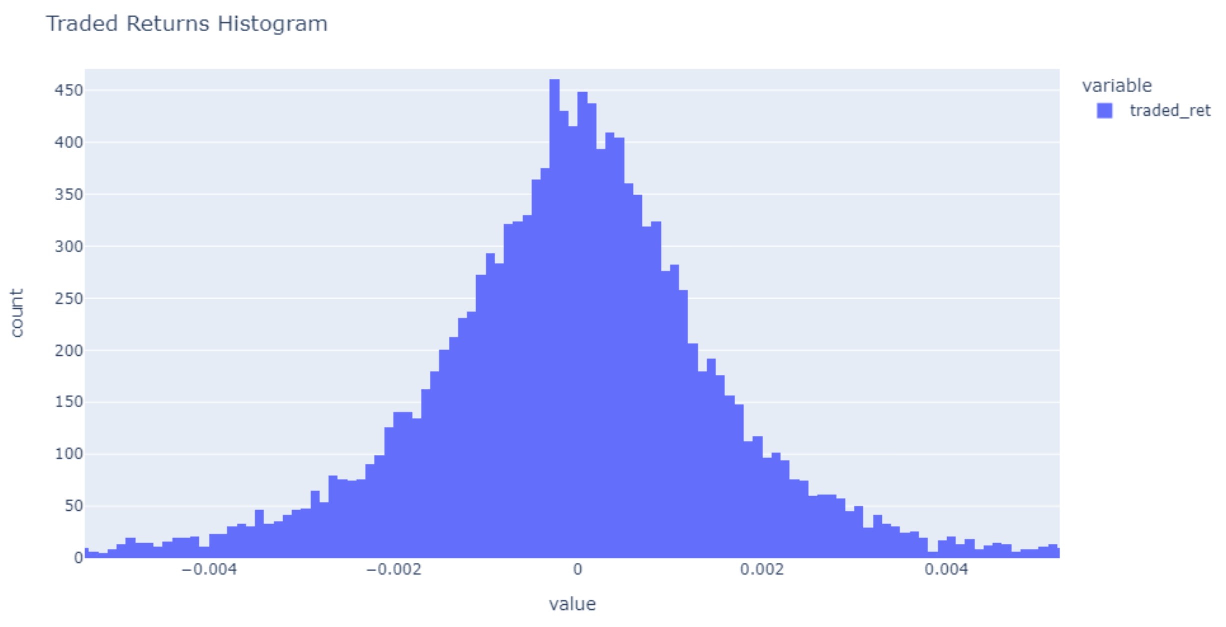
$$\begin{aligned} \text{buy to open if } s_i &< -\bar{s}_{bo} \\ \text{sell to open if } s_i &> +\bar{s}_{so} \\ \text{close short position if} s_i &< +\bar{s}_{bc} \\ \text{close long position } s_i &> -\bar{s}_{sc} \end{aligned} \qquad (16)$$

Based on the above brief overview of the flow of the implementation, we have traded in each hour based on the universe of the top market cap coins/tickers between Mar 2021 – Sep 2022 where we have generated signal based on the data of past 240 hours/10 days.

Cumulative return based on this strategy is -60%, if we have invested 1 dollar at the start, we would end up with 40 cents by the end of period. These results are in sync with market if we consider BTC as market benchmark (which fell from around $49,000 to around $19,000, refer appendix for the trend of BTC). Sharpe Ratio of -0.03143 with benchmark return considered as 0. Maximum Drawdown for the period is 256%.



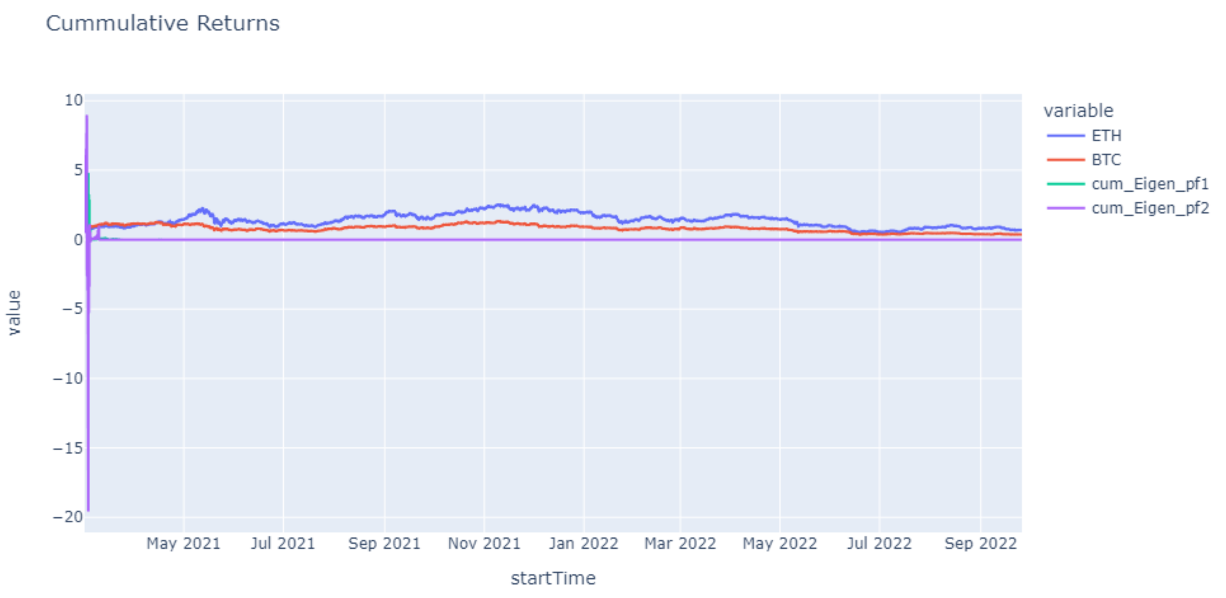
Return Distribution is negatively skewed with near normal distribution which is inline with the portfolio performance.



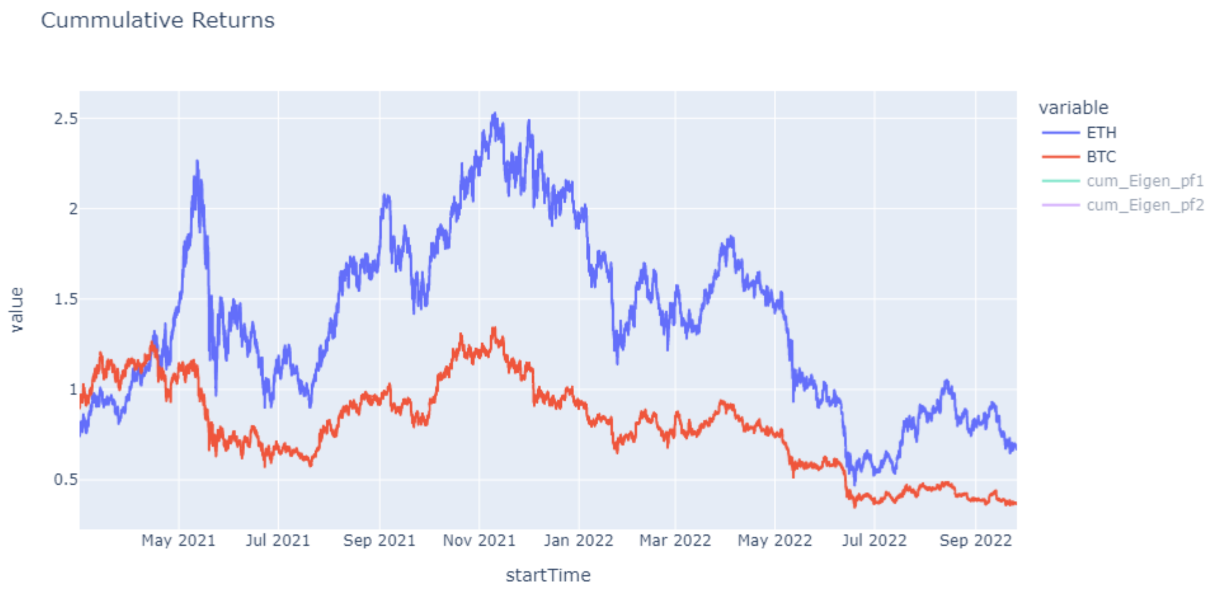
Results:

Plots of cumulative return curves of the 4 assets over the testing period in one figure.

In the below we had to scale the weights of the Eigen Portfolio in order to get the returns inbound

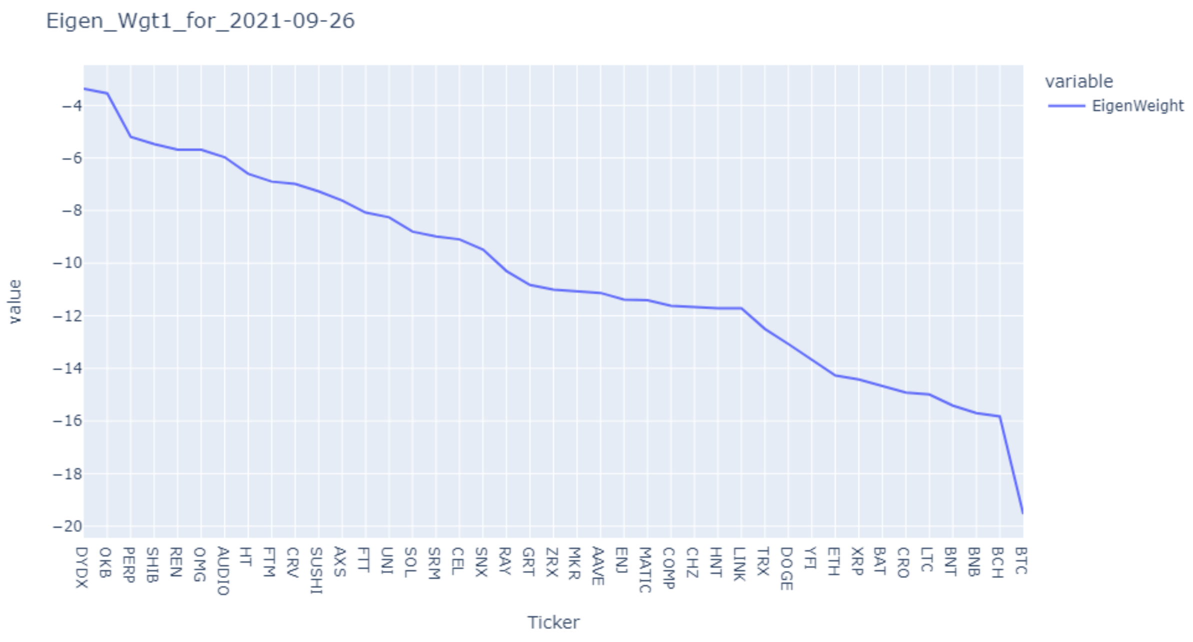


BTC, ETH cumulative return

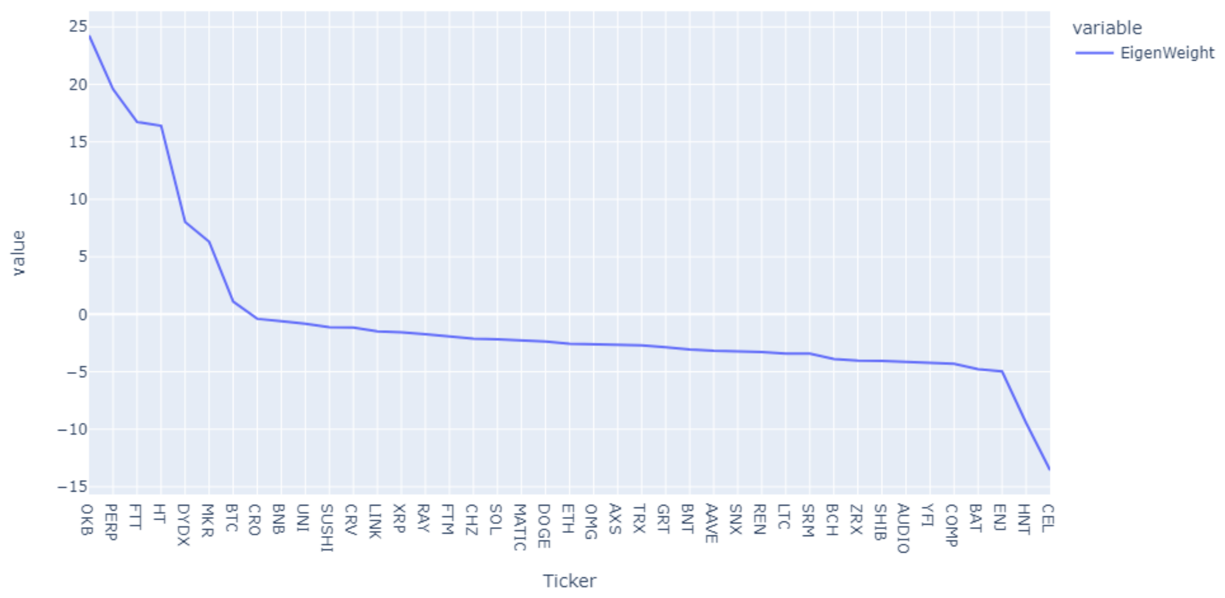


Plots of Eigen Vector weights in descending order for requested timepoints

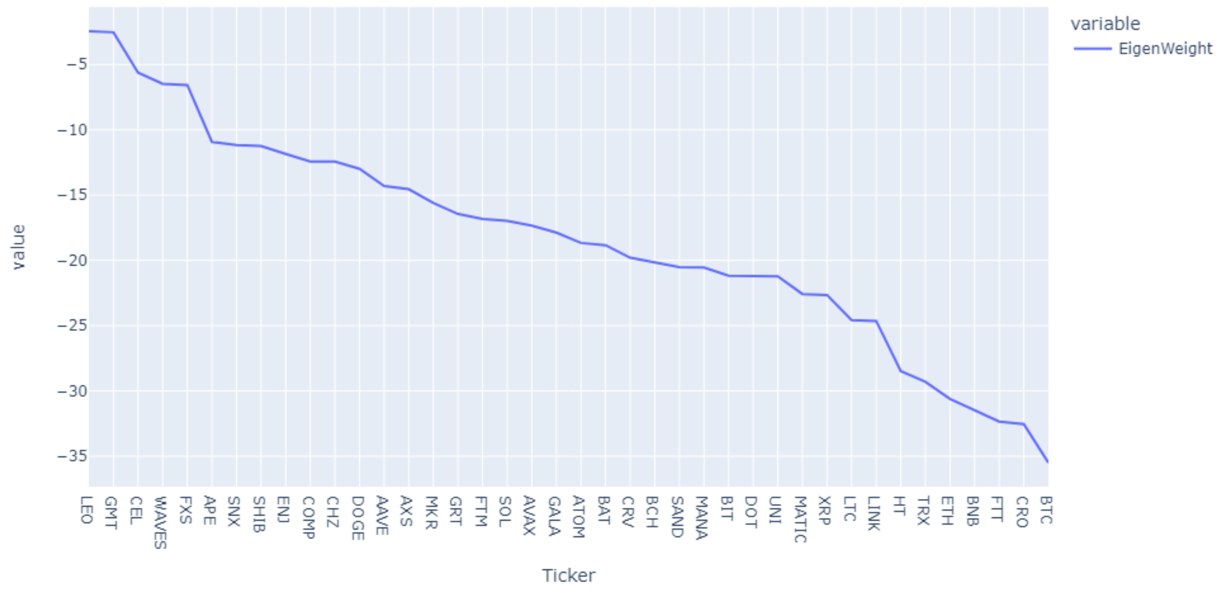
The first eigen portfolio weight curve is steep like a line, whereas the second eigen portfolio weights is in the form of titled curve

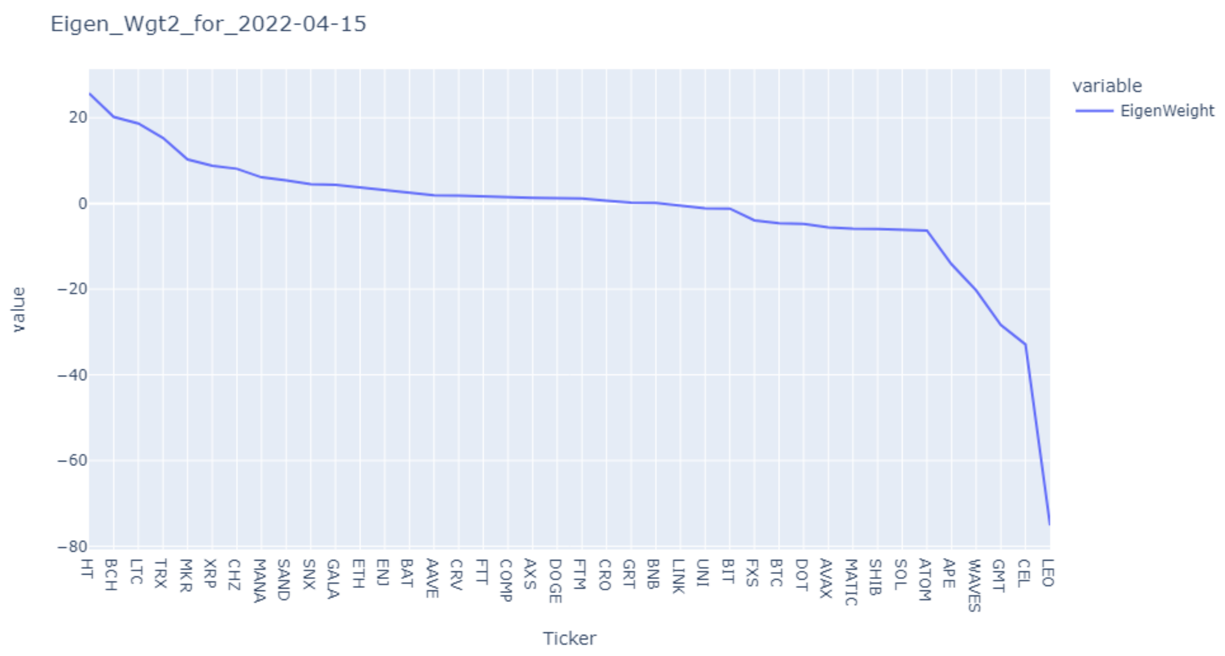


Eigen_Wgt2_for_2021-09-26



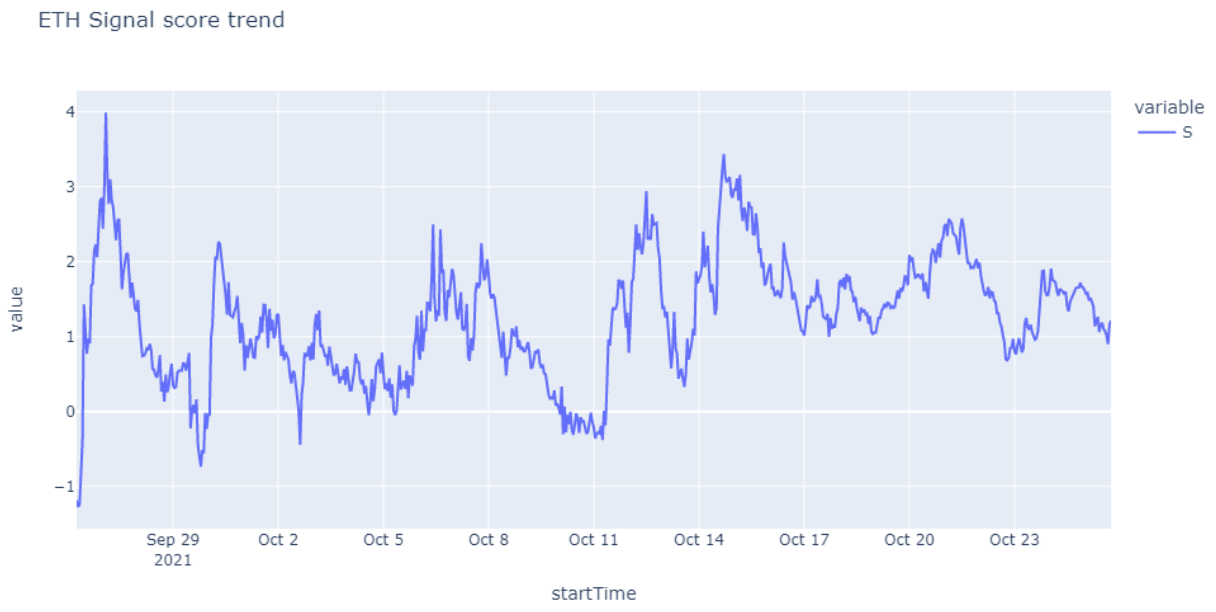
Eigen_Wgt1_for_2022-04-15

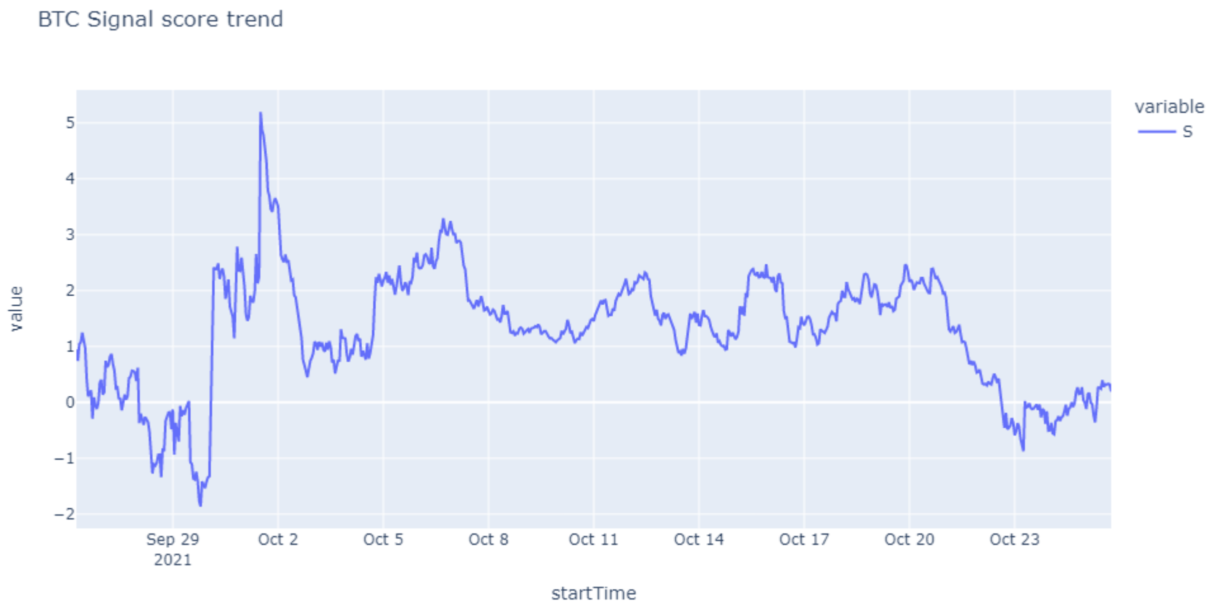




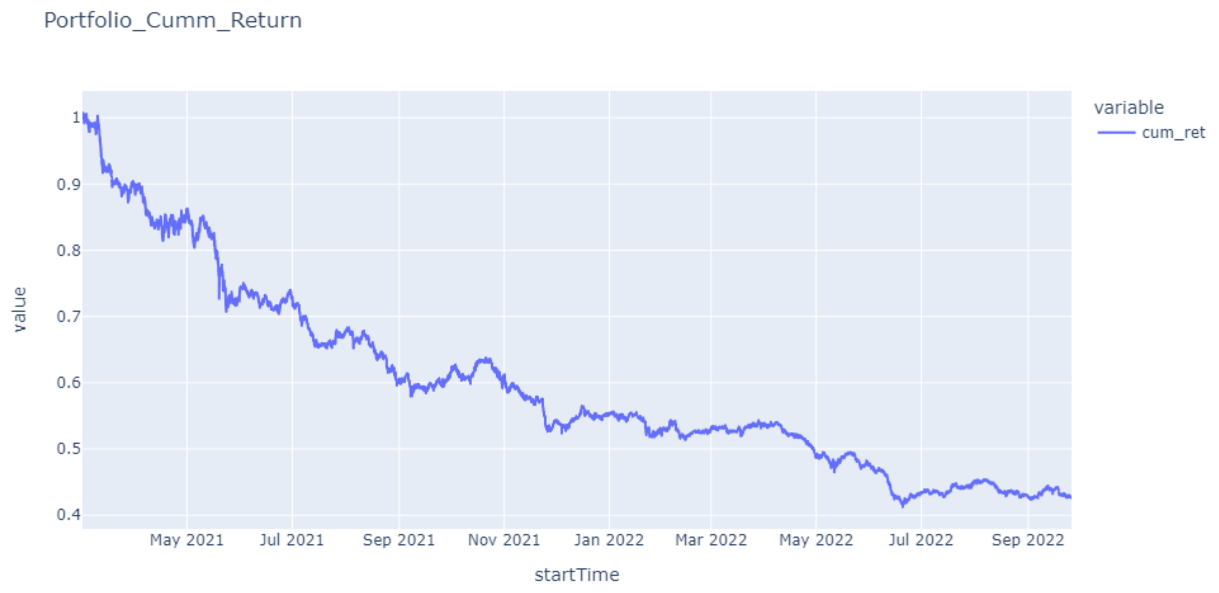
Eigen_Wgt2_for_2022-04-15
variable
EigenWeight
value
20
0
−20
−40
−60
−80
Ticker
HT
BCH
LTC
TRX
MKR
XRP
CHZ
MANA
SAND
SNX
GALA
ETH
ENJ
BAT
AAVE
CRV
FTT
COMP
AXS
DOGE
FTM
CRO
GRT
BNB
LINK
UNI
BIT
FXS
BTC
DOT
AVAX
MATIC
SHIB
SOL
ATOM
APE
WAVES
GMT
CEL
LEO

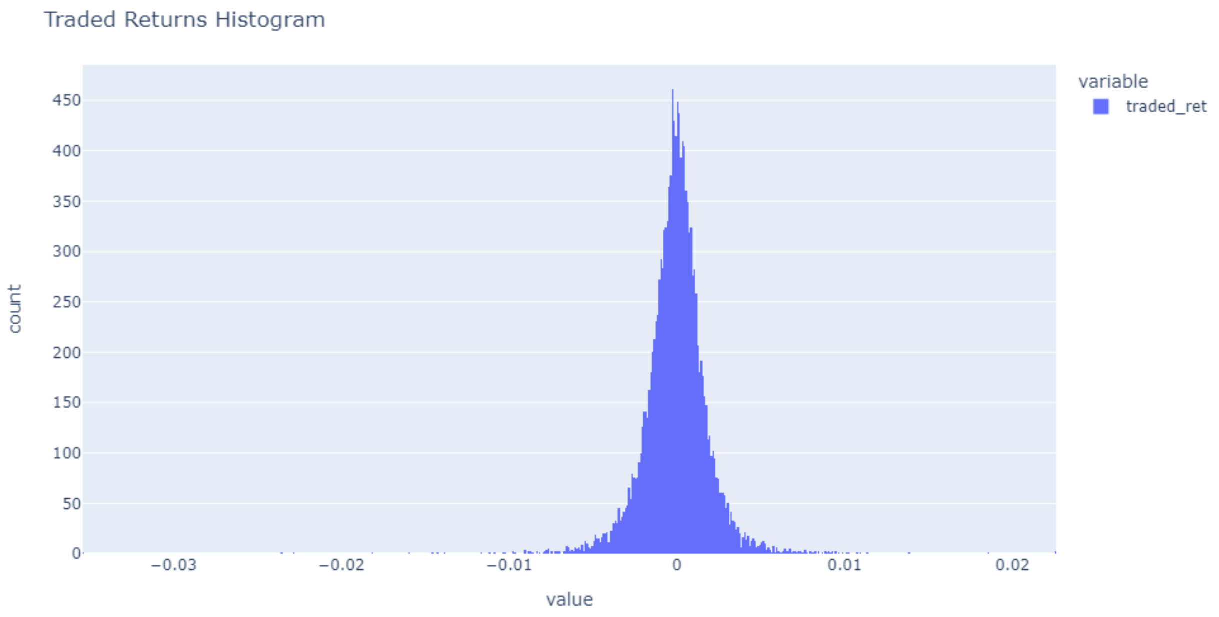
Plot the evolution of s-score of BTC and ETH from 2021-09-26 00:00:00 to 2021-10-25 23:00:00



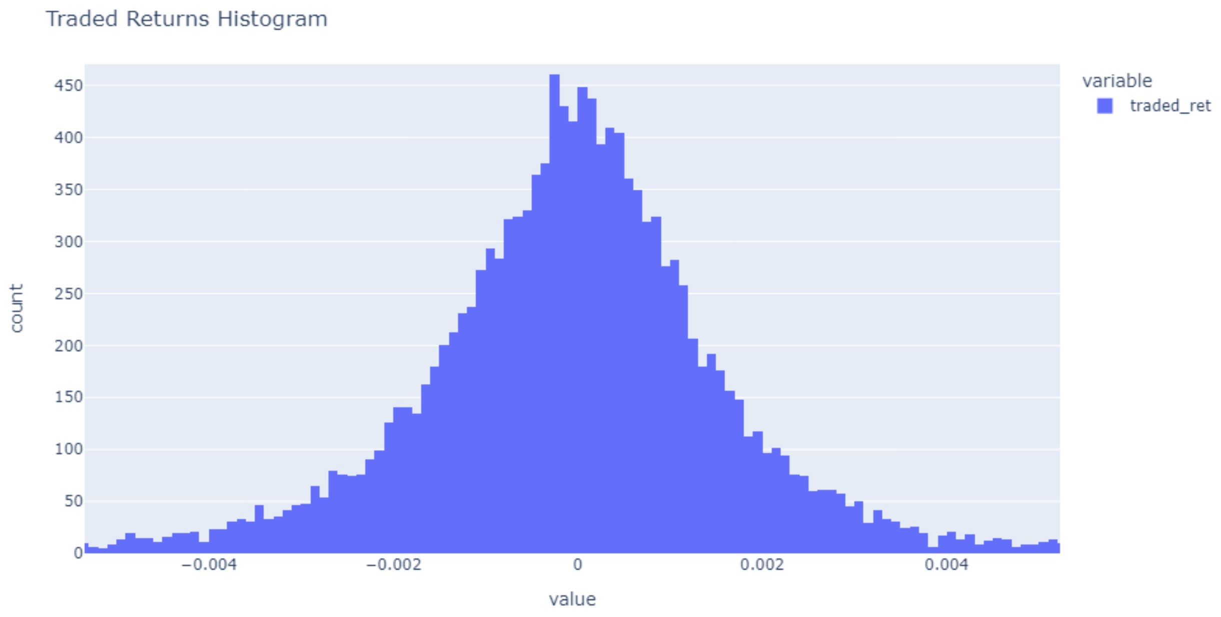


Plots of cumulative return and traded return distribution

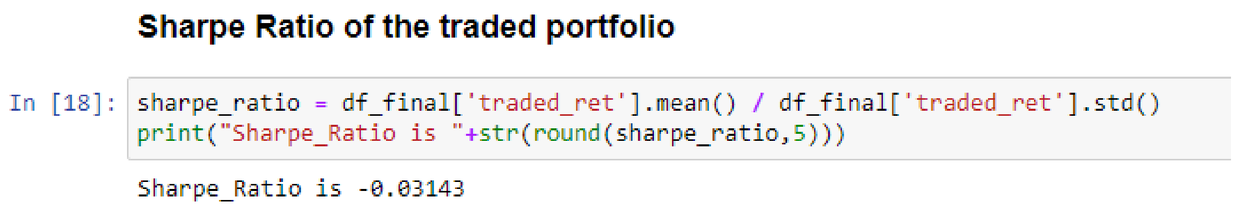




Zoomed image of histogram:



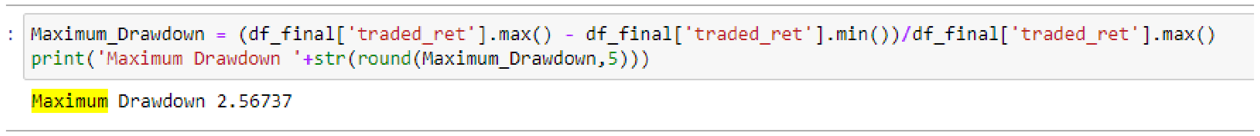
Sharpe Ratio: -0.03143 is the Sharpe ratio of the portfolio based on traded signals



Sharpe Ratio of the traded portfolio

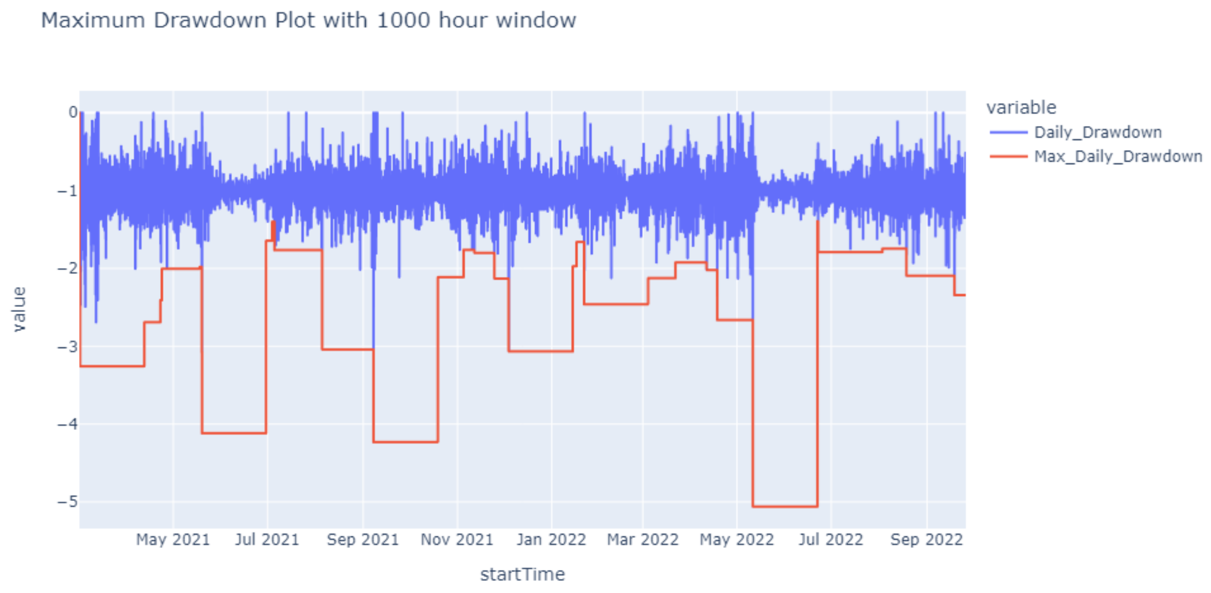
```
In [18]: sharpe_ratio = df_final['traded_ret'].mean() / df_final['traded_ret'].std()
         print("Sharpe_Ratio is "+str(round(sharpe_ratio,5)))

         Sharpe_Ratio is -0.03143
```

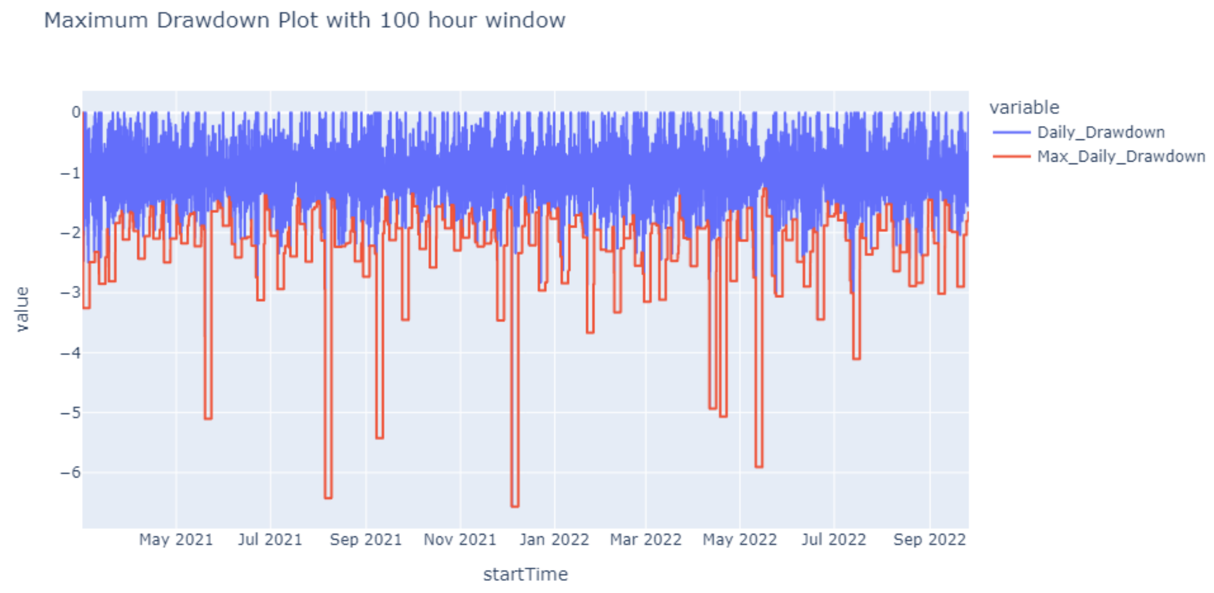
Maximum Drawdown: 2.567 i.e. 256%

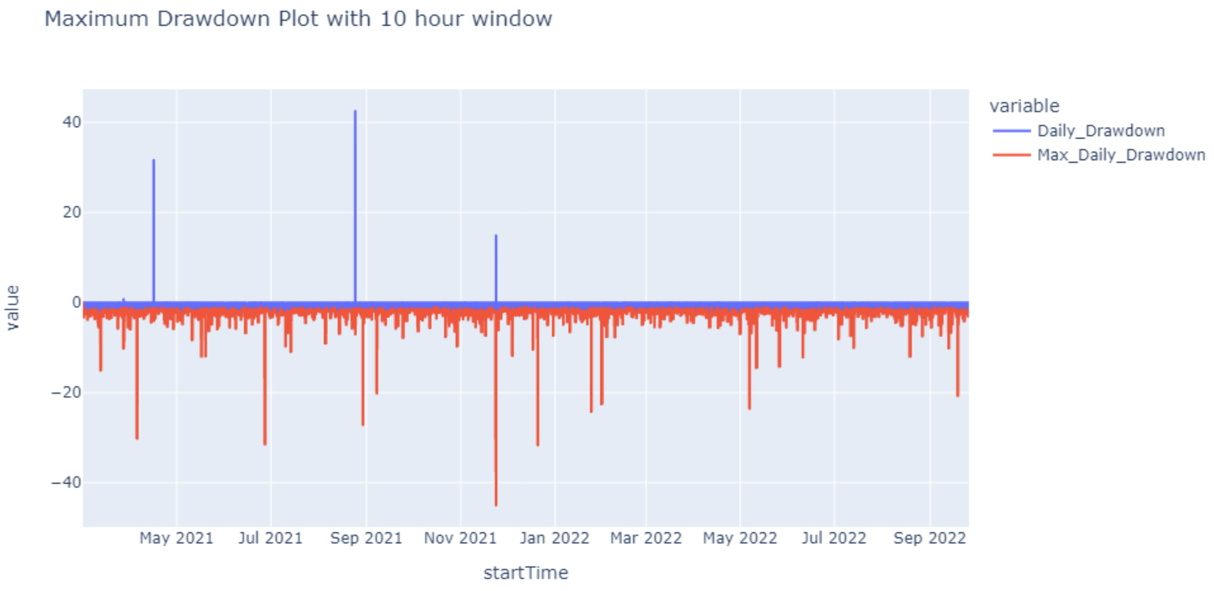


```
: Maximum_Drawdown = (df_final['traded_ret'].max() - df_final['traded_ret'].min())/df_final['traded_ret'].max()
  print('Maximum Drawdown '+str(round(Maximum_Drawdown,5)))

  Maximum Drawdown 2.56737
```

Below are the plots of Maximum Drawdown for various rolling windows


Maximum Drawdown Plot with 1000 hour window
variable
Daily_Drawdown
Max_Daily_Drawdown
value
0
−1
−2
−3
−4
−5
May 2021
Jul 2021
Sep 2021
Nov 2021
Jan 2022
Mar 2022
May 2022
Jul 2022
Sep 2022
startTime





Code Implementation:

We have loaded top 40 tickers each hour and price data in the dataframe in df_price and df_ticker dataframes respectively. Then we have created a column in df_ticker (ticker_list) which contains list of all the top 40 tickers and merged both the dataframes.

We have iterated over all the rows in merged dataframe created in above step to perform all the below tasks on all the timepoints.

Inside loop we are creating new dataframe for each timepoint by storing data pertaining to only top 40 tickers in that timepoint, then we are creating a time difference column which is basically giving difference between that timepoint and time of data each row is storing and then we are only keeping the data where this difference is less than 240 and equal or more than 240.

The dataframe generated above are created for each datapoint and stored in a dictionary "dict1".

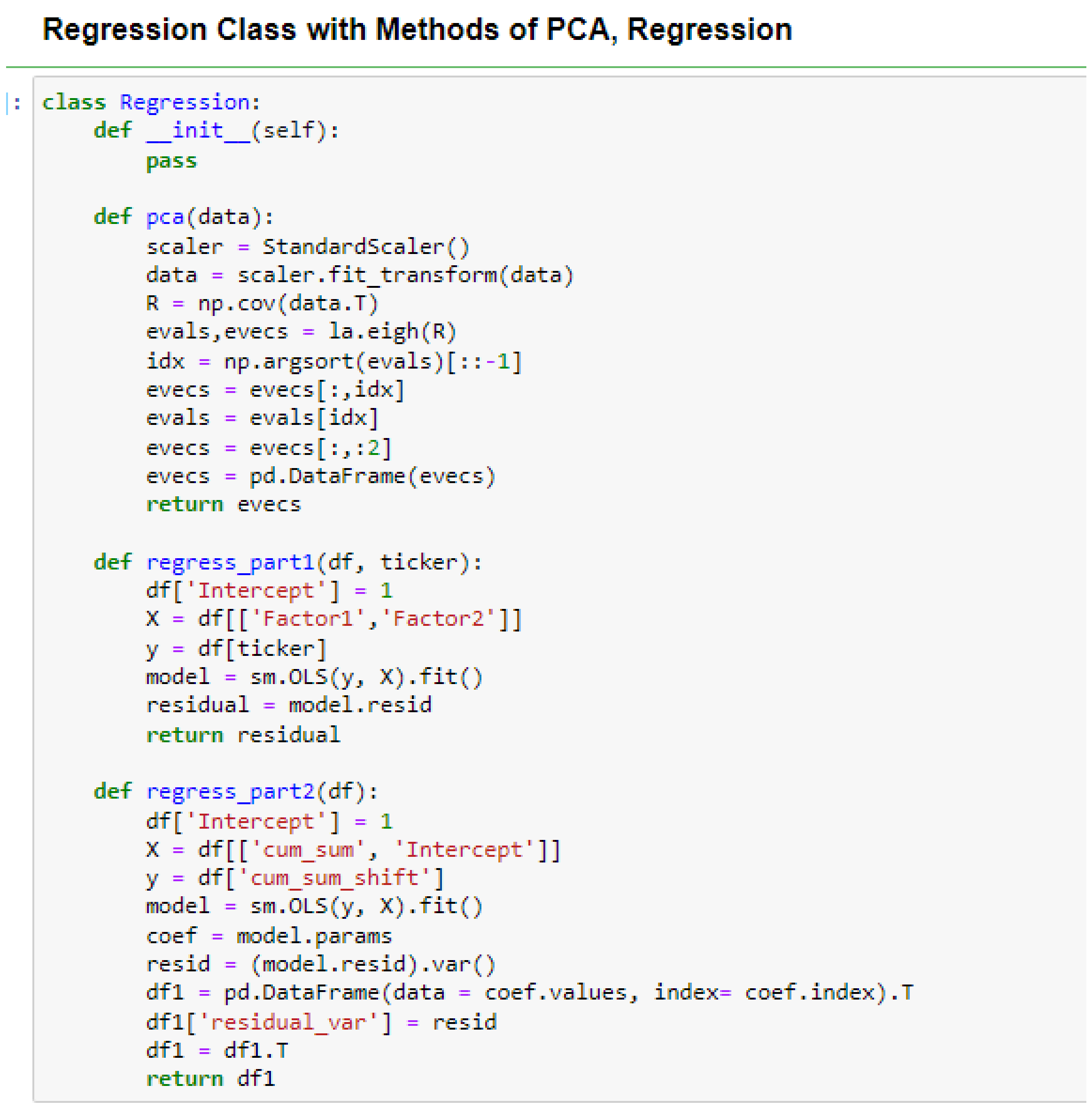
We have calculated daily returns of each ticker in the "data" dataframe.

Classes & Function:

1. **Regression Class:**
   - **PCA:** The function takes dataframe containing returns of all the tickers and perform principal component analysis and returns 2 eigen vectors (evecs) as output dataframe.

   - **Regression Part 1:** The function takes dataframe (containing ticker returns and eigen portfolio returns) and ticker as input and performs regression of portfolio return of eigen portfolios with return of each ticker and it gives residual error series (residual) as output.

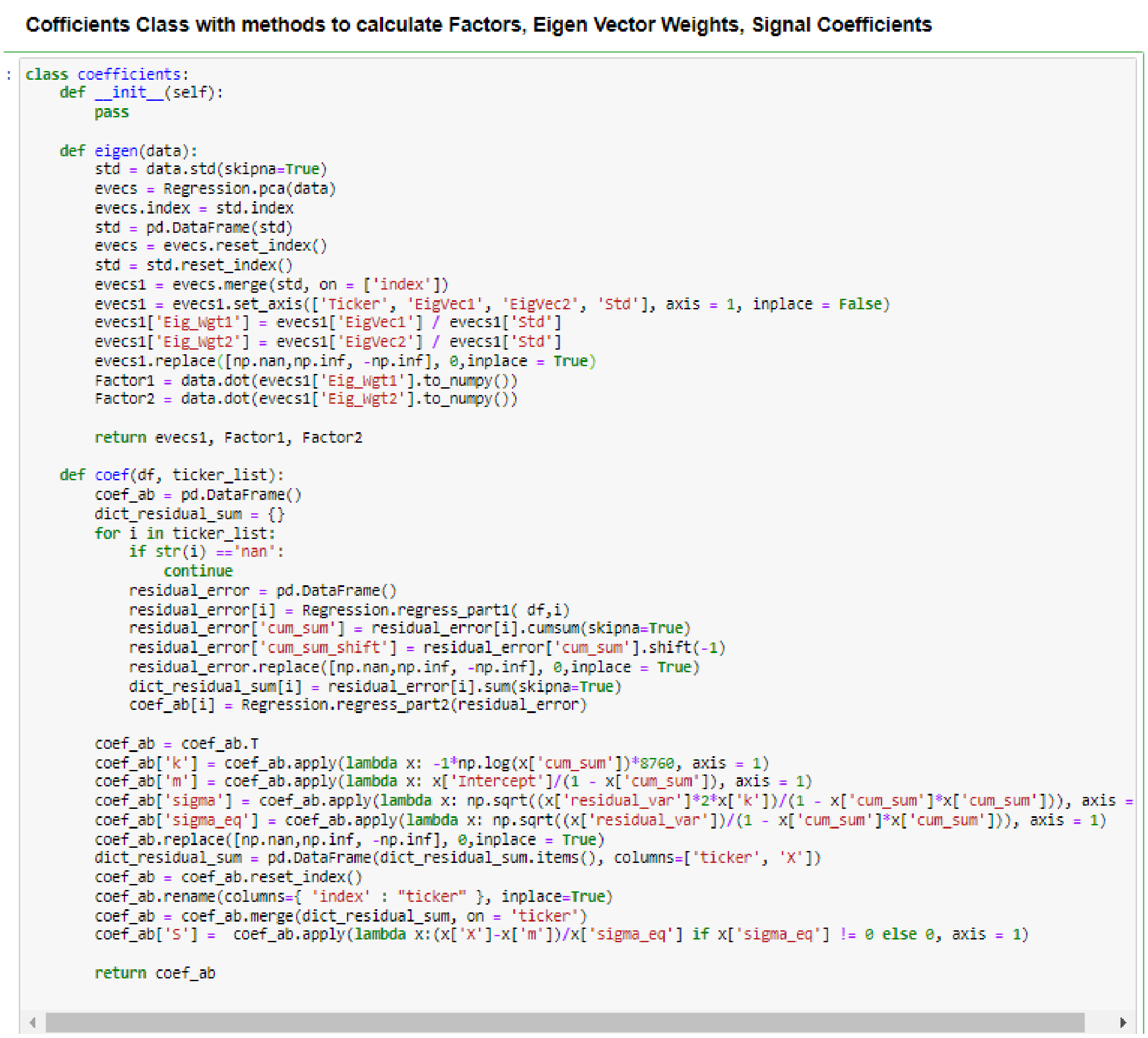
   - **Regression Part 2:** The function takes dataframe (containing cumulative sum of residual error with shifted cumulative sum of residual error) performs regression of cumulative sum of residual error with shifted cumulative sum of residual error and it gives variance of residual error (residual_var), regression beta (b) and intercept (a) as output dataframe.



Regression Class with Methods of PCA, Regression

```
class Regression:
    def __init__(self):
        pass

    def pca(data):
        scaler = StandardScaler()
        data = scaler.fit_transform(data)
        R = np.cov(data.T)
        evals,evecs = la.eigh(R)
        idx = np.argsort(evals)[::-1]
        evecs = evecs[:,idx]
        evals = evals[idx]
        evecs = evecs[:,:2]
        evecs = pd.DataFrame(evecs)
        return evecs

    def regress_part1(df, ticker):
        df['Intercept'] = 1
        X = df[['Factor1','Factor2']]
        y = df[ticker]
        model = sm.OLS(y, X).fit()
        residual = model.resid
        return residual

    def regress_part2(df):
        df['Intercept'] = 1
        X = df[['cum_sum', 'Intercept']]
        y = df['cum_sum_shift']
        model = sm.OLS(y, X).fit()
        coef = model.params
        resid = (model.resid).var()
        df1 = pd.DataFrame(data = coef.values, index= coef.index).T
        df1['residual_var'] = resid
        df1 = df1.T
        return df1
```

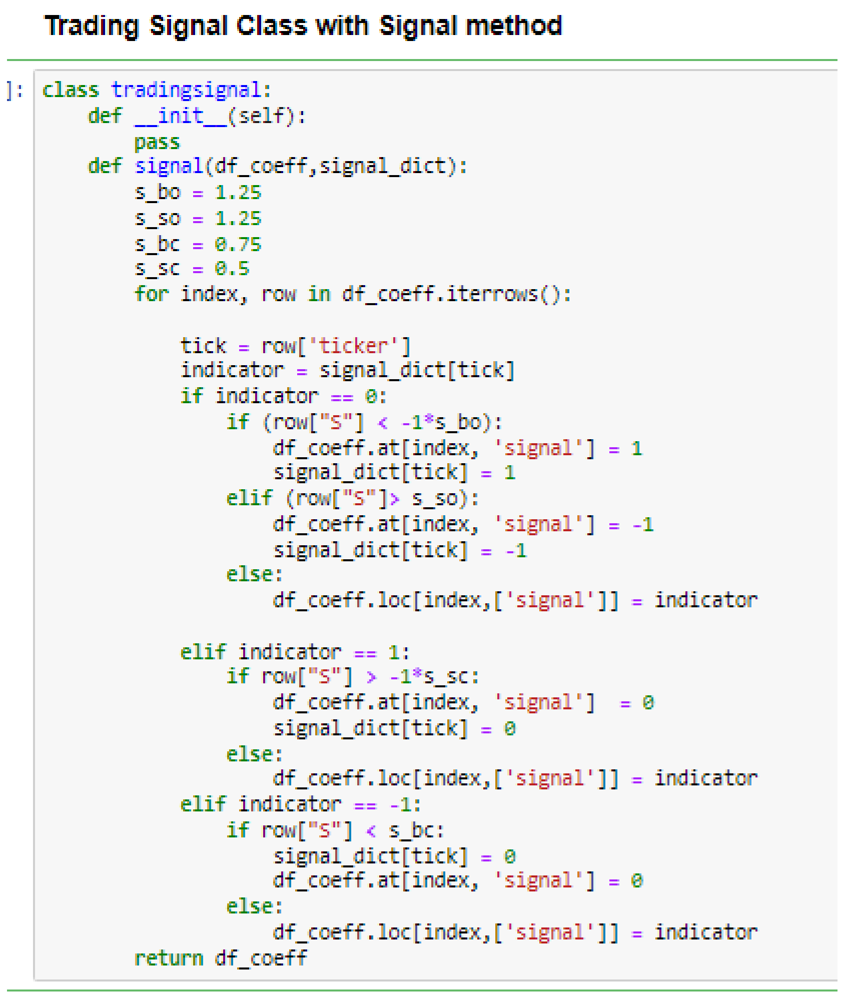
2. **Coefficient Class:**
    - **Eigen:** The function takes dataframe containing returns of each ticker and calls PCA function which returns 2 top eigen vectors, then we are calculating standard deviation of returns for each ticker and dividing values of eigen vectors with standard deviation to get weights of eigen portfolio.
    - **Coef:** Coef function takes dataframe of returns and ticker list as input and calls regress_part1 function with gives residual series which is then modified by adding 2 additional columns cumulative sum of residual error with shifted cumulative sum of residual error. The modified dataframe is then passed to regress_part2 which gives regression alpha, beta, and variance of residual error. The results are then utilized to calculate k, m, sigma, sigma_eq and s-score.



Cofficients Class with methods to calculate Factors, Eigen Vector Weights, Signal Coefficients

```
class coefficients:
    def __init__(self):
        pass

    def eigen(data):
        std = data.std(skipna=True)
        evecs = Regression.pca(data)
        evecs.index = std.index
        std = pd.DataFrame(std)
        evecs = evecs.reset_index()
        std = std.reset_index()
        evecs1 = evecs.merge(std, on = ['index'])
        evecs1 = evecs1.set_axis(['Ticker', 'EigVec1', 'EigVec2', 'Std'], axis = 1, inplace = False)
        evecs1['Eig_Wgt1'] = evecs1['EigVec1'] / evecs1['Std']
        evecs1['Eig_Wgt2'] = evecs1['EigVec2'] / evecs1['Std']
        evecs1.replace([np.nan,np.inf, -np.inf], 0,inplace = True)
        Factor1 = data.dot(evecs1['Eig_Wgt1'].to_numpy())
        Factor2 = data.dot(evecs1['Eig_Wgt2'].to_numpy())

        return evecs1, Factor1, Factor2

    def coef(df, ticker_list):
        coef_ab = pd.DataFrame()
        dict_residual_sum = {}
        for i in ticker_list:
            if str(i) =='nan':
                continue
            residual_error = pd.DataFrame()
            residual_error[i] = Regression.regress_part1( df,i)
            residual_error['cum_sum'] = residual_error[i].cumsum(skipna=True)
            residual_error['cum_sum_shift'] = residual_error['cum_sum'].shift(-1)
            residual_error.replace([np.nan,np.inf, -np.inf], 0,inplace = True)
            dict_residual_sum[i] = residual_error[i].sum(skipna=True)
            coef_ab[i] = Regression.regress_part2(residual_error)

        coef_ab = coef_ab.T
        coef_ab['k'] = coef_ab.apply(lambda x: -1*np.log(x['cum_sum'])*8760, axis = 1)
        coef_ab['m'] = coef_ab.apply(lambda x: x['Intercept']/(1 - x['cum_sum']), axis = 1)
        coef_ab['sigma'] = coef_ab.apply(lambda x: np.sqrt((x['residual_var']*2*x['k'])/(1 - x['cum_sum']*x['cum_sum'])), axis =
        coef_ab['sigma_eq'] = coef_ab.apply(lambda x: np.sqrt((x['residual_var'])/(1 - x['cum_sum']*x['cum_sum'])), axis = 1)
        coef_ab.replace([np.nan,np.inf, -np.inf], 0,inplace = True)
        dict_residual_sum = pd.DataFrame(dict_residual_sum.items(), columns=['ticker', 'X'])
        coef_ab = coef_ab.reset_index()
        coef_ab.rename(columns={ 'index' : "ticker" }, inplace=True)
        coef_ab = coef_ab.merge(dict_residual_sum, on = 'ticker')
        coef_ab['S'] =  coef_ab.apply(lambda x:(x['X']-x['m'])/x['sigma_eq'] if x['sigma_eq'] != 0 else 0, axis = 1)

        return coef_ab
```

3. **Trading Signal:**
    - **Signal:** The function takes dataframe containing coefficients (a, b, residual_var, k, m, sigma, sigma_eg, s-score etc. returned by Coef function) and uses s-score to determine whether to long or short stock and checks if a long or short positions needs to be closed. This is accomplished by a signal dictionary we have created which keeps track of our position on every ticker.



Trading Signal Class with Signal method

```
]: class tradingsignal:
       def __init__(self):
           pass
       def signal(df_coeff,signal_dict):
           s_bo = 1.25
           s_so = 1.25
           s_bc = 0.75
           s_sc = 0.5
           for index, row in df_coeff.iterrows():

               tick = row['ticker']
               indicator = signal_dict[tick]
               if indicator == 0:
                   if (row["S"] < -1*s_bo):
                       df_coeff.at[index, 'signal'] = 1
                       signal_dict[tick] = 1
                   elif (row["S"]> s_so):
                       df_coeff.at[index, 'signal'] = -1
                       signal_dict[tick] = -1
                   else:
                       df_coeff.loc[index,['signal']] = indicator

               elif indicator == 1:
                   if row["S"] > -1*s_sc:
                       df_coeff.at[index, 'signal']  = 0
                       signal_dict[tick] = 0
                   else:
                       df_coeff.loc[index,['signal']] = indicator
               elif indicator == -1:
                   if row["S"] < s_bc:
                       signal_dict[tick] = 0
                       df_coeff.at[index, 'signal'] = 0
                   else:
                       df_coeff.loc[index,['signal']] = indicator
           return df_coeff
```

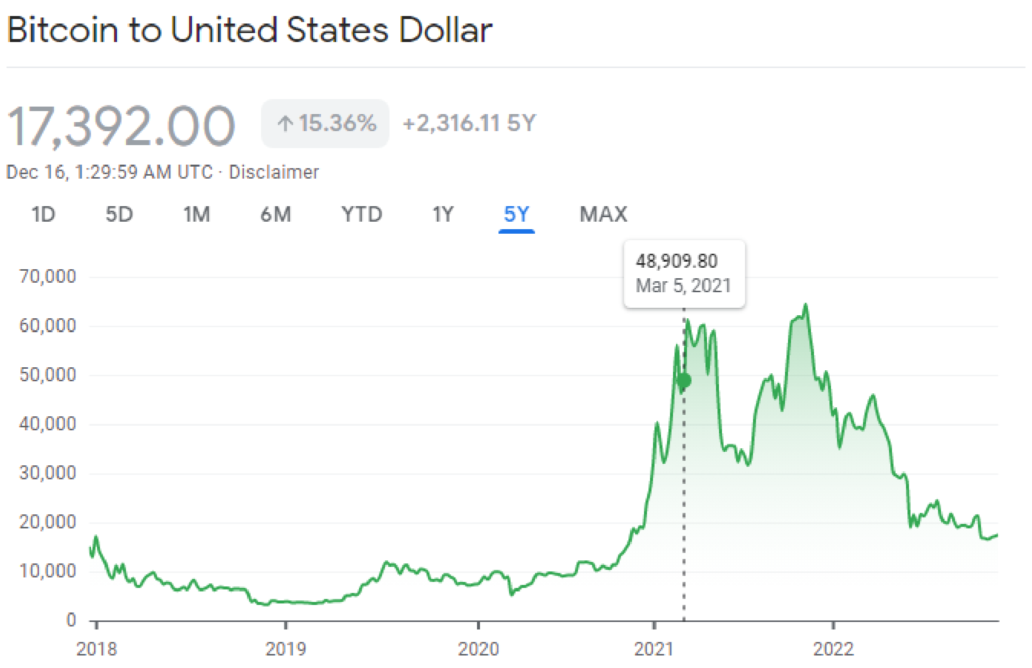
4. **Func:**
   - **time_plot:** This function plots the dataframe using plotly.
   - **Eigenweights:** This function plots the eigen weights in descending order for a particular data.
   - **Hist:** This function plots the histogram distribution for the data.

func class for plots methods

]:

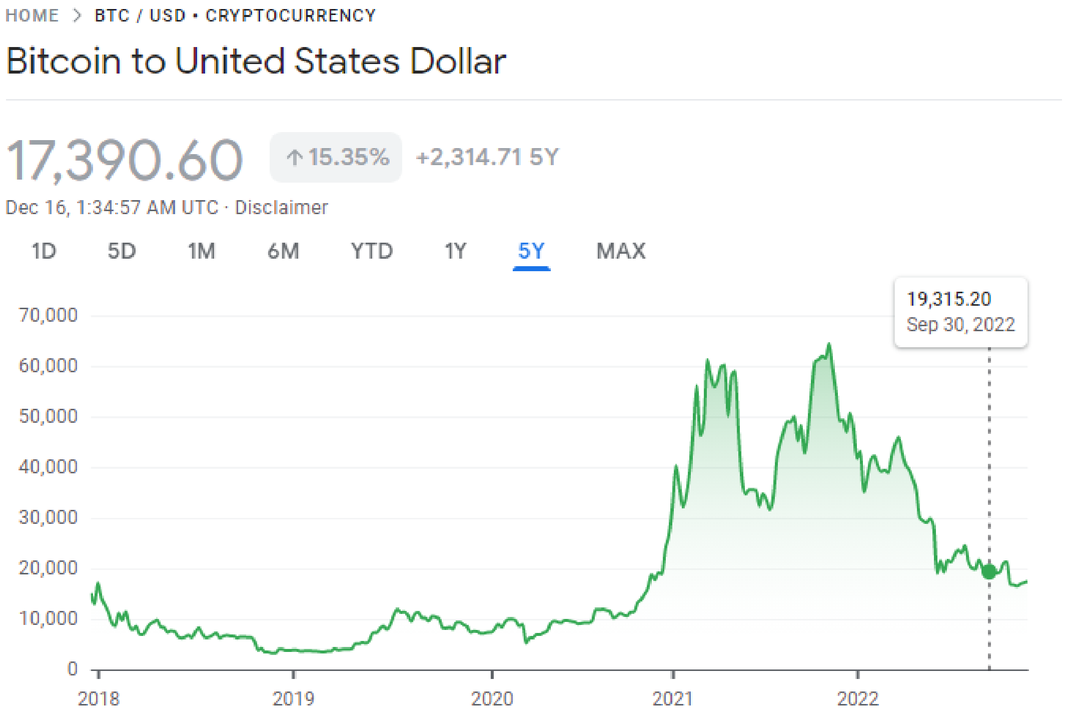
```
class func:
    def __init__(self):
        pass
    def time_plot(data, name):
        data = pd.melt(data, id_vars='startTime', value_vars=data.columns[:])
        fig = px.line(data, x='startTime', y='value', color='variable',title = name)
        fig.show()
    def eigenweights(df_eigen, eigen_weight,date,title):
        df_eigenWeight1 = df_eigen[df_eigen['index'] == eigen_weight]
        df_eigenWeight1 = df_eigenWeight1[df_eigenWeight1['startTime'] == date]
        del df_eigenWeight1['startTime'], df_eigenWeight1['index']
        df_eigenWeight1 = df_eigenWeight1.T
        df_eigenWeight1.rename(columns = {df_eigenWeight1.columns[0]:'EigenWeight'}, inplace = True)
        df_eigenWeight1 = df_eigenWeight1.sort_values(by = ['EigenWeight'], ascending=False)
        df_eigenWeight1.dropna(inplace = True)
        fig = px.line(df_eigenWeight1)
        fig.update_layout(
            title_text=title, # title of plot
        )
        fig.show()
    def hist(data,title):
        fig = px.histogram(data)
        fig.update_layout(
            title_text=title, # title of plot
        )
        fig.show()
```

Appendix:


Bitcoin to United States Dollar

17,392.00 ↑15.36% +2,316.11 5Y

Dec 16, 1:29:59 AM UTC · Disclaimer


HOME > BTC / USD • CRYPTOCURRENCY

Bitcoin to United States Dollar

17,390.60 ↑15.35% +2,314.71 5Y

Dec 16, 1:34:57 AM UTC · Disclaimer