```
import os
print(os.getcwd())  # This will print your current working directory


```

/content

```
# from google.colab import drive
# drive.mount('/content/drive',force_remount=True)

```

```
Author = 'Conrad'
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

from datetime import datetime
import time
import pytz
local_timezone = 'America/Los_Angeles'
from time import gmtime, strftime

from scipy import stats
import numpy as np
import pandas as pd
import pickle
#from pandas.plotting import scatter_matrix
import xarray as xr
from pandas import Series
import seaborn as sns
from IPython.display import display # Allows the use of display() for DataFrames
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15.0, 8.0)

import yfinance as yf

from sklearn import preprocessing
from sklearn.preprocessing import FunctionTransformer, maxabs_scale, MinMaxScaler, Binarizer, StandardScaler


from sklearn.decomposition import FastICA, PCA
from sklearn.feature_selection import SelectKBest, chi2, f_regression, mutual_info_regression, r_regression

from sklearn import cluster, tree
from sklearn.linear_model import LinearRegression, Ridge, ElasticNet, Lasso, ElasticNetCV,HuberRegressor, RidgeCV
from lightgbm import LGBMClassifier as lgb

#from sklearn.linear_model import LinearRegression, Ridge, ElasticNet, Lasso, HuberRegressorm
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor

from sklearn import mixture
from sklearn.preprocessing import PowerTransformer

from sklearn.metrics import mean_squared_error as rmse
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import explained_variance_score as evs
from sklearn.metrics import r2_score as r_squared
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

from sklearn.pipeline import Pipeline, make_pipeline, clone
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin
from scipy.stats.mstats import winsorize
from scipy.stats import norm
from statsmodels.tools import add_constant
from sklearn.dummy import DummyRegressor

import statsmodels.api as sm
from statsmodels.regression.linear_model import OLS
#import lightgbm as lgb

last_printed_time = datetime.now()

from sklearn import set_config
set_config(display="diagram")

zscaler = StandardScaler().set_output(transform='pandas')

from sklearn.multioutput import MultiOutputRegressor

```

```
# import sys
# sys.path.append('drive/content/drive/My Drive/Colab_Notebooks/QuantEasy/')
# #
```

```
#sys.path
```

```

def simplify_teos(df):
```

```
    " convert 'datetime64[ns, UTC]' to 'datetime64[ns, UTC]' "
    df.index = df.index.tz_localize(None).normalize()
    return df

def log_returns(df):
    df = np.log(df)
    df = df - df.shift(1)
    #df.index.name = None
    return df

def L_func_2(df,pred_col='predicted',params=[]):
    t_conditions = [ df[pred_col] <= 0, df[pred_col] > 0 ]
    t_positions =  [ params[0],  params[1] ]
    return np.select(t_conditions,t_positions, default=np.nan) # Apply trading logic here

def L_func_3(df, pred_col='preds_index', params=[]):
    t_conditions = [
        (df[pred_col].between(0.00, 0.25)),
        (df[pred_col].between(0.25, 0.50)),
        (df[pred_col].between(0.50, 0.75)),
        (df[pred_col].between(0.75, 1.00))
    ]
    t_positions = params
    return np.select(t_conditions, t_positions, default=np.nan)

def L_func_4(ds, params=[]):
    t_conditions = [
        (ds.between(0.00, 0.25)),
        (ds.between(0.25, 0.50)),
        (ds.between(0.50, 0.75)),
        (ds.between(0.75, 1.00))
    ]
    t_positions = params
    return np.select(t_conditions, t_positions, default=np.nan)

def sim_stats(regout_list,sweep_tags,author='CG',trange = None):
    df = pd.DataFrame()
    df.index.name = 'teo'
    print('SIMULATION RANGE: ','from ',trange.start,'to ', trange.stop)
    for n, testlabel in enumerate(sweep_tags):
        reg_out = regout_list[n].loc[trange,:]
        df.loc['return',testlabel] = mean = 252*reg_out.perf_ret.mean()
        df.loc['stdev',testlabel] = std = (np.sqrt(252))*reg_out.perf_ret.std()
        df.loc['sharpe',testlabel] = mean / std
        df.loc['avg_beta',testlabel] = reg_out.leverage.mean()
        df.loc['beta_1_return',testlabel] = df.loc['return',testlabel] / reg_out.leverage.mean()
        df.loc['pos_bet_ratio',testlabel] = np.sum(np.isfinite(reg_out['prediction']) & (reg_out['prediction'] > 0)) / np.sum(np.isfinite(reg_out['prediction']))
        df.loc['rmse',testlabel] = np.sqrt(rmse(reg_out.prediction,reg_out.actual))
        df.loc['mae',testlabel] = mae(reg_out.prediction,reg_out.actual)
        #df.loc['evs',testlabel] = evs(reg_out.prediction,reg_out.actual)
        df.loc['r2',testlabel] = r_squared(reg_out.actual,reg_out.prediction)
        df.loc['benchmark return',testlabel] = bench_ret = 252*reg_out.actual.mean()
        df.loc['benchmark std',testlabel] = bench_std = (np.sqrt(252))*reg_out.actual.std()
        df.loc['benchmark sharpe',testlabel] = bench_ret / bench_std
        df.loc['beg_pred',testlabel] = min(reg_out.prediction.index).date()
        df.loc['end_pred',testlabel] = max(reg_out.prediction.index).date()
        #df.loc['train window',testlabel] = window=cal['regression_est_window']
        #df.loc['z_score_window',testlabel] = cal['z_score_window']
        df.loc['sim_time',testlabel] =  datetime.now(pytz.timezone(local_timezone)).strftime("%x %-I:%-m%p")
        df.loc['author',testlabel] = author
    return df

def p_by_slice(X,y,t_list,t_list_labels):
    feat_stats = pd.DataFrame(index=X.columns)

    for n,idx in enumerate(t_list):
        X_fit = X.loc[idx,:].dropna()
        y_fit = y.reindex(X_fit.index)
        feat_stats.loc[:,t_list_labels[n]] = r_regression(X_fit, y_fit,center=True)

    print('from ',X_fit.index.min(),' to ',X_fit.index.max())
    return feat_stats

def p_by_year(X,y,sort_by = 'p_value',t_list=None):
    feat_stats = pd.DataFrame(index=X.columns)

    for year in X.index.year.unique():
        X_fit = X.loc[str(year),:].dropna()
        y_fit = y.reindex(X_fit.index)
        feat_stats.loc[:,str(year)] = r_regression(X_fit, y_fit,center=True)

    print('from ',X_fit.index.min(),' to ',X_fit.index.max())
    return feat_stats

def feature_profiles(X,y,sort_by = 'pearson',t_slice=None):
    if not t_slice:
        t_slice = slice(X.index.min(),X.index.max())
        print(t_slice)

    if t_slice != None:
        X_fit = X.loc[t_slice,:].dropna().ravel()
        y_fit = y.reindex(X_fit.index).ravel()
    else:
        X_fit=X.ravel()
        y_fit=y.ravel()

    pear_test = r_regression(X_fit, y_fit,center=True)
    abs_pear_test = np.abs(pear_test)
```

```
    f_test, p_value = f_regression(X_fit, y_fit,center=False)
    t_test = np.sqrt(f_test)/np.sqrt(1)
    mi = mutual_info_regression(X_fit, y_fit)
    nobs = X_fit.count().unique()[0]
    feat_stats = pd.DataFrame({ 'nobs':nobs,
                                'mutual_info':mi,
                                'p_value':p_value,
                                't_test':t_test,
                                'pearson':pear_test,
                                'abs_pearson':abs_pear_test},
                                index=X.columns)
    print('from ',X_fit.index.min(),' to ',X_fit.index.max())
    return feat_stats.sort_values(by=[sort_by],ascending=False)

def generate_train_predict_calender(df, window_type = None, window_size=None):
    date_ranges = []
    index = df.index
    num_days = len(index)

    if window_type == 'fixed':
        for i in range(0, num_days - window_size):
            train_start_date = index[i]
            train_end_date = index[i + window_size - 1]
            prediction_date = index[i + window_size]
            date_ranges.append([train_start_date, train_end_date, prediction_date])

    if window_type == 'expanding':
        for i in range(0, num_days - window_size):
            train_start_date = index[0]
            train_end_date = index[i + window_size - 1]
            prediction_date = index[i + window_size]
            date_ranges.append([train_start_date, train_end_date, prediction_date])

    return date_ranges

def graph_df(df,w=10,h=15):
    fig, axes = plt.subplots(nrows=len(df.columns), ncols=1, figsize=(w, h))

    # Loop through columns and create a plot for each
    for i, col in enumerate(df.columns):
        axes[i].plot(df[col])
        axes[i].set_title(col)

    # Adjust layout
    plt.tight_layout()
    plt.show()

from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin
class StatsModelsWrapper_with_OLS(BaseEstimator, RegressorMixin):
    def __init__(self, exog, endog):
        self.exog = exog
        self.endog = endog

    def fit(self, X_fit, y_fit):
        X_with_const = add_constant(X_fit, has_constant='add')  # Add constant to the features
        self.model_ = OLS(y_fit, X_with_const)
        self.results_ = self.model_.fit()
        return self

    def predict(self, X_pred):
        X_pred_constant = add_constant(X_pred, has_constant='add')
        return self.results_.predict(X_pred_constant)

    def summary(self,title):
        if title is not None:
            return self.results_.summary(title=title)
        else:
            generic_title = f"OLS Estimation Summary"
            return self.results_.summary(title=generic_title)


class EWMTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, halflife=3):
        self.halflife = halflife

    def fit(self, X, y=None):
        return self

    def set_output(self, as_dataframe=True):
        self.output_as_dataframe = as_dataframe
        return self

    def transform(self, X):
        # Convert to DataFrame if not already one
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)

        # Apply the EWM transformation
        X_transformed = X.ewm(halflife=self.halflife).mean()

        # If output_as_dataframe is False, convert to NumPy array
        if not self.output_as_dataframe:
            return X_transformed.values
        return X_transformed
```

```
198
```

```
all_etfs = ['SPY','AOA', 'AOK', 'AOM', 'AOR', 'IAGG', 'IDEV', 'IEMG', 'IJH', 'IJR','IUSB', 'IVV']
spdr_etfs = ['SPY','XLK', 'XLF', 'XLV', 'XLY', 'XLP', 'XLE', 'XLI', 'XLB', 'XLU']
target_risk_etfs = ['AOA', 'AOK', 'AOM', 'AOR']
component_etfs = ['IAGG', 'IDEV', 'IEMG', 'IJH', 'IJR', 'IUSB', 'IVV']
#X.loc[:,component_etfs].dropna().describe()
```

Double-click (or enter) to edit

```
# Convert index to feature teo's by datetime index and expand time stamp in inclue NYC closing time - then convert to UTC. (TEO = Time of Effective Observation)

all_etf_closing_prices_df = yf.download(spdr_etfs)['Close']
etf_features_df = log_returns(all_etf_closing_prices_df).dropna()
etf_features_df.index = etf_features_df.index.tz_localize('America/New_York').map(lambda x: x.replace(hour=16, minute=00)).tz_convert('UTC')
etf_features_df.index.name = 'teo'
#print(etf_features_df)

# convert teo's to target teos by shifting the time to the beginning of the return window

etf_targets_df = etf_features_df.copy()
etf_targets_df.loc[:,'teo_original'] = etf_targets_df.index
etf_targets_df.loc[:,'teo'] = etf_targets_df['teo_original'].shift(1)
etf_targets_df.set_index('teo',inplace=True)
etf_targets_df = etf_targets_df.drop(['teo_original'],axis=1).dropna()
etf_targets_df = etf_targets_df.iloc[1:,:]
etf_targets_df

# Sync dates between features and targets

common_dates = etf_features_df.index.intersection(etf_targets_df.index)
#print(common_dates)
etf_features_df = etf_features_df.loc[common_dates]
etf_targets_df = etf_targets_df.loc[common_dates]

# Now that features and targets are aligned we can simplyfy to date format

etf_targets_df = simplify_teos(etf_targets_df)
etf_features_df = simplify_teos(etf_features_df)
#print(etf_features_df)
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  10 of 10 completed
```

```
# ROLL FORWARD SIMULATOR - 1) TRAIN UP TO T-1, 2) PREDICT AT T, 3) Store predictions in 'regout' and  list of 'fit_obj" for each fit date in fit_list which contains in-sample

def Simulate(X,y,window_size=400,window_type='expanding',pipe_steps={}, param_grid={}, tag = None):
    regout = pd.DataFrame(index=y.index)
    #attrs_df = pd.DataFrame(0, index=X.index, columns=X.columns)
    fit_list = []

    date_ranges = generate_train_predict_calender(X, window_type=window_type,window_size = window_size)

    fit_obj = Pipeline(steps=pipe_steps).set_output(transform="pandas")  #### NOTE THAT .SET_OUTPUT IS NEEDED IF YOU WANT TO SEE FEATURE NAMES IN .SUMMARY() BUT IT WILL SLOW D
    fit_obj.set_params(**param_grid)
    fit_obj

    for n, dates in enumerate(date_ranges):

        start_training, end_training, prediction_date = dates[0], dates[1],dates[2]

        fit_X = X[start_training:end_training]
        fit_y = y[start_training:end_training]
        pred_X = X[prediction_date:prediction_date]
        trange = slice(fit_X.index[0].strftime('%Y-%m-%d'), fit_X.index[-1].strftime('%Y-%m-%d'), None)
        title = tag + trange.start + ' to ' + trange.stop


        #if n%252 == 0 and n != 0:
        if n == 1000 and n != 0:
            print('tag=',tag,' ',prediction_date.floor('D'), "  ", n, '  ', prediction )

        if n%22 == 0:
            #print(fit_X,fit_y)
            with warnings.catch_warnings():
                warnings.simplefilter(action='ignore')
                fit_obj.fit(fit_X, fit_y)

        if hasattr(fit_obj.predict(pred_X), 'values'):
            prediction = np.round(fit_obj.predict(pred_X).values[0],5)
        else:
            prediction = np.round(fit_obj.predict(pred_X)[0],5)

        if hasattr(fit_obj, 'summary'):
            print('has summary')
            tuple = (tag, prediction_date, fit_obj['final_estimator'].summary(title=title))
        else:
            tuple = (tag, prediction_date, fit_obj)

        fit_list.append(tuple)

        regout.loc[prediction_date,'prediction'] = prediction
```

```

    return regout.dropna(), fit_list
```

```
# Wrapper to Roll the Simulator and save in Results_xr
learner_tag = 'OLS'
#sweep_tags = ['HT '+learner_tag,'BW '+learner_tag, 'Both '+learner_tag] #,'Both '+learner_tag]
n_ewa_lags_list = [1,2,3,4,5,6,7]
sweep_tags = ['ewa_halflife_n ='+str(x) for x in n_ewa_lags_list ]

X = etf_features_df.drop(['SPY'],axis=1)
y = etf_targets_df['SPY']

X_list = [X.ewm(halflife=n,min_periods=n).mean().dropna() for n in n_ewa_lags_list]
y_list = [y[X_list[n].index] for n in range(len(X_list))]

regout_list = []
xr_list = []
Results_xr = xr.Dataset()
sweep_fit_list = []

for n, tag in enumerate(sweep_tags):

    title_prefix =  sweep_tags[n]
    y.index.name = 'teo'

    pipe_steps=[
        ('scaler', StandardScaler()),
        #('final_estimator', LinearRegression()),
        ('final_estimator', StatsModelsWrapper_with_OLS(X,y))
              ]

    #param_grid = {'ewm__halflife' : n_ewa_lags_list[n]}

    regout_df, fit_list = Simulate(X_list[n],
                               y_list[n],
                               window_size=400,
                               window_type='expanding',
                               pipe_steps=pipe_steps,
                               param_grid={},
                               tag=sweep_tags[n])

    regout_list.append(regout_df)

    sweep_fit_list.append(fit_list)
```

```
tag= ewa_halflife_n =1   2004-07-22 00:00:00    1000    0.00021
tag= ewa_halflife_n =2   2004-07-23 00:00:00    1000    0.00015
tag= ewa_halflife_n =3   2004-07-26 00:00:00    1000    0.00022
tag= ewa_halflife_n =4   2004-07-27 00:00:00    1000    0.00024
tag= ewa_halflife_n =5   2004-07-28 00:00:00    1000    0.00022
tag= ewa_halflife_n =6   2004-07-29 00:00:00    1000    4e-05
tag= ewa_halflife_n =7   2004-07-30 00:00:00    1000    -0.00011
```

```
print(sweep_fit_list[3][100][2]['final_estimator'].summary(title='test title here'))
```

```
                               test title here
==============================================================================
Dep. Variable:                    SPY   R-squared:                       0.010
Model:                            OLS   Adj. R-squared:                  0.008
Method:                 Least Squares   F-statistic:                     7.207
Date:                Tue, 08 Apr 2025   Prob (F-statistic):           1.74e-10
Time:                        14:59:08   Log-Likelihood:                 19776.
No. Observations:                6604   AIC:                        -3.953e+04
Df Residuals:                    6594   BIC:                        -3.946e+04
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0003      0.000      2.001      0.045    6.04e-06       0.001
XLB            0.0003      0.000      0.829      0.407      -0.000       0.001
XLE           -0.0005      0.000     -2.177      0.030      -0.001   -4.71e-05
XLF           -0.0006      0.000     -2.046      0.041      -0.001   -2.37e-05
XLI            0.0007      0.000      1.740      0.082   -8.81e-05       0.001
XLK           -0.0006      0.000     -2.377      0.017      -0.001   -9.93e-05
XLP           -0.0002      0.000     -0.844      0.399      -0.001       0.000
XLU           -0.0005      0.000     -2.712      0.007      -0.001      -0.000
XLV           -0.0007      0.000     -2.784      0.005      -0.001      -0.000
XLY            0.0007      0.000      2.249      0.025    8.76e-05       0.001
==============================================================================
Omnibus:                     1441.113   Durbin-Watson:                   2.079
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            29384.087
Skew:                          -0.520   Prob(JB):                         0.00
Kurtosis:                      13.281   Cond. No.                         7.33
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Lisa

```
regout_df
```

teo	prediction
2000-08-03	0.00382
2000-08-04	0.00508
2000-08-07	0.00215
2000-08-08	0.00307
2000-08-09	0.00472
...	...
2025-04-01	0.00008
2025-04-02	0.00016
2025-04-03	0.00039
2025-04-04	0.00342
2025-04-07	0.00340

6206 rows × 1 columns

Next steps: Generate code with regout_df | View recommended plots | New interactive sheet

Double-click (or enter) to edit

```
xr_list = []
for n, tag in enumerate(sweep_tags):
    regout_df = regout_list[n]
    fit_tuple = fit_list[n]

    regout_df['preds_index'] = norm.cdf(zscaler.fit_transform(regout_df[['prediction']]))
    regout_df['actual'] = etf_targets_df['SPY'].loc[regout_df.prediction.index].dropna()
    regout_df['leverage'] = L_func_3(regout_df,pred_col='preds_index',params=[0,.5,1.5,2])
    regout_df['perf_ret'] = regout_df['leverage']*regout_df['actual']
    regout_df['avg_beta'] = regout_df['leverage'].expanding().mean()
    regout_df.index.name = 'teo'


    xr_list.append(regout_df.to_xarray().expand_dims(tag=[sweep_tags[n]]))
    regout_list[n] = regout_df
    sweep_fit_list.append(fit_list)
```

```
pd.DataFrame(y)
```

teo	SPY
1998-12-23	-0.004321
1998-12-24	-0.002550
1998-12-28	0.015709
1998-12-29	-0.008076
1998-12-30	0.000000
...	...
2025-04-01	0.006308
2025-04-02	-0.050537
2025-04-03	-0.060327
2025-04-04	-0.001783
2025-04-07	0.031191

6612 rows × 1 columns

```
Results_xr = xr.Dataset()
Results_xr = xr.merge(xr_list)
```

```
Results_xr = xr.merge([Results_xr,pd.DataFrame(y).to_xarray().expand_dims(tag=['SP500']).rename({'SPY':'perf_ret'})])
Results_xr
```

xarray.Dataset

▶ Dimensions: (**teo**: 6612, **tag**: 8)

▼ Coordinates:

teo	(teo)	datetime64[ns]	1998-12-23 ... 2025-04-07
tag	(tag)	object	'SP500' ... 'ewa_halflife_n =7'

▼ Data variables:

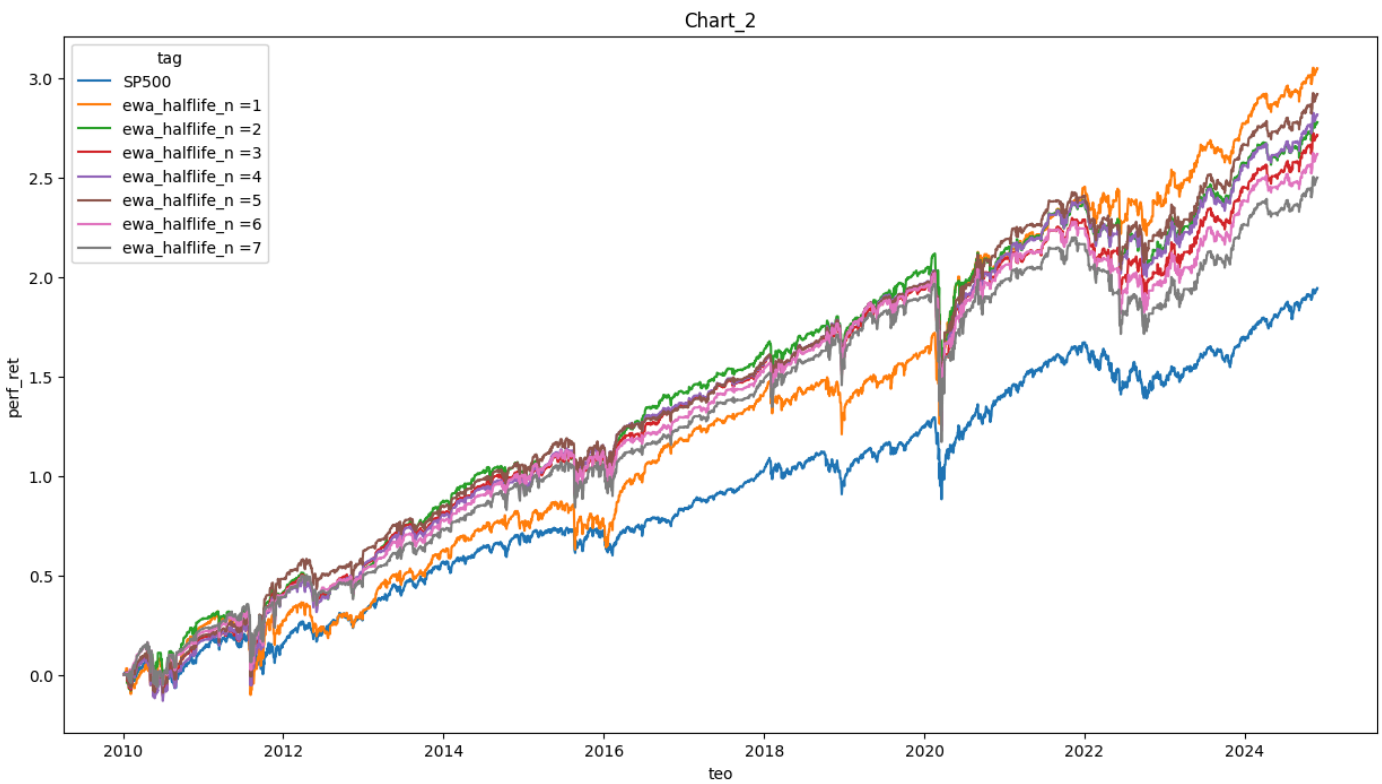
prediction	(tag, teo)	float64	nan nan nan ... 0.00342 0.0034
preds_index	(tag, teo)	float64	nan nan nan ... 0.549 0.9881 0.9877
actual	(tag, teo)	float64	nan nan nan ... -0.001783 0.03119
leverage	(tag, teo)	float64	nan nan nan nan ... 0.5 1.5 2.0 2.0
perf_ret	(tag, teo)	float64	-0.004321 -0.00255 ... 0.06238
avg_beta	(tag, teo)	float64	nan nan nan ... 0.9157 0.9159

▶ Indexes: (2)

▶ Attributes: (0)

```
Results_xr['perf_ret'].sel(teo=slice('2010-01-01','2024-11-28')).cumsum(dim='teo').plot.line(x='teo')
plt.title("Chart_2")
```

Text(0.5, 1.0, 'Chart_2')



Double-click (or enter) to edit

```

trange = slice(regout_list[-1].index[0],regout_list[-1].index[-1])

sim_stats(regout_list,sweep_tags,author='CG',trange = trange)
```

```
SIMULATION RANGE:  from  2000-08-03 00:00:00 to  2025-04-07 00:00:00
```

teo	ewa_halflife_n=1	ewa_halflife_n=2	ewa_halflife_n=3	ewa_halflife_n=4	ewa_halflife_n=5	ewa_halflife_n=6	ewa_halflife_n=7
return	0.129281	0.127278	0.115437	0.110636	0.111914	0.105734	0.10093
stdev	0.267514	0.275753	0.277365	0.282408	0.284253	0.286627	0.286417
sharpe	0.483267	0.461566	0.416192	0.391759	0.393713	0.368891	0.352389
avg_beta	0.959878	0.943281	0.937077	0.934499	0.930551	0.922172	0.915888
beta_1_return	0.134685	0.134932	0.123188	0.118391	0.120267	0.114658	0.110199
pos_bet_ratio	0.533999	0.51853	0.514341	0.509507	0.51418	0.519497	0.526587
rmse	0.012132	0.012139	0.012153	0.012171	0.01217	0.012173	0.012176
mae	0.008056	0.00805	0.008047	0.00805	0.008046	0.008045	0.008044
r2	0.002871	0.001599	-0.000588	-0.003588	-0.003513	-0.003995	-0.004361
benchmark return	0.069934	0.069934	0.069934	0.069934	0.069934	0.069934	0.069934
benchmark std	0.192877	0.192877	0.192877	0.192877	0.192877	0.192877	0.192877
benchmark sharpe	0.362585	0.362585	0.362585	0.362585	0.362585	0.362585	0.362585
beg_pred	2000-08-03	2000-08-03	2000-08-03	2000-08-03	2000-08-03	2000-08-03	2000-08-03
end_pred	2025-04-07	2025-04-07	2025-04-07	2025-04-07	2025-04-07	2025-04-07	2025-04-07

```
1 p_by_year(X_list[0],y_list[0],sort_by = 'p_value',t_list=None).T
```

```
from  2025-01-02 00:00:00  to  2025-04-07 00:00:00
```

Ticker	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
1998	-0.587210	-0.374294	-0.506994	-0.280492	-0.177197	-0.502744	-0.209148	-0.700171	-0.533026
1999	-0.071519	-0.061252	0.059149	-0.018341	-0.044764	0.011468	-0.058285	-0.005569	-0.027985
2000	0.044946	0.036438	0.015815	-0.081600	-0.095882	-0.006675	0.004315	-0.038148	-0.002018
2001	-0.010815	0.052548	-0.083981	-0.033722	-0.045207	0.037237	-0.020312	-0.013132	-0.005674
2002	-0.054589	-0.048429	-0.056093	-0.064544	-0.045041	-0.013822	-0.054438	-0.003021	-0.063441
2003	-0.088468	-0.078263	-0.047911	-0.057864	-0.050468	0.007877	-0.014729	-0.175393	-0.060857
2004	0.069259	-0.052878	-0.029115	0.028633	-0.026441	-0.032288	0.034724	-0.049986	0.018488
2005	-0.032689	-0.025743	-0.101314	-0.091652	-0.041514	-0.177229	-0.157255	-0.128898	-0.093580
2006	-0.092085	-0.107380	-0.057890	-0.032647	0.037901	-0.070698	-0.107961	-0.064684	-0.013032
2007	-0.186103	-0.280784	-0.065470	-0.110744	-0.119056	-0.051921	-0.105277	-0.044397	-0.061188
2008	-0.162453	-0.248657	-0.098522	-0.163903	-0.109947	-0.130863	-0.206661	-0.271083	-0.050044
2009	-0.064311	-0.062385	-0.094716	-0.042322	-0.073561	-0.066924	0.020160	0.029804	-0.065518
2010	0.023578	-0.020579	-0.035487	-0.022676	-0.060134	-0.048782	-0.045703	-0.096340	-0.021732