Final Asset Allocation Project

In [ ]:
```
market = "SPY"
tickers = ["DG", "MRK", "CB", "NOC", "BP"] # chosen to be diverse across sec

# start date test
test_start = "2012-12-01"
test_end = "2017-10-01"

# eval date test
eval_start = "2017-10-02"
eval_end = "2019-10-01"
```

In [ ]:
```
from urllib.request import urlretrieve
import os

query = "https://query1.finance.yahoo.com/v7/finance/download/{}?period1=132
for ticker in tickers+[market]:
    os.makedirs("data/{}".format(ticker), exist_ok=True)
    urlretrieve(query.format(ticker), "data/{}/{}.csv".format(ticker, ticker
```

We will now compute the returns of our data

In [ ]:
```
# compute returns using the close  for each and add drop all other columns
import pandas as pd
import numpy as np
import os
for ticker in tickers+[market]:
    df = pd.read_csv("data/{}/{}.csv".format(ticker, ticker))
    df["return"] = df["Close"].pct_change()
    df = df.dropna()
    df = df[["Date", "Close", "return"]]
    df.to_csv("data/{}/{}.csv".format(ticker, ticker), index=False)
```

Next we will collect our rate data and split it into monthly data

In [ ]:
```
# load the risk free rate and save to rates.csv
rates_static = "FRB_H15.csv"
rates_df = pd.read_csv(rates_static)

# format the date form YYYY-MM to YYYY-MM-DD
rates_df["Date"] = rates_df["Date"].apply(lambda x: x+"-01")
rates_df["return"] = rates_df["return"]/1200

# save it to rates.csv
os.makedirs("data/rates", exist_ok=True)
rates_df.to_csv("data/rates/rates.csv", index=False)
```

Now we will split our data into train and test periods

In [ ]:
```
# split the data into train and test
for ticker in tickers+[market, "rates"]:
    # make dir in data
    os.makedirs("data/{}".format(ticker), exist_ok=True)
    df = pd.read_csv("data/{}/{}.csv".format(ticker, ticker))
    df["Date"] = pd.to_datetime(df["Date"])
    df = df.sort_values("Date")
    df_train = df[(df["Date"] >= test_start) & (df["Date"] <= test_end)]
    df_test = df[(df["Date"] >= eval_start) & (df["Date"] <= eval_end)]
    df_train.to_csv("data/{}/{}_train.csv".format(ticker, ticker), index=Fal
    df_test.to_csv("data/{}/{}_test.csv".format(ticker, ticker), index=False
```

Next we will aggregate our data into a training data frame

In [ ]:
```
# compute the combined training set
train_data = pd.DataFrame()
for ticker in tickers+[market]+["rates"]:
    df = pd.read_csv("data/{}/{}_train.csv".format(ticker, ticker))
    df = df[["Date", "return"]]
    df = df.rename(columns={"return": ticker})
    if train_data.empty:
        train_data = df
    else:
        train_data = pd.merge(train_data, df, on="Date", how="inner")
```

Now we can compute the basic statistics we need to devise multiple portfolios

In [ ]:
```
train_data = train_data.dropna()
train_data = train_data.set_index("Date")
train_data.to_csv("data/train_data.csv")

# cov
cov_matrix = train_data.cov()
cov_matrix.to_csv("data/cov_matrix.csv")

# corr
corr_matrix = train_data.corr()
corr_matrix.to_csv("data/corr_matrix.csv")

# deviation
dev_matrix = train_data.std()
dev_matrix.to_csv("data/dev_matrix.csv")

# expected return
expected_return = train_data.mean()
expected_return.to_csv("data/expected_return.csv")

print("Covariance Matrix")
print(cov_matrix)
print("\n")
```

```
print("Correlation Matrix")
print(corr_matrix)
print("\n")

print("Deviation Matrix")
print(dev_matrix)
print("\n")

print("Expected Return")
print(expected_return)
print("\n")
```

```
Covariance Matrix
                 DG       MRK            CB           NOC        BP  \
DG     4.508122e-03  0.000342  8.105542e-04  7.840403e-04  0.000359
MRK    3.424459e-04  0.002364  4.898439e-04  5.629656e-04  0.000636
CB     8.105542e-04  0.000490  1.456910e-03  6.276540e-04  0.000565
NOC    7.840403e-04  0.000563  6.276540e-04  1.848072e-03  0.000525
BP     3.588692e-04  0.000636  5.645264e-04  5.252179e-04  0.003952
SPY    6.894545e-04  0.000634  7.518302e-04  5.584653e-04  0.000739
rates  4.232179e-07 -0.000002  2.624179e-07 -4.793574e-07  0.000002

                SPY         rates
DG     6.894545e-04  4.232179e-07
MRK    6.339753e-04 -1.820762e-06
CB     7.518302e-04  2.624179e-07
NOC    5.584653e-04 -4.793574e-07
BP     7.392004e-04  2.135279e-06
SPY    7.845685e-04  3.310369e-07
rates  3.310369e-07  5.669386e-08


Correction Matrix
```

```
Correlation Matrix
             DG       MRK        CB       NOC        BP       SPY     rates
DG     1.000000  0.104901  0.316277  0.271632  0.085027  0.366600  0.026473
MRK    0.104901  1.000000  0.263953  0.269344  0.208180  0.465524 -0.157279
CB     0.316277  0.263953  1.000000  0.382511  0.235281  0.703215  0.028874
NOC    0.271632  0.269344  0.382511  1.000000  0.194356  0.463790 -0.046831
BP     0.085027  0.208180  0.235281  0.194356  1.000000  0.419822  0.142661
SPY    0.366600  0.465524  0.703215  0.463790  0.419822  1.000000  0.049636
rates  0.026473 -0.157279  0.028874 -0.046831  0.142661  0.049636  1.000000


Deviation Matrix
DG       0.067143
MRK      0.048620
CB       0.038169
NOC      0.042989
BP       0.062861
SPY      0.028010
rates    0.000238
dtype: float64


Expected Return
DG       0.010427
MRK      0.004874
CB       0.011683
NOC      0.026432
BP       0.001467
SPY      0.010480
rates    0.000171
dtype: float64
```

In [ ]:

```python
# compute beta for each stock
market_train = pd.read_csv("data/{}/{}_train.csv".format(market, market))
```

```
market_train = market_train.set_index("Date")
market_train = market_train.rename(columns={"return": market})
market_train = market_train.dropna()

# read in the rates test data
rates_train = pd.read_csv("data/rates/rates_train.csv")
rates_train = rates_train.set_index("Date")
rates_train = rates_train.rename(columns={"return": "rates"})
rates_train = rates_train.dropna()

# compute the excess return for the market
market_train[market] = market_train[market] - rates_train["rates"]

betas = pd.DataFrame()
intercepts = pd.DataFrame()
residual_dev = pd.DataFrame()
for ticker in tickers:
    df = pd.read_csv("data/{}/{}_train.csv".format(ticker, ticker))
    df = df.set_index("Date")
    df = df.rename(columns={"return": ticker})
    df = df.dropna()
    df[ticker] = df[ticker] - rates_train["rates"]
    df = pd.merge(df, market_train, on="Date", how="inner")
    # beta = (df[ticker].cov(df[market]))/(df[market].var())
    # intercept = df[ticker].mean() - beta*df[market].mean()
    # betas.loc[ticker, "beta"] = beta
    # intercepts.loc[ticker, "intercept"] = intercept

    # perform linear regression
    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    model.fit(df[[market]], df[[ticker]])
    intercept = model.intercept_[0]
    beta = model.coef_[0][0]
    intercepts.loc[ticker, "intercept"] = intercept
    betas.loc[ticker, "beta"] = beta

    # plot the regression line
    import matplotlib.pyplot as plt
    # plt.scatter(df[[market]], df[[ticker]])
    # plt.plot(df[[market]], model.predict(df[[market]]), color="red")

    # plot the residuals
    # plt.scatter(df[[market]], df[[ticker]] - model.predict(df[[market]]))

    residuals = df[[ticker]] - model.predict(df[[market]])
    residuals = residuals**2
    residual_dev.loc[ticker, "residual_dev"] = residuals.sum().values[0]**0.

    print("Beta {ticker}: {beta}".format(ticker=ticker, beta=beta))
    print("Intercept {ticker}: {intercept}".format(ticker=ticker, intercept=
    print("Residual Dev {ticker}: {residual_dev}".format(ticker=ticker, resi
    print("\n")


betas.to_csv("data/betas.csv")
```

```
intercepts.to_csv("data/intercepts.csv")
residual_dev.to_csv("data/residual_dev.csv")
```

```
Beta DG: 0.8785577507728043
Intercept DG: 0.0011989674873874815
Residual Dev DG: 0.4757402186651434


Beta MRK: 0.8106525543922
Intercept MRK: -0.0036544905788099104
Residual Dev MRK: 0.32778068959050455


Beta CB: 0.9583274954573505
Intercept CB: 0.0016325255467508955
Residual Dev CB: 0.20667489290719473


Beta NOC: 0.7126231420051568
Intercept NOC: 0.01891431827260464
Residual Dev NOC: 0.2900966104172669


Beta BP: 0.9398282702149634
Intercept BP: -0.008393210878596829
Residual Dev BP: 0.43448845086189986
```

In [ ]:

```
# alphas
# compute expected return using analyst data
analyst = pd.read_csv("analysts.csv")
alphas = pd.DataFrame()
# pull the price
for ticker in tickers:
    expected_price = analyst[analyst["Ticker"] == ticker]["Price"].values[0]
    dividend = analyst[analyst["Ticker"] == ticker]["Dividend"].values[0]
    train_data = pd.read_csv("data/{}/{}_train.csv".format(ticker, ticker))
    last_price = analyst[analyst["Ticker"] == ticker]["Current"].values[0]
    analyst_return = (expected_price + dividend - last_price) / last_price

    # compute the alpha
    last_rf = rates_train["rates"].iloc[-1]
    last_rm = market_train[market].mean()
    erm = analyst_return/12.0
    alpha = (erm - last_rf - betas.loc[ticker, "beta"]*(last_rm - last_rf))*
    alphas.loc[ticker, "alpha"] = alpha
    # print all intermediate values
    print("Expected Return {ticker}: {erm}".format(ticker=ticker, erm=erm))
    print("Expected Market Return: {last_rm}".format(last_rm=last_rm))
    print("Expected Risk Free Rate: {last_rf}".format(last_rf=last_rf))
    print("Alpha {ticker}: {alpha}".format(ticker=ticker, alpha=alpha))
    print("\n")

alphas.to_csv("data/alphas.csv")
```

```
Expected Return DG: 0.011904761904761904
Expected Market Return: 0.010309632619618819
Expected Risk Free Rate: 0.0008166666666666
Alpha DG: 0.27479764223067926


Expected Return MRK: 0.01445978454136596
Expected Market Return: 0.010309632619618819
Expected Risk Free Rate: 0.0008166666666666
Alpha MRK: 0.594762077618046


Expected Return CB: 0.011966748880971942
Expected Market Return: 0.010309632619618819
Expected Risk Free Rate: 0.0008166666666666
Alpha CB: 0.20527119281507408


Expected Return NOC: 0.007151112395261482
Expected Market Return: 0.010309632619618819
Expected Risk Free Rate: 0.0008166666666666
Alpha NOC: -0.04304614957459062


Expected Return BP: 0.009515570934256057
Expected Market Return: 0.010309632619618819
Expected Risk Free Rate: 0.0008166666666666
Alpha BP: -0.022285350318316903
```

In [ ]:
```
# # compute residual variance for all stocks
# residuals = pd.DataFrame()
# for ticker in tickers:
#     df = pd.read_csv("data/{}/{}_train.csv".format(ticker, ticker))
#     df = df.set_index("Date")
#     df = df.rename(columns={"return": ticker})
#     df = df.dropna()
#     df[ticker] = df[ticker] - rates_train["rates"]
#     df = pd.merge(df, market_train, on="Date", how="inner")
#     df["residual"] = df[ticker] - betas.loc[ticker, "beta"]*df[market] - i
#     residuals.loc[ticker, "residual_variance"] = df["residual"].std()
# residuals.to_csv("data/residuals.csv")
```

List of Strategies

The assumption we are utilizing is that the CAL has a consistent slope hence we are only computing the risky portfolio.

1. Equities equal allocation portfolio
2. SP500 only
3. optimal risky portfolio that matches SP500 expected return

4. optimal risky portfolio that matches SP500 deviation
5. Beta Scaled Portfolio (Growth Portfolio)
6. Treynor-Black Allocation
7. Markowitz allocation (market as an asset to simulate comparison to treynor black)

In [ ]:
```
final_weights = pd.DataFrame()
# add the tickers + market to index
final_weights.index = tickers + [market]

# Algorithm 1 all equal weights
weights = np.ones(len(tickers)+1)/(len(tickers)+1)
final_weights["equal"] = weights

# Algorithm 2 only market
weights = np.zeros(len(tickers)+1)
weights[-1] = 1
final_weights["market"] = weights
```

In [ ]:
```
# optimization data
from scipy.optimize import minimize

train_data = pd.read_csv("data/train_data.csv")

# choose only the tickers and the market
train_data = train_data[tickers+[market]]

# compute the expected return
expected_return = train_data.mean()

# compute the covariance matrix
cov_matrix = train_data.cov()
```

In [ ]:
```
# Algorithm 3 minimum variance equal return to market
targ_ret = expected_return.loc[market]

def portfolio_dev(weights, cov_matrix):
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

def constraint(weights, targ_ret, expected_return):
    return np.dot(weights, expected_return) - targ_ret

def constraint2(weights):
    return np.sum(weights) - 1

constraints = [{"type": "eq", "fun": constraint, "args": (targ_ret, expectec

weights_guess = np.ones(len(tickers)+1)/(len(tickers)+1)

optimal_portfolio = minimize(portfolio_dev, weights_guess, args=(cov_matrix)
final_weights.loc[[ticker for ticker in tickers]+[market], "min_var_mark_ret
```

In [ ]:
```
# Algorithm 4 equal variance maximal return

targ_var = dev_matrix.loc[market]

def ret_func(weights, expected_return):
    return -np.dot(weights, expected_return)

def constraint(weights, targ_var, cov_matrix):
    return portfolio_dev(weights, cov_matrix) - targ_var

def constraint2(weights):
    return np.sum(weights) - 1

constraints = [{"type": "eq", "fun": constraint, "args": (targ_var, cov_matr

optimal_portfolio = minimize(ret_func, weights_guess, args=(expected_return)
final_weights.loc[[ticker for ticker in tickers]+[market], "equal_var_max_re
```

In [ ]:
```
# Algorithm 5 Beta Scaled
betas = pd.read_csv("data/betas.csv")
betas.set_index("Unnamed: 0", inplace=True)
betas.index.name = "Ticker"
betas.loc[market, "beta"] = 1

scaled_betas = betas["beta"] / betas["beta"].sum()

final_weights.loc[[ticker for ticker in tickers]+[market], "beta_scaled"] =
```

In [ ]:
```
# Algorithm six - Markowitz

# optimization function
def portfolio_dev(weights, cov_matrix):
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

# constraints - we are okay with short selling
def constraint(weights):
    return weights.sum() - 1
constraints = {"type": "eq", "fun": constraint}

# initial guess
weights_guess = np.random.rand(len(tickers)+1)

# minimize
from typing import final
from scipy.optimize import minimize
optimal_portfolio = minimize(portfolio_dev, weights_guess, args=(cov_matrix)

# save to final weights
final_weights.loc[[ticker for ticker in tickers]+[market], "markowitz"] = op
```

In [ ]:
```
# Algorithm seven - Treynor Black

# compute the starting weights alpha / resiudal variance
weights = pd.DataFrame()
```

```
for ticker in tickers:
    alpha = alphas.loc[ticker, "alpha"]
    residual_variance = residual_dev.loc[ticker, "residual_dev"]**2
    weight = alpha/residual_variance
    weights.loc[ticker, "weight"] = weight

# scale the weights
weights["weight"] = weights["weight"]/weights["weight"].sum()


# residual variance of portfolio
residual_variance_port = 0
alpha_port = 0
for ticker in tickers:
    residual_variance_port += weights.loc[ticker, "weight"]**2 * residual_de
    alpha_port += weights.loc[ticker, "weight"] * alphas.loc[ticker, "alpha"

initial_position = (alpha_port/residual_variance_port) / (expected_return[ma
print("Initial Position: ", initial_position)

beta_port = 0
for ticker in tickers:
    beta_port += weights.loc[ticker, "weight"]*betas.loc[ticker, "beta"]
print("Beta Portfolio: ", beta_port)

adjusted_initial_position = initial_position / (1+ (1-beta_port)*initial_pos
print("Adjusted Initial Position: ", adjusted_initial_position)

m_initial_position = 1-adjusted_initial_position

expected_return_port = (m_initial_position+beta_port*adjusted_initial_positi
variance_port = (m_initial_position+beta_port*adjusted_initial_position)**2
print("Expected Return Portfolio: ", expected_return_port)
print("Variance Portfolio: ", variance_port)

for ticker in weights.index:
    final_weights.loc[ticker, "treynor"] = adjusted_initial_position*weights

# add market to the weights
final_weights.loc[market, "treynor"] = m_initial_position
print("Weights: ", final_weights)
```

```
Initial Position:  0.8179381580415437
Beta Portfolio:  0.8863449402080142
Adjusted Initial Position:  0.7483677855865897
Expected Return Portfolio:  0.32721092755967873
Variance Portfolio:  0.02241204501304662
Weights:           equal  market  min_var_mark_return  equal_var_max_return
beta_scaled  \
DG   0.166667     0.0             0.025749             -0.001143     0.16576
6
MRK  0.166667     0.0             0.100610              0.015739     0.15295
4
CB   0.166667     0.0             0.046923              0.027365     0.18081
7
NOC  0.166667     0.0             0.053161              0.295512     0.13445
7
BP   0.166667     0.0             0.037610             -0.023064     0.17732
6
SPY  0.166667     1.0             0.735948              0.685590     0.18868
0

     markowitz   treynor
DG    0.017442  0.083162
MRK   0.072848  0.379167
CB    0.036422  0.329159
NOC   0.133218 -0.035035
BP    0.015318 -0.008086
SPY   0.724751  0.251632
```

In [ ]:

```python
final_weights.to_csv("data/final_weights.csv")

train_data = None
for ticker in tickers+[market]:

    # read in rates
    rates = pd.read_csv("data/rates/rates_train.csv")
    rates = rates.set_index("Date")
    rates = rates[["return"]]

    df = pd.read_csv("data/{}/{}_train.csv".format(ticker, ticker))
    df = df.set_index("Date")
    df = df[["return"]]
    df = df - rates

    df = df.rename(columns={"return": ticker})
    if train_data is None:
        train_data = df
    else:
        train_data = pd.merge(train_data, df, on="Date", how="inner")

train_data = train_data.dropna()
train_data.to_csv("data/test_data.csv")

dates_to_compute_return = ["2017-10-02"]
# weighted returns
weighted_returns = pd.DataFrame()
weighted_returns.index = train_data.index
```

```
for strat in final_weights.columns:
    weights = final_weights.loc[:, strat]
    temp_returns = np.dot(train_data, weights)
    weighted_returns.loc[:, strat] = temp_returns

# weighted returns
weighted_returns.to_csv("data/weighted_returns_train.csv")

for date in dates_to_compute_return:
    returns_at_dates = pd.DataFrame(index=final_weights.columns, columns=["a
    weighted_returns_to_now = weighted_returns[weighted_returns.index <= dat

    excess_returns = weighted_returns_to_now

    # compute the average return, total return, and sharpe ratio
    average_return = excess_returns.mean()
    total_return = excess_returns.sum()
    std_dev = excess_returns.std()
    sharpe_ratio = average_return/std_dev

    returns_at_dates["average_return"] = average_return
    returns_at_dates["total_return"] = total_return
    returns_at_dates["std_dev"] = std_dev
    returns_at_dates["sharpe_ratio"] = sharpe_ratio
    os.makedirs("data/FINAL_DATA/", exist_ok=True)
    print("DATE: ", date)
    print(returns_at_dates)
    print("\n")
    returns_at_dates.to_csv("data/FINAL_DATA/returns_at_dates_" + date + ".c
```

```
DATE:  2017-10-02
                      average_return  total_return   std_dev  sharpe_ratio
equal                       0.010723      0.632667  0.030302      0.353877
market                      0.010310      0.608268  0.027999      0.368210
min_var_mark_return         0.010310      0.608268  0.027413      0.376085
equal_var_max_return        0.015176      0.895392  0.028011      0.541797
beta_scaled                 0.010207      0.602228  0.030321      0.336636
markowitz                   0.011931      0.703932  0.027192      0.438767
treynor                     0.008089      0.477272  0.031477      0.256995
```

After computation of all strategies we evaluate them below

1. Evaluation is done using EXCESS RETURNS to account for varying risk free data
2. We evaluate at the five periods described in the announcement
3. The first data period does not have all metrics as the deviation cannot be calculated over one period

In [ ]:
```
final_weights.to_csv("data/final_weights.csv")
```

```
test_data = None
for ticker in tickers+[market]:

    # read in rates
    rates = pd.read_csv("data/rates/rates_test.csv")
    rates = rates.set_index("Date")
    rates = rates[["return"]]

    df = pd.read_csv("data/{}/{}_test.csv".format(ticker, ticker))
    df = df.set_index("Date")
    df = df[["return"]]
    # df = df - rates

    df = df.rename(columns={"return": ticker})
    if test_data is None:
        test_data = df
    else:
        test_data = pd.merge(test_data, df, on="Date", how="inner")

test_data = test_data.dropna()
test_data.to_csv("data/test_data.csv")

dates_to_compute_return = ["2017-11-01", "2018-01-01", "2018-04-01", "2018-1
# weighted returns
weighted_returns = pd.DataFrame()
weighted_returns.index = test_data.index
for strat in final_weights.columns:
    weights = final_weights.loc[:, strat]
    temp_returns = np.dot(test_data, weights)
    weighted_returns.loc[:, strat] = temp_returns

# weighted returns
weighted_returns.to_csv("data/weighted_returns.csv")

for date in dates_to_compute_return:
    returns_at_dates = pd.DataFrame(index=final_weights.columns, columns=["a
    weighted_returns_to_now = weighted_returns[weighted_returns.index <= dat

    excess_returns = weighted_returns_to_now

    # compute the average return, total return, and sharpe ratio
    average_return = excess_returns.mean()
    total_return = excess_returns.sum()
    std_dev = excess_returns.std()
    sharpe_ratio = average_return/std_dev

    returns_at_dates["average_return"] = average_return
    returns_at_dates["total_return"] = total_return
    returns_at_dates["std_dev"] = std_dev
    returns_at_dates["sharpe_ratio"] = sharpe_ratio
    os.makedirs("data/FINAL_DATA/", exist_ok=True)
    print("DATE: ", date)
    print(returns_at_dates)
    print("\n")
    returns_at_dates.to_csv("data/FINAL_DATA/returns_at_dates_" + date + ".c
```

```
DATE:  2017-11-01
                     average_return  total_return  std_dev  sharpe_ratio
equal                      0.026226      0.026226      NaN           NaN
market                     0.030566      0.030566      NaN           NaN
min_var_mark_return        0.027111      0.027111      NaN           NaN
equal_var_max_return       0.033348      0.033348      NaN           NaN
beta_scaled                0.025444      0.025444      NaN           NaN
markowitz                  0.029389      0.029389      NaN           NaN
treynor                    0.017906      0.017906      NaN           NaN


DATE:  2018-01-01
                     average_return  total_return   std_dev  sharpe_ratio
equal                      0.036701      0.110104  0.028575      1.284395
market                     0.031302      0.093906  0.024697      1.267417
min_var_mark_return        0.031582      0.094746  0.025800      1.224098
equal_var_max_return       0.036286      0.108859  0.035515      1.021719
beta_scaled                0.035748      0.107244  0.027665      1.292170
markowitz                  0.033272      0.099816  0.028951      1.149265
treynor                    0.026595      0.079785  0.031850      0.835011


DATE:  2018-04-01
                     average_return  total_return   std_dev  sharpe_ratio
equal                      0.011120      0.066723  0.042492      0.261704
market                     0.005237      0.031423  0.035560      0.147278
min_var_mark_return        0.006669      0.040013  0.036443      0.182995
equal_var_max_return       0.007627      0.045765  0.038647      0.197363
beta_scaled                0.010475      0.062850  0.043029      0.243440
markowitz                  0.007063      0.042377  0.035993      0.196230
treynor                    0.002656      0.015934  0.048535      0.054716


DATE:  2018-10-01
                     average_return  total_return   std_dev  sharpe_ratio
equal                      0.007442      0.089306  0.035601      0.209045
market                     0.004855      0.058262  0.035631      0.136260
min_var_mark_return        0.006069      0.072829  0.034107      0.177940
equal_var_max_return       0.000949      0.011385  0.042165      0.022501
beta_scaled                0.007285      0.087419  0.035570      0.204805
markowitz                  0.004470      0.053644  0.035925      0.124437
treynor                    0.008767      0.105201  0.039973      0.219316


DATE:  2019-10-01
                     average_return  total_return   std_dev  sharpe_ratio
equal                      0.011345      0.272276  0.032529      0.348762
market                     0.007764      0.186328  0.042018      0.184772
min_var_mark_return        0.009044      0.217046  0.037163      0.243347
equal_var_max_return       0.008544      0.205049  0.043355      0.197064
beta_scaled                0.010906      0.261746  0.032396      0.336650
markowitz                  0.008941      0.214591  0.038576      0.231786
treynor                    0.012252      0.294052  0.034365      0.356532
```