Porfolio Optimization

October 12, 2020

[107]:

```
import pandas as pd
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
import pandas as pd
import pandas_datareader as web
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from os.path import isfile
from datetime import datetime


def get_tickers_adj_close_values(tickers, tickers_list_title,␣
 ↪date_start="2012-01-01", date_end="2020-10-10"):
    unavailables = ["AGN", "CELG", "PCLN"]
    missing_data_titles = ["UTX", "ABBV"]

    tickers = [t for t in tickers if t not in unavailables]
    tickers = [t for t in tickers if t not in missing_data_titles]

    backup_filename = "{}_{}_{}.data".format(tickers_list_title, date_start,␣
 ↪date_end)
    # S&P_100_2010.....data
    if isfile(backup_filename):
        print("Loading...")
        prices = pd.read_pickle(backup_filename)
        return prices

    print("Downloading...")

    prices = web.get_data_yahoo(tickers, start = date_start , end = date_end)
    prices = prices['Adj Close']
    prices = prices.dropna(axis='columns')
    prices.to_pickle(backup_filename)
    return prices
```

```
def print_full_dataframe(df):
    with pd.option_context('display.max_rows', None, 'display.max_columns',␣
 ↪None):
        print(df)

def get_best_portfolio(tickers, tickers_list_name,date_start="2012-01-01",␣
 ↪date_end="2020-10-10", tickers_dataframe=None):

    df = None
    if tickers_dataframe is not None:
        df = tickers_dataframe
    else:
        df = get_tickers_adj_close_values(tickers, tickers_list_name,␣
 ↪date_start, date_end)

    # Calculate expected returns and sample covariance
    mu = expected_returns.mean_historical_return(df)
#     print("mu", mu)
    S = risk_models.sample_cov(df)

    # Optimise for maximal Sharpe ratio
    ef = EfficientFrontier(mu, S, weight_bounds=(0, 0.1))
    ef.add_constraint(lambda x : sum(x) == 1.0)
    weights = ef.max_sharpe()

    #  From ordered dict to pd.Series
    title_names = list(weights.keys())
    title_weight = [weights[name] for name in title_names]
    weights = pd.Series(title_weight, index=title_names)

#     print_full_dataframe(weights)
#     print(weights.sum())

    assert weights.sum() <= 1.00000001     # because of float errors

    portfolio_performance = mu.multiply(weights)

    formatted_weights = pd.Series(["{0:.2f}%".format(val * 100) for val in␣
 ↪title_weight], index = title_names)
    formatted_expected_return = pd.Series(["{0:.2f}%".format(val * 100) for val␣
 ↪in mu], index = title_names)
    formatted_portfolio_performance = pd.Series(["{0:.2f}%".format(val * 100)␣
 ↪for val in portfolio_performance], index = title_names)
```

```
    frame = {
        "Weights": weights,
        "Expected return": mu,
        "Portfolio performance": portfolio_performance,
        "Formatted weights": formatted_weights,
        "Formatted expected return": formatted_expected_return,
        "Formatted portfolio performance": formatted_portfolio_performance
    }

    portfolio_dataframe = pd.DataFrame(frame)

    expected_annual_return, volatility, sharpe_ratio = ef.
 ↪portfolio_performance(verbose=True)

    return portfolio_dataframe, ef


def back_testing(tickers, tickers_list_name, date_start="2012-01-01",␣
 ↪date_end="2020-10-10"):
    prices = get_tickers_adj_close_values(tickers, tickers_list_name,␣
 ↪date_start, date_end)

    step = 20
    next_price_point = 20
    N = len(prices) - (step + next_price_point)
    analysis_period_days = 252 * 4
#     print(len(prices))

    weights_dataframe = prices
#     print(weights_dataframe)
    daily_performance_complete = pd.DataFrame()

    for i in range(0, N, step):
        analysis_period_start = i
        analysis_period_end = i + analysis_period_days
        analysis_period = prices[analysis_period_start:analysis_period_end]
        analysis_period_next_price_point = analysis_period_end +␣
 ↪next_price_point

        montly_return_period = prices[analysis_period_end:
 ↪analysis_period_next_price_point]


        print(i, i + analysis_period_days, analysis_period_next_price_point)
        dataframe_portion = prices[analysis_period_start:analysis_period_end] #␣
 ↪Green area
```

```
        analysis_period_next_price_point_data = prices[
            analysis_period_next_price_point:analysis_period_next_price_point+1
        ]  # Red line

#         print(dataframe_portion)
#         print(analysis_period_next_price_point_data)


        analysis_period_porfolio, _ = get_best_portfolio(tickers,␣
 ↪tickers_list_name=tickers_list_name,
                                                      ␣
 ↪tickers_dataframe=dataframe_portion)
#         print(montly_return_period.pct_change())
        weights = analysis_period_porfolio['Weights']

        for j in range(next_price_point):
#             partial_monthly_return = [analysis_period_end:
 ↪analysis_period_end+j+1]
            partial_monthly_return_data = prices[analysis_period_end:
 ↪analysis_period_end+j+1]
            partial_monthly_return_data_change = partial_monthly_return_data.
 ↪pct_change()
#             print(partial_monthly_return_data_change)

            last_partial_monthly_return_data_change =␣
 ↪partial_monthly_return_data_change.tail(1)
#             print(last_partial_monthly_return_data_change)

            portfolio_daily_return = weights.
 ↪multiply(last_partial_monthly_return_data_change)
            print_full_dataframe(portfolio_daily_return)
            daily_performance = portfolio_daily_return.sum(axis=1)
            print('\n'*2)
            print(daily_performance)
    #             portfolio_returns = sum(portfolio_daily_return)
#             print(portfolio_returns)
            print("\n"*5)
            daily_performance_complete = pd.concat([daily_performance_complete,␣
 ↪daily_performance])



        # use current month weights
        #
```

```
#         break
    print("\n"*3)
    print(daily_performance_complete)
    daily_performance_complete.to_csv("portfolio_returns.csv")


tickers =␣
 ↪"AAPL,ABBV,ABT,ACN,AGN,AIG,ALL,AMGN,AMZN,AXP,BA,BAC,BIIB,BK,BLK,BMY,C,CAT,CELG,CL,CMCSA,COF
tickers = tickers.split(',')

porfolio, _ = get_best_portfolio(tickers,"S&P_100")
print(porfolio)
print(porfolio['Weights'].to_csv("weights.csv"))
# print("\n"*5)
# back_testing(tickers, "S&P_100")
```

```
Loading…
Expected annual return: 27.1%
Annual volatility: 16.4%
Sharpe Ratio: 1.53
          Weights  Expected return  Portfolio performance Formatted weights  \
Symbols
AAPL     0.074985         0.288653               0.021645             7.50%
ABT      0.000000         0.197675               0.000000             0.00%
ACN      0.000000         0.207762               0.000000             0.00%
AIG      0.000000         0.043610               0.000000             0.00%
ALL      0.000000         0.171002               0.000000             0.00%
…            …              …                    …                 …
USD      0.000000         0.369875               0.000000             0.00%
V        0.046985         0.277829               0.013054             4.70%
VZ       0.000000         0.096414               0.000000             0.00%
WBA      0.000000         0.036396               0.000000             0.00%
WFC      0.000000         0.017901               0.000000             0.00%

        Formatted expected return Formatted portfolio performance
Symbols
AAPL                       28.87%                           2.16%
ABT                        19.77%                           0.00%
ACN                        20.78%                           0.00%
AIG                         4.36%                           0.00%
ALL                        17.10%                           0.00%
…                            …                             …
USD                        36.99%                           0.00%
V                          27.78%                           1.31%
VZ                          9.64%                           0.00%
WBA                         3.64%                           0.00%
WFC                         1.79%                           0.00%
```

```
[86 rows x 6 columns]
None
```