1. Download data and primary data process

In [ ]:

```
import pandas as pd
import yfinance
import matplotlib.pyplot as plt
from pandas_datareader import data as pdr
from pandas_datareader.famafrench import get_available_datasets

# download rf and rm
get_available_datasets()
ff = pdr.DataReader('F-F_Research_Data_Factors_daily', 'famafrench', start='1991-1-1')[0]
mm = pdr.DataReader('F-F_Momentum_Factor_daily', 'famafrench', start='1991-1-1')[0]
factors = pd.merge(ff, mm, left_index=True, right_index=True)
factors.head()

yfinance.pdr_override()

price = pd.DataFrame()
volume = pd.DataFrame()
tickers = ['AAPL', 'MSFT', 'NVDA', 'BABA', '^TNX', 'GOLD', 'DAL', 'PFE', 'BAC', 'DIS']

for i in tickers:
    price[i] = pdr.get_data_yahoo(i, start='1991-1-1')['Adj Close']
    volume[i] = pdr.get_data_yahoo(i, start='1991-1-1')['Volume']
price.head()

sec_returns = ((price / price.shift(1)) - 1) * 100
sec_returns = sec_returns.dropna()
sec_returns.head()

total = pd.merge(factors, sec_returns, left_index=True, right_index=True)
total = total.dropna()
total.head()

writer1 = pd.ExcelWriter('merge_data.xlsx')
writer2 = pd.ExcelWriter('data.xlsx')
total.to_excel(writer1, index=False)
sec_returns.to_excel(writer2, index=False)
writer1.save()
writer2.save()

# data discribe
total.describe()
```

```python
# visualize
plt.rcParams['font.family'] = 'Arial Unicode MS'
plt.rcParams['axes.unicode_minus'] = False

for i in price:  # price line chart
    price[i].plot()
    plt.ylabel(f'{i} Price - $')
    plt.savefig(f'{i}.png')
    plt.show()
    plt.close('all')

for i in volume:  # volume line chart
    volume[i].plot()

plt.savefig('volume_line.png')
plt.show()

# volume bar chart
volume = volume.drop('^TNX', axis=1)
for i in volume:
    plt.bar(i, volume[i].mean())

plt.savefig('volume_bar.png')
plt.show()
```

1. Construct the portfolio: 4-Factors Model, Efficient frontier

In [ ]:

```python
import numpy as np
import pandas as pd
import numpy.random as npr
import matplotlib.pyplot as plt
import scipy.optimize as sco
import statsmodels.api as sm
plt.rcParams['font.family'] = 'Arial Unicode MS'
plt.rcParams['axes.unicode_minus'] = False

# read our portfolio's data
io = r'/Users/yimingzhang/PycharmProjects/fin240/group_project/New GW/merge_data.xlsx'
total = pd.read_excel(io)

io = r'/Users/yimingzhang/PycharmProjects/fin240/group_project/New GW/data.xlsx'
sec_returns = pd.read_excel(io)
```

```python
# we use 4-factors module to compute the beta of our portfolio
tickers = ['AAPL', 'MSFT', 'NVDA', 'BABA', '^TNX', 'GOLD', 'DAL', 'PFE', 'BAC', 'DIS']

sec_beta = pd.DataFrame(np.nan, index=tickers, columns=['const', 'Mkt-RF', 'SMB', 'HML'])
for t in tickers:
    X = total[['Mkt-RF', 'SMB', 'HML', 'Mom   ']]
    X1 = sm.add_constant(X)
    Y = total[t] - total['RF']
    reg = sm.OLS(Y, X1).fit()
    sec_beta.loc[t, :] = reg.params
sec_beta

# Compute annually market return, risk-free rate
rm_minus_rf = (np.exp(np.mean(np.log(total['Mkt-RF'] / 100 + 1))) ** 252) - 1

rf = (np.exp(np.mean(np.log(total['RF'] / 100 + 1))) ** 252) - 1

# Using CAPM module to compute the theoretical annual return
expected_returns = []
for i in sec_returns:
    r_annually = (sec_beta['Mkt-RF'][i] * rm_minus_rf) + rf
    expected_returns = np.append(expected_returns, r_annually)
print(expected_returns)

# calculate the cov matrix:
cov_matrix = sec_returns.cov() * 252 / 100

# use Monte Carlos method to construct 200,000 portfolio, compute every portfolio's return and variation.
# plot a return-sigma scatter plot.
# To make it easier, we assume there's no short sell. Therefore, each weight vector is between 0-1
# plot a return-sigma scatter plot——Efficient frontier
number_assets = 10
portfolio_returns = []
portfolio_sigma = []
sharpe_ratio = []
for single_portfolio in range(200000):
    weights = np.random.random(number_assets)
    weights = weights / (np.sum(weights))
    returns = np.dot(weights, expected_returns)
    sigma = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    portfolio_returns.append(returns)
    portfolio_sigma.append(sigma)
    sharpe = (returns - rf) / sigma
    sharpe_ratio.append(sharpe)
portfolio_returns = np.array(portfolio_returns)
portfolio_sigma = np.array(portfolio_sigma)
```

```python
plt.style.use('seaborn-dark')
plt.figure(figsize=(9, 5))
plt.scatter(portfolio_sigma, portfolio_returns)
plt.grid(True)
plt.xlabel('expected sigma')
plt.ylabel('expected return')
plt.savefig('Efficient frontier.png')
plt.show()


# we choose the portfolio with highest Sharpe-ratio, which equivalent to minimum -(sharpe_ratio).
# In that case we can use the minimization optimization algorithm sco.minimize to find the optimal portfolio.
# The boundary condition is each item of the weight needs to be between 0 and 1,
# and the constraint condition is that the sum of the weights is 1.

def statistics(weights):
    weights = np.array(weights)
    port_returns = np.dot(expected_returns, weights)
    port_sigma = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return (port_returns - rf) / port_sigma


def min_func_sharpe(weights):
    return -statistics(weights)

# minimization optimization algorithm sco.minimize to find the optimal portfolio
bnds = tuple((0, 1) for x in range(number_assets))  # limit weight in [0,1)
cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
opts = sco.minimize(min_func_sharpe, number_assets * [1. / number_assets, ], method='SLSQP', bounds=bnds,
                    constraints=cons)

# we print out the weight and the sharpe-ratio of the 10 assets.
weights = opts['x'].round(10)
sharpe_ratio = np.dot(opts['x'], sec_beta['Mkt-RF']).round(10)
print(f'the weights are:{weights}')
print(f'the sharpe-ratio of our portfolio is {sharpe_ratio}')
```

1. run OLS regression and t-test

In [ ]:

```python
import numpy as np
import pandas as pd
from pandas_datareader import data as pdr
import scipy.stats as sst
```

```python
import matplotlib.pyplot as plt
import yfinance as yf

yf.pdr_override()

tickers = ['AAPL', 'MSFT', 'NVDA', 'BABA', '^TNX', 'GOLD', 'DAL', 'PFE', 'BAC', 'DIS']
stocklist = {'AAPL': 'Apple', 'MSFT': 'Microsoft', 'NVDA': 'Nvidia', 'BABA': 'Alibaba', '^TNX': 'Treas Yld Index',
             'GOLD': 'Barrick', 'DAL': 'Delta', 'PFE': 'Pfizer', 'BAC': 'Bank of America', 'DIS': 'Walt Disney'}
weight = [0.17070402, 0.26714263, 0.04696567, 0.00771267, 0.00833391, 0.03757462,
          0.03810221, 0.13953412, 0.21689562, 0.06703453]

# read data we got previously and change the scale
price = pd.DataFrame()
for i in tickers:
    price[i] = pdr.get_data_yahoo(i, start='2017-3-11', end='2020-3-11')['Adj Close']
sec_returns1 = ((price / price.shift(1)) - 1)

price = pd.DataFrame()
for i in tickers:
    price[i] = pdr.get_data_yahoo(i, start='2020-3-11', end='2023-3-11')['Adj Close']
sec_returns2 = ((price / price.shift(1)) - 1)

# compute the portfolio's return
portfolio = pd.DataFrame()
portfolio['before'] = np.dot(sec_returns1, weight)
portfolio['after'] = pd.Series(np.dot(sec_returns2, weight))
portfolio = portfolio.dropna()

# Descriptive Statistical Analysis
portfolio.describe()
# plot the portfolio's price line plot
portfolio['before'].plot()
portfolio['after'].plot()
plt.legend()
plt.savefig('portfolio_return comparison.png')
plt.show()

# h-test, we want to see whether our portfolio's mean is:
# equal before and after 2020-3-11  --(null hypothesis)
# or not -- (Alternative hypothesis)

# since we are using one porfolio from different time scale, we should use relative t-test

p_value = sst.ttest_1samp(portfolio['before'], popmean=0.000, nan_policy='omit')
print(f'mean before mean = 0 ttest result: {p_value}')
```

```
p_value = sst.ttest_1samp(portfolio['after'], popmean=0.000, nan_policy='omit')
print(f'mean after mean = 0 result: {p_value}')

p_value = sst.ttest_rel(a=portfolio['before'], b=portfolio['after'], nan_policy='omit')
print(f'mean before = after result: {p_value}')


# h-test, we want to see whether our portfolio's var is:
# equal before and after 2020-3-11  --(null hypothesis)
# or not -- (Alternative hypothesis)
F_stat = max(portfolio['before'].var() / portfolio['after'].var(), portfolio['after'].var() / portfolio['before'].var())
p_value = 1 - sst.f.cdf(F_stat, dfn=portfolio['after'].count() - 1, dfd=portfolio['before'].count() - 1)
print(f'var t-test result: {p_value}')
```

4.visualize price over time as supplementary material

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas_datareader import data as pdr
import yfinance as yf

yf.pdr_override()

# visualize portfolio's price and trading volume

weight = [0.17070402, 0.26714263, 0.04696567, 0.00771267, 0.00833391, 0.03757462,
          0.03810221, 0.13953412, 0.21689562, 0.06703453]

price = pd.DataFrame()
volume = pd.DataFrame()
tickers = ['AAPL', 'MSFT', 'NVDA', 'BABA', '^TNX', 'GOLD', 'DAL', 'PFE', 'BAC', 'DIS']
for i in tickers:
    price[i] = pdr.get_data_yahoo(i, start='2017-3-11')['Adj Close']

price['portfolio'] = np.dot(price, weight)
price['portfolio'].plot()
plt.ylabel('Portfolio Price - $')
plt.savefig('portfolio_price.png')
plt.show()
plt.close('all')
```