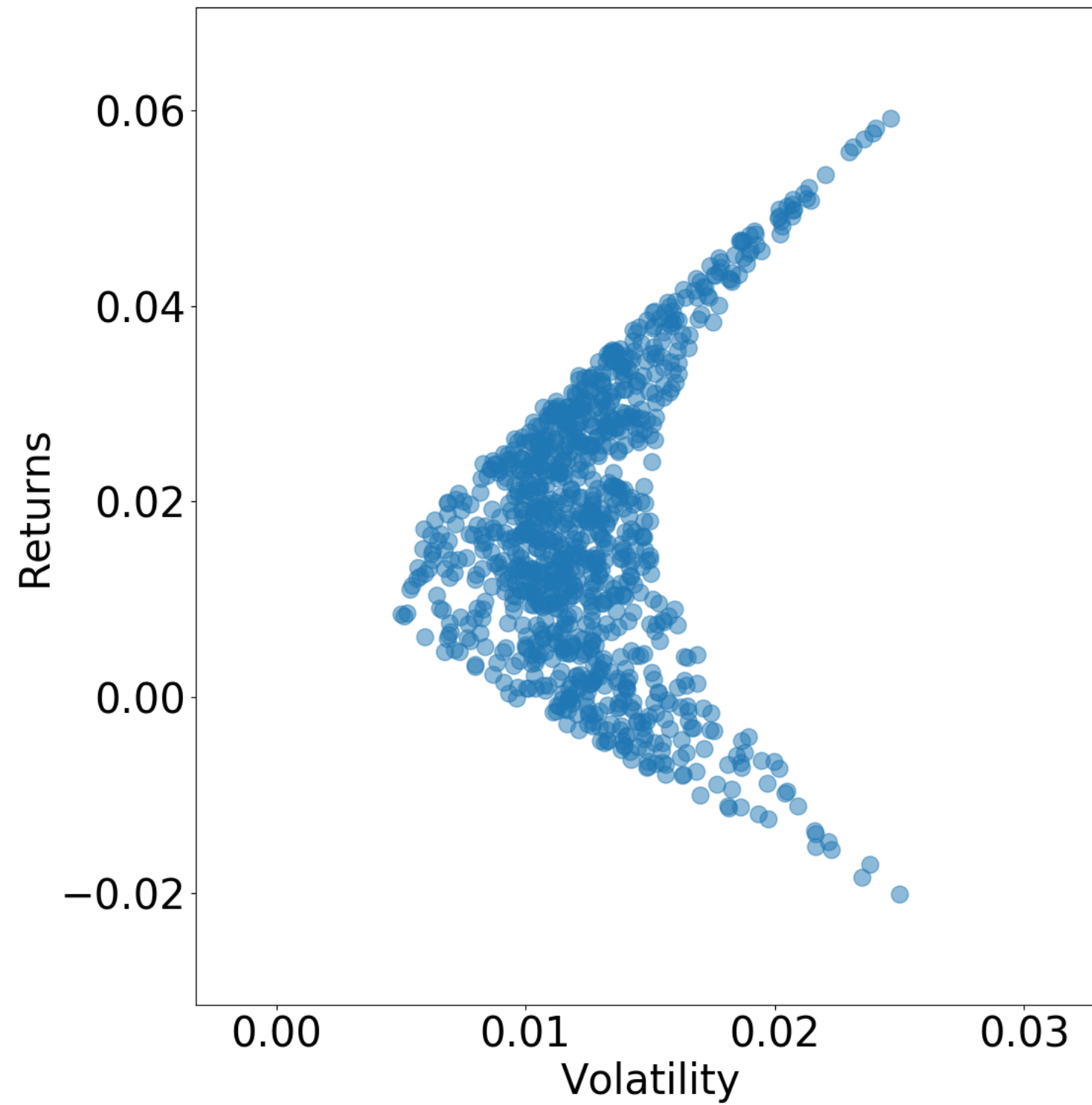MACHINE LEARNING FOR FINANCE IN PYTHON

Modern portfolio theory (MPT); efficient frontiers

Nathan George
Data Science Professor

Joining data

```python
stocks = ['AMD', 'CHK', 'QQQ']
full_df = pd.concat([amd_df, chk_df, qqq_df], axis=1).dropna()
full_df.head()
```

```
                AMD         CHK          QQQ
Date
1999-03-10    8.690    0.904417    45.479603
1999-03-11    8.500    0.951617    45.702324
1999-03-12    8.250    0.951617    44.588720
1999-03-15    8.155    0.951617    45.880501
1999-03-16    8.500    0.951617    46.281398
```

Calculating returns

```python
# calculate daily returns of stocks
returns_daily = full_df.pct_change()

# resample the full dataframe to monthly timeframe
monthly_df = full_df.resample('BMS').first()

# calculate monthly returns of the stocks
returns_monthly = monthly_df.pct_change().dropna()

print(returns_monthly.tail())
```

```
                 AMD       CHK       QQQ
Date
2018-01-01  0.023299  0.002445  0.028022
2018-02-01  0.206740 -0.156098  0.059751
2018-03-01 -0.101887 -0.190751 -0.020719
2018-04-02 -0.199160  0.060714 -0.052971
2018-05-01  0.167891  0.003367  0.046749
```

Covariances

```python
# daily covariance of stocks (for each monthly period)
covariances = {}
for i in returns_monthly.index:
    rtd_idx = returns_daily.index
    # mask daily returns for each month (and year) and calculate covariance
    mask = (rtd_idx.month == i.month) & (rtd_idx.year == i.year)
    covariances[i] = returns_daily[mask].cov()

print(covariances[i])
```

```
            AMD       CHK       QQQ
AMD   0.000257  0.000177  0.000068
CHK   0.000177  0.002057  0.000108
QQQ   0.000068  0.000108  0.000051
```

Generating portfolio weights

```python
for date in covariances.keys():
    cov = covariances[date]
    for single_portfolio in range(5000):
        weights = np.random.random(3)
        weights /= np.sum(weights)
```

Calcuclating returns and volatility

```python
portfolio_returns, portfolio_volatility, portfolio_weights = {}, {}, {}

# get portfolio performances at each month
for date in covariances.keys():
    cov = covariances[date]
    for single_portfolio in range(5000):
        weights = np.random.random(3)
        weights /= np.sum(weights)

        returns = np.dot(weights, returns_monthly.loc[date])
        volatility = np.sqrt(np.dot(weights.T, np.dot(cov, weights)))

        portfolio_returns.setdefault(date, []).append(returns)
        portfolio_volatility.setdefault(date, []).append(volatility)
        portfolio_weights.setdefault(date, []).append(weights)
```

Plotting the efficient frontier

```python
date = sorted(covariances.keys())[-1]

# plot efficient frontier
plt.scatter(x=portfolio_volatility[date],
            y=portfolio_returns[date],
            alpha=0.5)
plt.xlabel('Volatility')
plt.ylabel('Returns')
plt.show()
```

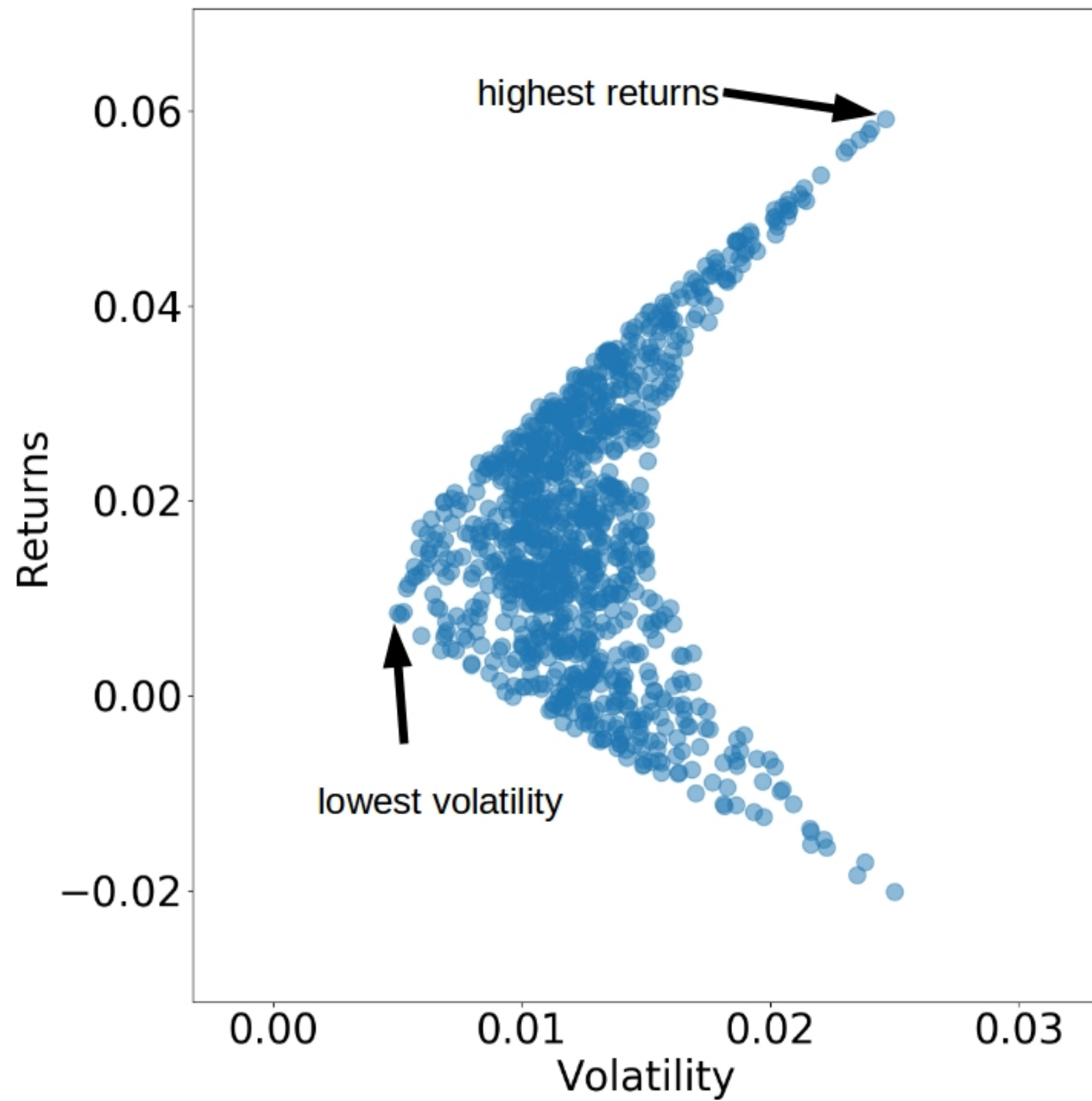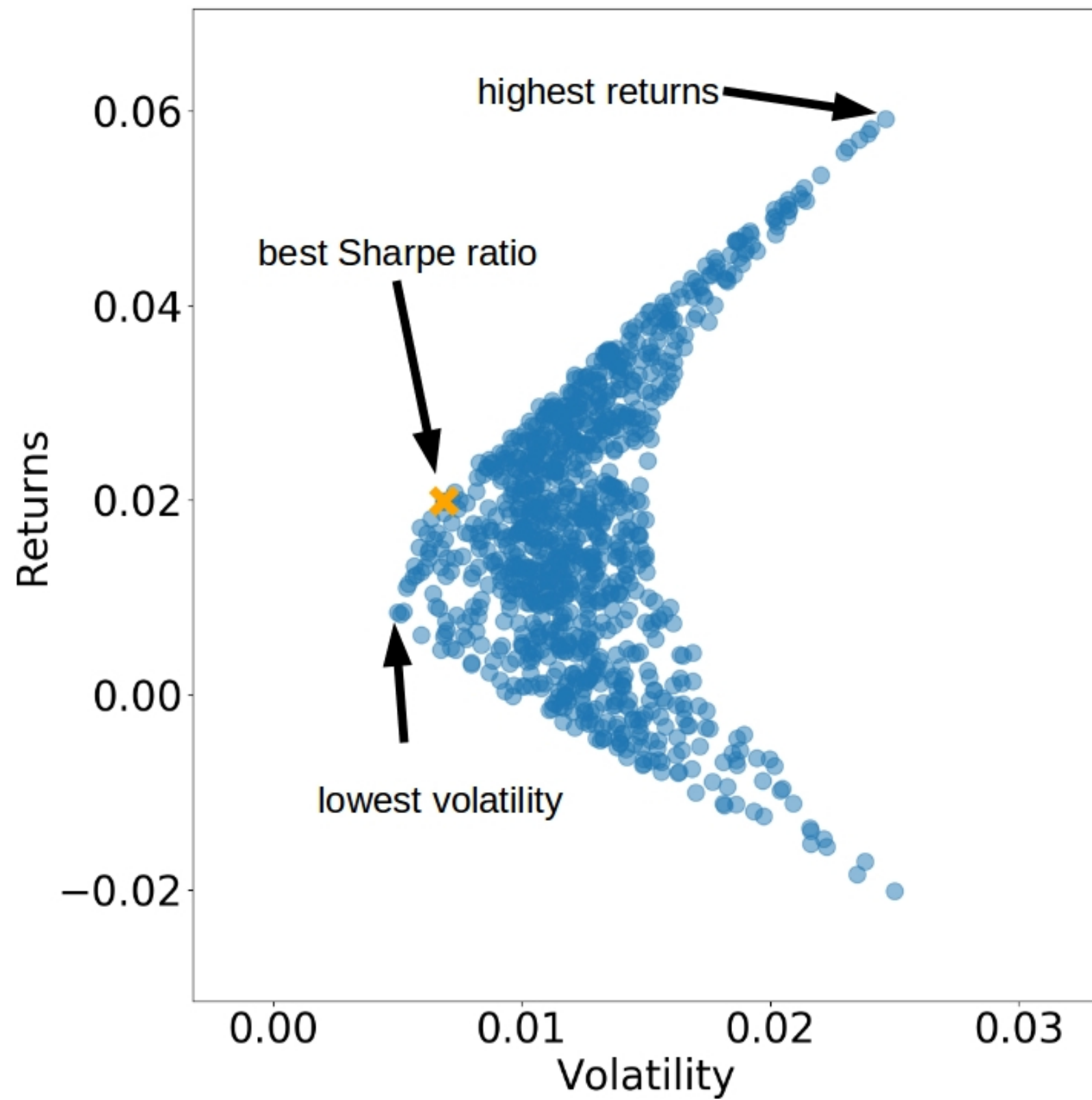MACHINE LEARNING FOR FINANCE IN PYTHON

Calculate MPT portfolios!

MACHINE LEARNING FOR FINANCE IN PYTHON

Sharpe ratios; features and targets

Nathan George
Data Science Professor

$$\text{Sharpe ratio} = \frac{\text{portfolio return} - \text{risk free return}}{\text{portfolio standard deviation}}$$

Getting our Sharpe ratios

```python
# empty dictionaries for sharpe ratios and best sharpe indexes by date
sharpe_ratio, max_sharpe_idxs = {}, {}

# loop through dates and get sharpe ratio for each portfolio
for date in portfolio_returns.keys():
    for i, ret in enumerate(portfolio_returns[date]):
        volatility = portfolio_volatility[date][i]
        sharpe_ratio.setdefault(date, []).append(ret / volatility)

    # get the index of the best sharpe ratio for each date
    max_sharpe_idxs[date] = np.argmax(sharpe_ratio[date])
```

Create features

```python
# calculate exponentially-weighted moving average of daily returns
ewma_daily = returns_daily.ewm(span=30).mean()

# resample daily returns to first business day of the month
ewma_monthly = ewma_daily.resample('BMS').first()

# shift ewma 1 month forward
ewma_monthly = ewma_monthly.shift(1).dropna()
```

Calculate features and targets

```python
targets, features = [], []


# create features from price history and targets as ideal portfolio
for date, ewma in ewma_monthly.iterrows():
    # get the index of the best sharpe ratio
    best_idx = max_sharpe_idxs[date]
    targets.append(portfolio_weights[date][best_idx])
    features.append(ewma)

targets = np.array(targets)
features = np.array(features)
```

Re-plot efficient frontier

```python
# latest date
date = sorted(covariances.keys())[-1]

cur_returns = portfolio_returns[date]
cur_volatility = portfolio_volatility[date]


plt.scatter(x=cur_volatility,
            y=cur_returns,
            alpha=0.1,
            color='blue')

best_idx = max_sharpe_idxs[date]

plt.scatter(cur_volatility[best_idx],
            cur_returns[best_idx],
            marker='x',
            color='orange')

plt.xlabel('Volatility')
plt.ylabel('Returns')
plt.show()
```
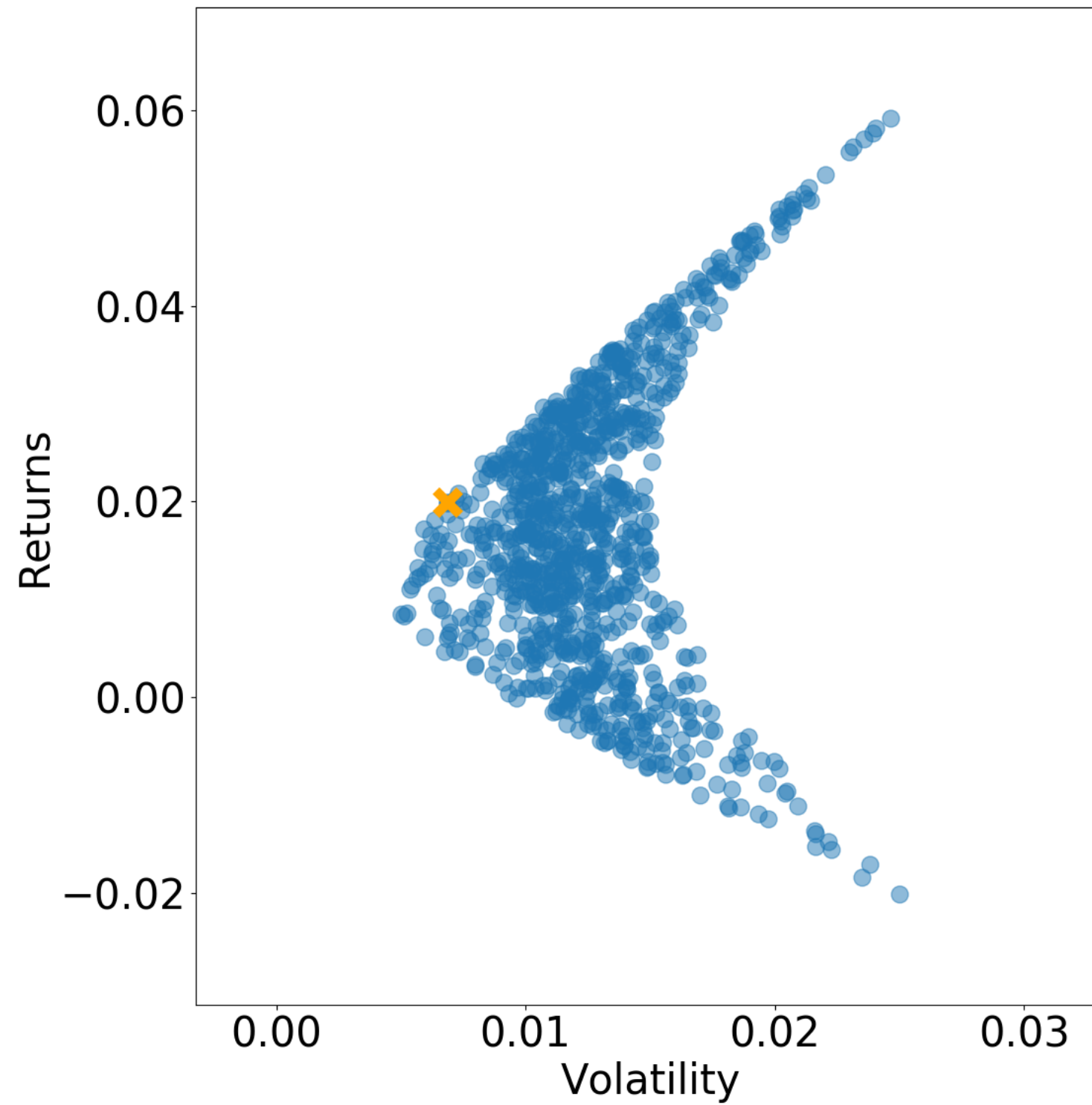
MACHINE LEARNING FOR FINANCE IN PYTHON

Get Sharpe!

MACHINE LEARNING FOR FINANCE IN PYTHON

Machine learning for MPT

Nathan George

Data Science Professor

Make train and test sets

```python
# make train and test features
train_size = int(0.8 * features.shape[0])
train_features = features[:train_size]
train_targets = targets[:train_size]

test_features = features[train_size:]
test_targets = targets[train_size:]
```

```python
print(features.shape)
```

```
(230, 3)
```

Fit the model

```python
from sklearn.ensemble import RandomForestRegressor

# fit the model and check scores on train and test
rfr = RandomForestRegressor(n_estimators=300, random_state=42)
rfr.fit(train_features, train_targets)

print(rfr.score(train_features, train_targets))
print(rfr.score(test_features, test_targets))
```

```
0.8382262317599827
0.09504859048985377
```

Evaluate the model's performance

```python
# get predictions from model on train and test
test_predictions = rfr.predict(test_features)

# calculate and plot returns from our RF predictions and the QQQ returns
test_returns = np.sum(returns_monthly.iloc[train_size:] * test_predictions,
                      axis=1)

plt.plot(test_returns, label='algo')
plt.plot(returns_monthly['QQQ'].iloc[train_size:], label='QQQ')
plt.legend()
plt.show()
```
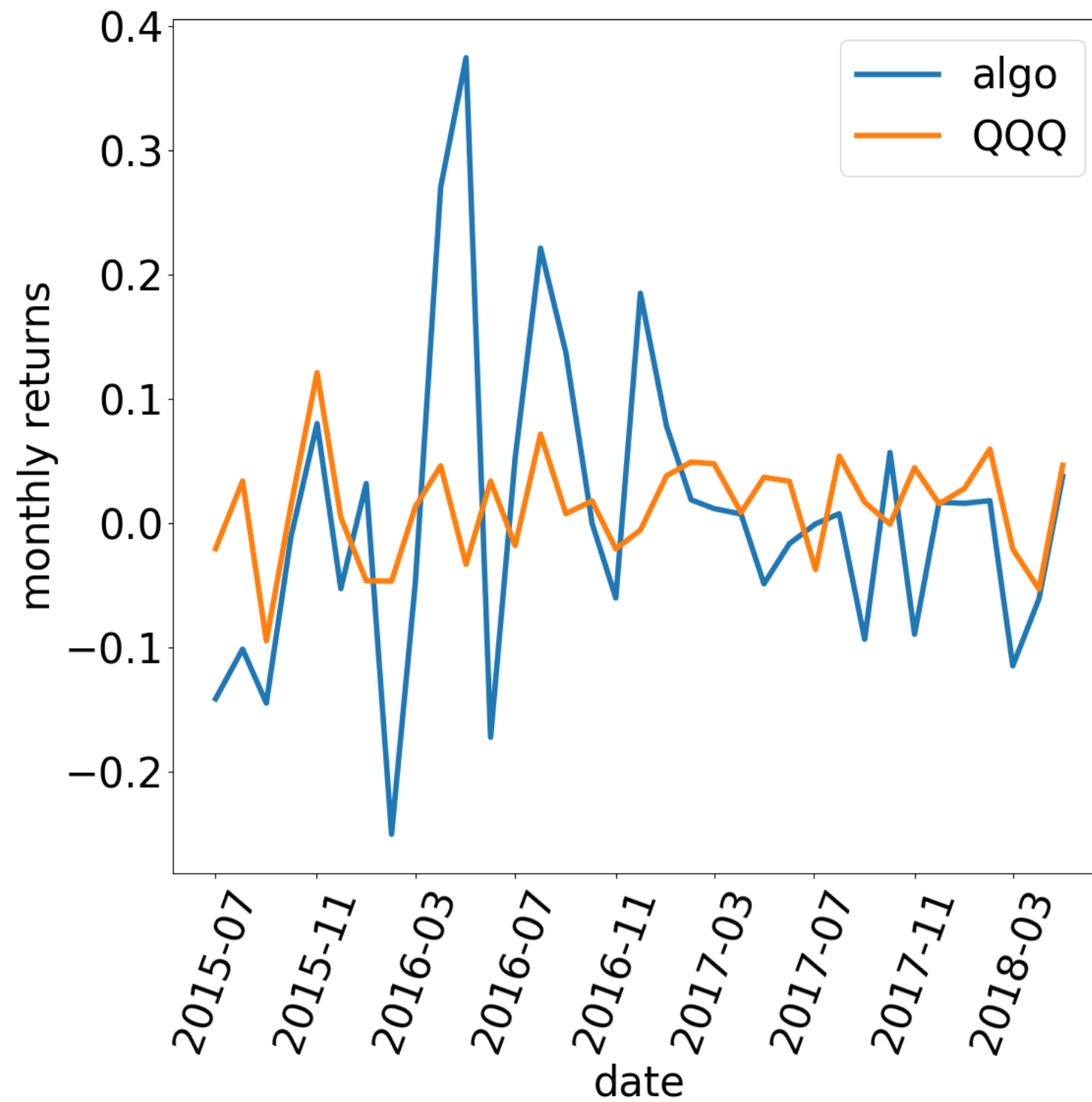
Calculate hypothetical portfolio

```python
cash = 1000
algo_cash = [cash]

for r in test_returns:
    cash *= 1 + r
    algo_cash.append(cash)

# calculate performance for QQQ
cash = 1000  # reset cash amount
qqq_cash = [cash]
for r in returns_monthly['QQQ'].iloc[train_size:]:
    cash *= 1 + r
    qqq_cash.append(cash)


print('algo returns:', (algo_cash[-1] - algo_cash[0]) / algo_cash[0])
print('QQQ returns:', (qqq_cash[-1] - qqq_cash[0]) / qqq_cash[0])
```
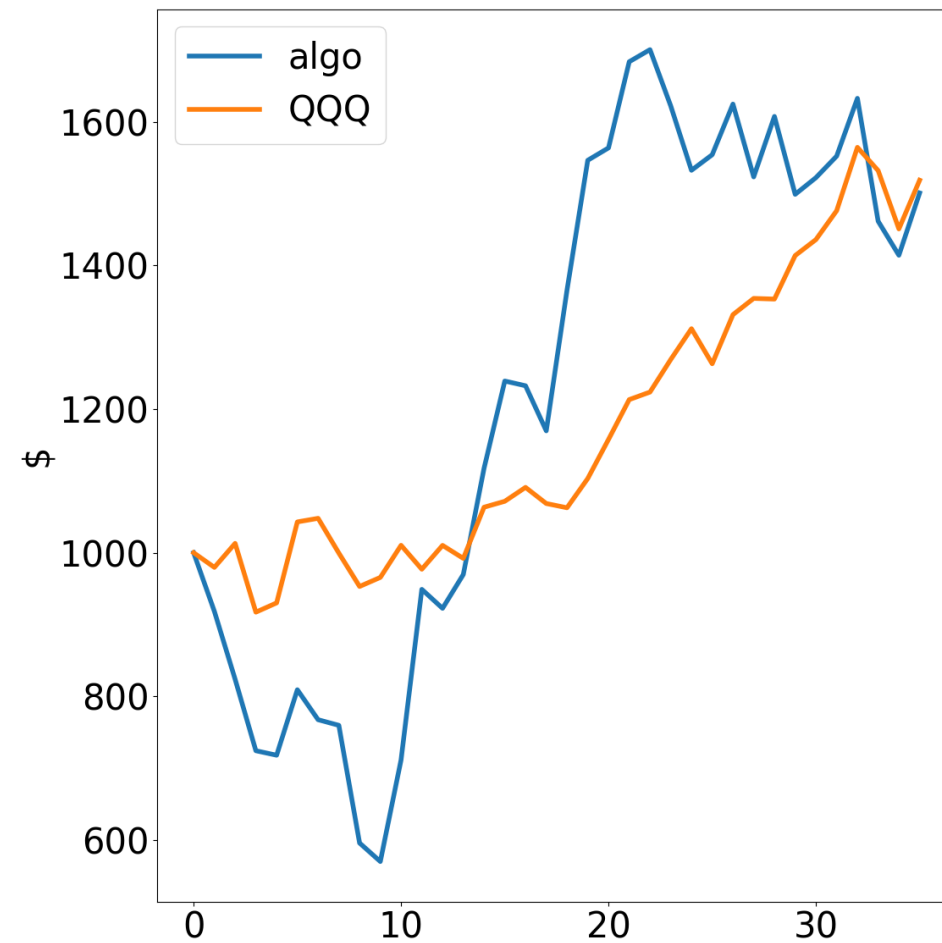
```
algo returns: 0.5009443507049591
QQQ returns: 0.5186775933696601
```

Plot the results

```python
plt.plot(algo_cash, label='algo')
plt.plot(qqq_cash, label='QQQ')
plt.ylabel('$')
plt.legend()  # show the legend
plt.show()
```

MACHINE LEARNING FOR FINANCE IN PYTHON

Train your model!

MACHINE LEARNING FOR FINANCE IN PYTHON

Final thoughts

Nathan George

Data Science Professor

Toy examples

Tools for bigger data:

- Python 3 multiprocessing

- Dask

- Spark

- AWS or other cloud solutions

Get more and better data

Data in this course:

- From Quandl.com/EOD (free subset available)

Alternative and other data:

- satellite images

- sentiment analysis (e.g. PsychSignal)

- analyst predictions

- fundamentals data

MACHINE LEARNING FOR FINANCE IN PYTHON

Be careful, and Godspeed!