Algorithmically Finding Suitable Pairs or Group Trades

Ryan Lagasse and Sahil Gandhi

Objective

- We were interested to see if we could use ML to generate good trading pairs or groups. While correlation matrices are industry standard, ML has the potential for greater explainability and in depth knowledge. Further research in this area could prove to be fruitful especially for beginners that lack the intuition for statistical models that can be hard to interpret.

Background in Pairs Trading

- Aims to profit from the temporary mispricing between two historically correlated stocks/securities
- Core idea is to identify two stocks that have historically high correlation and moved together and bet that the spread between the two stocks will eventually converge back to its historical norm
- Now with statistics arbitrage largely dead we want to evaluate ML augmented techniques and see how they perform

Data Science Pipeline

Scraping Data

Web Scraping Techniques such as using BeautifulSoup YF API and Selenium

Cleaning Data

Dropped all stocks with incomplete data (went public/private during our time frame)

Model Testing

We tested clustering algorithms like DB-SCAN along with PCA and DNNs

Testing Model

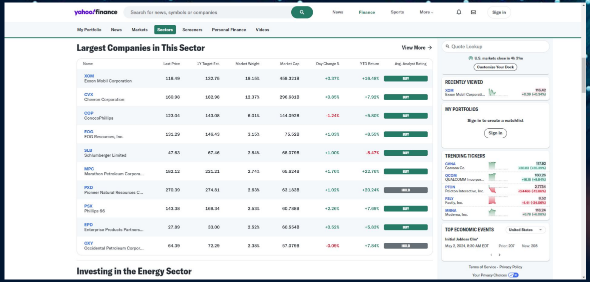
Created trading bot using professional policies to evaluate pairs (we understand this isn't perfect)

Results

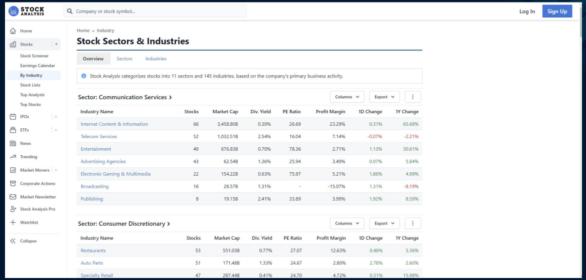
Graphed our results Compared between different models considering F1 and ROI (gross, net)

Data Collection

1. **Collecting Industry’s and Tickers**
   - Used Beautifulsoup to get industries
   - Used Industries to collect tickers for each



2. **Get Stock Data for Training**
   - Used YF to collect stock data into our dataset
   - We want this in industry batches to make computational costs lower and keep industry constant



Data Cleaning

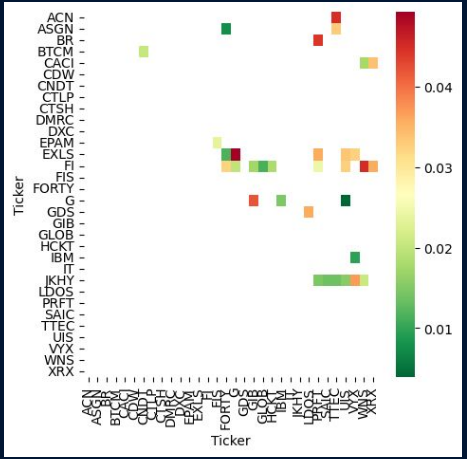
1. **Remove Incomplete Entries**
   - Many stocks go private/public so removing these is essential to get complete training data to learn correlation over the 6 month period
2. **Remove stagnant stocks**
   - Stocks with low trade velocity are not ideal fo pairs trading so we remove them off the bat

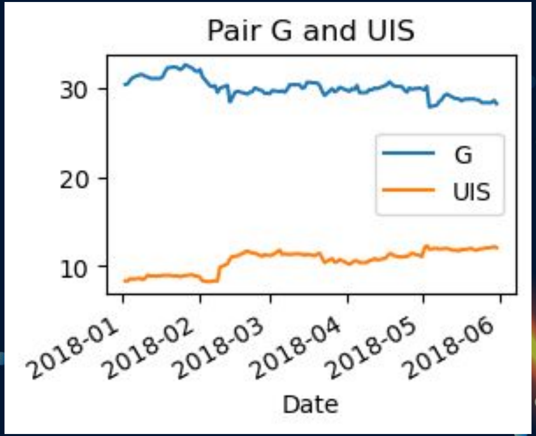
	ACN	ASGN	BR	BTCM	CACI	CDW	CNDT	CSPI	CTLP	CTSH	...	SAIC	TTEC	UIS	VN
Date															
2018-01-02	139.834549	63.360001	81.668030	107.500000	134.149994	64.948296	16.540001	5.839521	9.75	65.117874	...	69.789619	34.718552	8.35	8
2018-01-03	140.479935	63.110001	81.596077	114.400002	137.800003	66.847595	16.250000	5.473580	9.70	65.668556	...	69.636612	34.632179	8.35	9
2018-01-04	142.143356	64.190002	82.243652	114.400002	136.600006	68.020676	16.270000	5.644871	9.50	66.705711	...	70.158577	34.632179	8.60	8
2018-01-05	143.315918	65.250000	83.251015	116.900002	137.149994	67.908943	16.160000	5.578690	9.60	67.274742	...	70.185577	34.545818	8.55	9
2018-	144.461212	67.940002	83.763687	118.500000	138.250000	68.011368	16.459999	5.866773	9.50	67.338989	...	70.806519	35.107185	8.65	9

Calculating Stock Pairs

- Calculate stock return series
- Construct correlation matrix
- Look for highly correlated pairs
- Check spread stationarity
- Ensure fundamental reason for co-movement
- Rank pairs by correlation and stationarity

Corr(X,Y) = Cov(X,Y) / (σX * σY)
σX = Standard deviation of X
σY = Standard deviation of Y
Output:
N x N matrix with correlation coefficients





Trading Strategy For Model Evaluation

Trading Strategy

- Analyzes two stocksCalculates fair value using OLS regression
- Generates trading signals based on spread between stocks
- Enters positions (long/short) when spread deviates from fair value
- Applies stop-loss and profit-taking rules
- Accounts for transaction costs
- Backtests strategy over a defined period
- Tracks gross and net returns

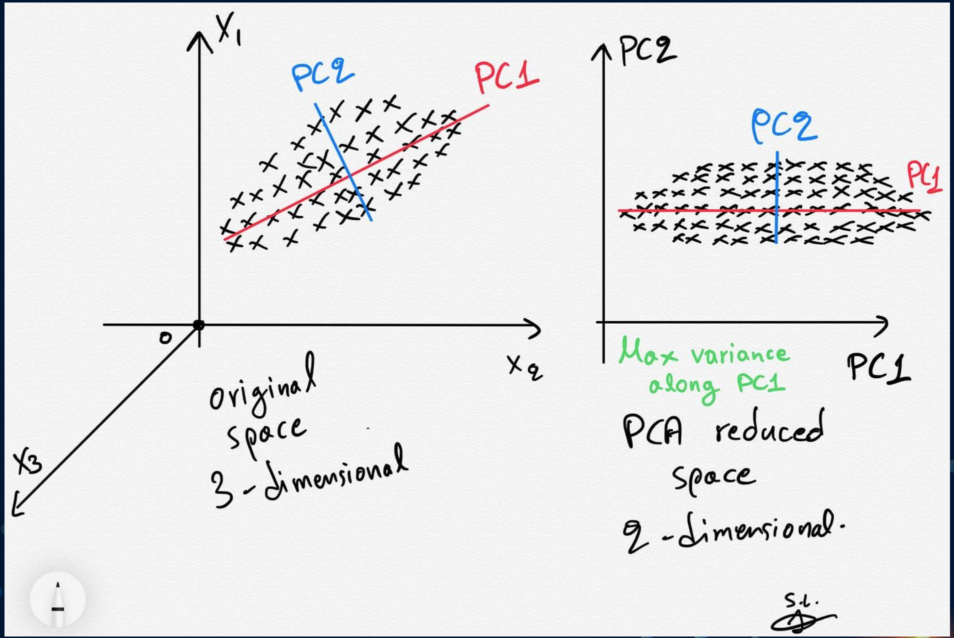
```python
def runSim(stock1, stock2):
    interface = 0
    stocks = [stock1,stock2]
    start = '2019-12-31'
    end = '2021-03-08'
    fee = 0.001
    window = 252
    t_threshold = -2.5

    data = pd.DataFrame()
    returns = pd.DataFrame()
    for stock in stocks:
        prices = yf.download(stock, start, end)
        data[stock] = prices['Close']
        returns[stock] = np.append(data[stock][1:].reset_index(drop=True)/data[stock][:-1].reset_index(drop=True) - 1 , 0)

    gross_returns = np.array([])
    net_returns = np.array([])
    t_s = np.array([])
    stock1 = stocks[0]
    stock2 = stocks[1]


    for t in range(window, len(data)):

        #defining the unit root function: stock2 = a + b*stock1
        def unit_root(b):
            a = np.average(data[stock2][t-window:t] - b*data[stock1][t-window:t])
            fair_value = a + b*data[stock1][t-window:t]
            diff = np.array(fair_value - data[stock2][t-window:t])
            diff_diff = diff[1:] - diff[:-1]
            reg = sm.OLS(diff_diff, diff[:-1])
            res = reg.fit()
            return res.params[0]/res.bse[0]

        res1 = spop.minimize(unit_root, data[stock2][t]/data[stock1][t], method='Nelder-Mead')
        t_opt = res1.fun
        b_opt = float(res1.x)
        a_opt = np.average(data[stock2][t-window:t] - b_opt*data[stock1][t-window:t])

        # Trade SIm
        fair_value = a_opt + b_opt*data[stock1][t]
        if t == window:
            old_signal = 0
        if t_opt > t_threshold:
            signal = 0
            gross_return = 0
        else:
            signal = np.sign(fair_value - data[stock2][t])
            gross_return = signal*returns[stock2][t] - signal*returns[stock1][t]
        fees = fee*abs(signal - old_signal)
        net_return = gross_return - fees
        gross_returns = np.append(gross_returns, gross_return)
        net_returns = np.append(net_returns, net_return)
        t_s = np.append(t_s, t_opt)

        # remove later
        if interface == 1:
            print('day '+str(data.index[t]))
            print('')
            if signal == 0:
                print('no trading')
            elif  signal == 1:
                print('long position on '+stock2+' and short position on '+stock1)
            else:
                print('long position on '+stock1+' and short position on '+stock2)
            print('gross daily return: '+str(round(gross_return*100,2))+'%')
            print('net daily return: '+str(round(net_return*100,2))+'%')
            print('cumulative net return so far: '+str(round(np.prod(1+net_returns)*100-100,2))+'%')
            print('')
```

Machine Learning Models

1. **PCA**
   - We used PCA to drop .95 and .9 variance and chose the closest remaining points. This was effective and had similar precision but lower recall
2. **PCA & DB-SCAN & Gridsearch**
   - We implemented PCA with DB-SCAN with Gridsearch next to find potential reading groups which worked effectively and found a successful group trade
3. **L2 & DB-SCAN**
   - We theorized L2 would drop sparsity, but this didn’t work in practice for us and made the model far too sensitive

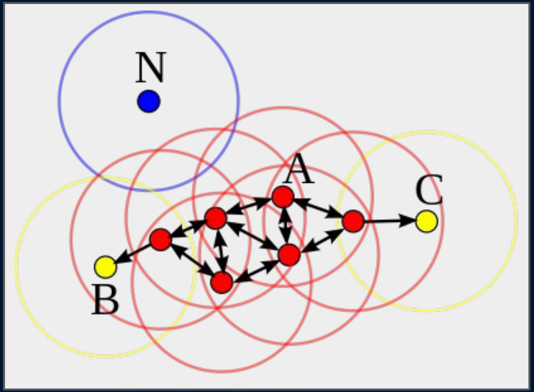
Principal Component Analysis (PCA)

- Dimensionality reduction technique
- Finds orthogonal directions of maximum variance
- Projects data onto new "principal component" axes
- Allows visualizing high-dimensional data in 2D/3D



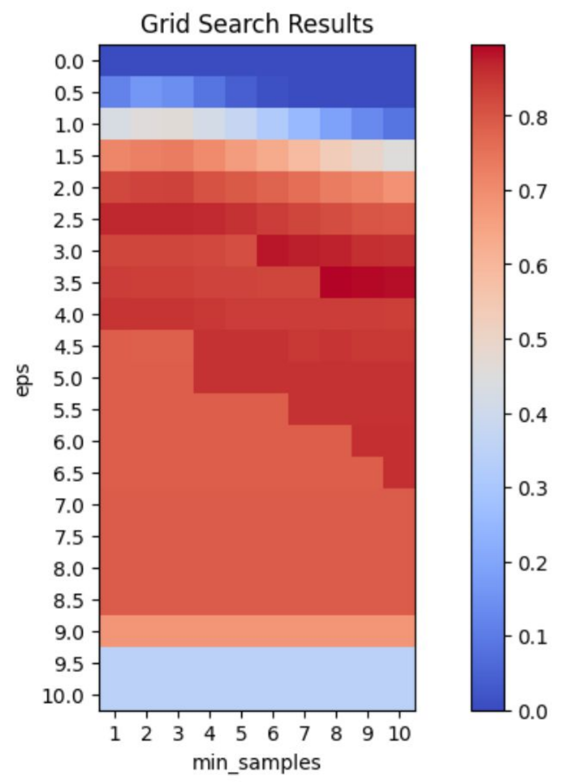
DB-SCAN

- Dimensionality reduction technique
- Finds orthogonal directions of maximum variance
- Projects data onto new "principal component" axes
- Allows visualizing high-dimensional data in 2D/3D



GridSearch

- Helps improve a model's performance by finding the best hyperparameters iteratively.
- Once it trains and evaluates each configuration, it picks the one that performs best on a held-out test set.
- For our purposes we just ran this to get pairs and used scaled correlation matrix to compare results



Example Output

Results

Correlation Baseline

```
ROI is 8.920862593567257 %
Total ROI is 10.030922492922922 %
```

PCA with DBSCAN & Gridsearch

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
 ROI is 2.758924638098259 %

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
 ROI is -2.4038380319772013 %

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
 ROI is 2.6225362469753266 %

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
 ROI is 0.948848262958113 %

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
 ROI is 6.507249389891956 %

 ROI is 7.436581981164658 %
Total ROI is 17.87030248711111 %
```

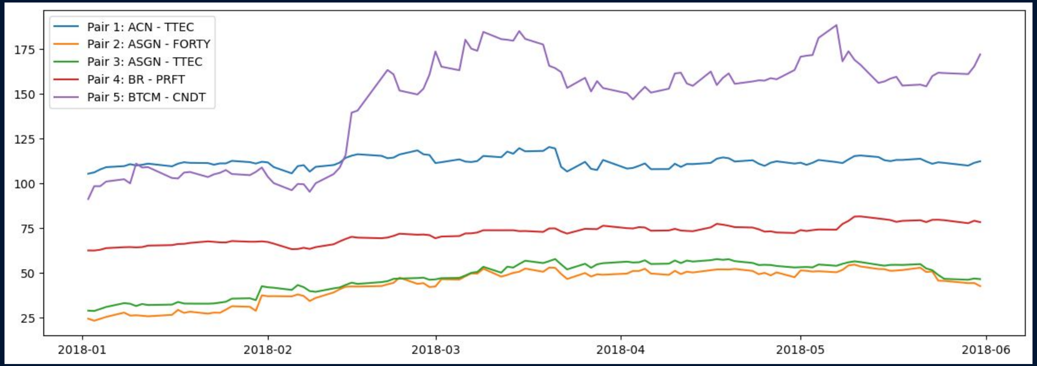
```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed-1.784879706799114

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed14.243955626312221

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed-0.20693000483447133

[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed-4.3341466601321095

4.105966210233558
12.023965464780083
```

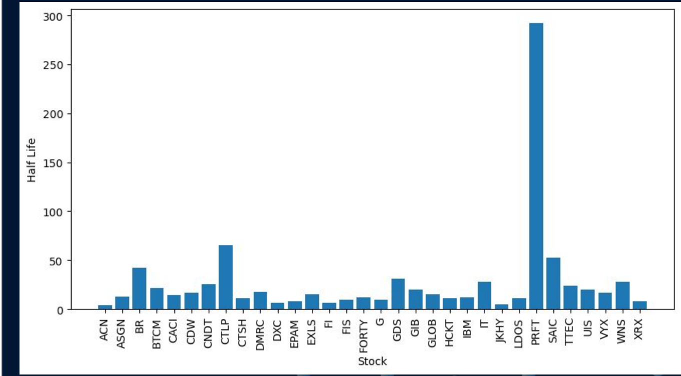
PCA

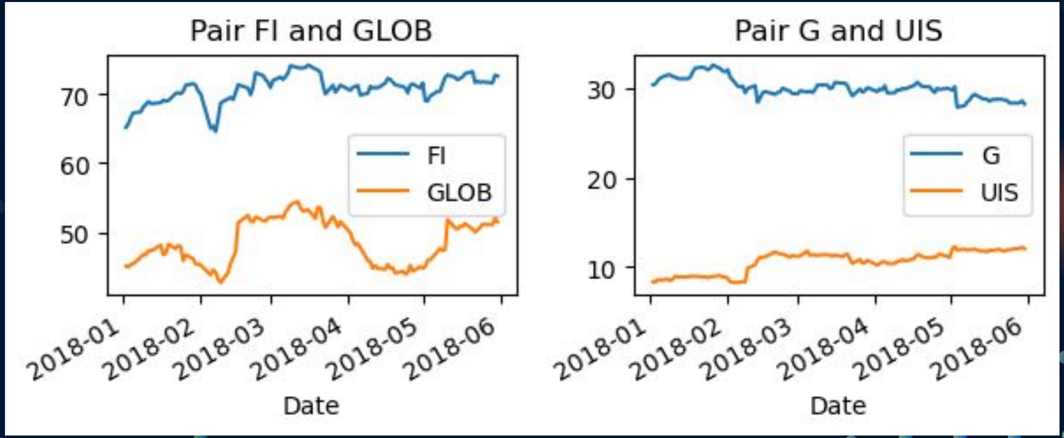
PCA with DBSCAN & Gridsearch

Analysis

F1: .8







Conclusion and Future Work

- **PCA with DB-SCAN and gridsearch** worked suspiciously well although the high variability of our metrics likely contributed to this and we would have to evaluate more on different time scales
- Very tricky to tune the hyperparameters & expensive to run gridsearch
- Our results were all from adjoining periods, so this may have lead to inflated results
- In the future we would like to run a DNN and use SHAP to get more explainable results or run a ViT on graphed pairs

Thank you!

Github Link