Pairs Trading strategy using Data Science

Kaushik Sanghani(I19MA013)

“Data science combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data”. We can use data science in various fields, including health care, technological development, experimental physics and finance. Here we will see how to use data science in financial trading. Hundreds of hedge funds, high-frequency trading firms and portfolio management companies use data science to generate wealth for investors and traders. These firms use time series analysis, advanced statistics, machine learning, deep learning and core concepts of calculus. Here we will develop Pairs Trading Strategy using statistics and time series analysis. We will also do backtesting and visualization of this trading strategy with the use of python and core concepts of data science.

Problem statement:
Make a pair trading strategy using statistical arbitrage, which can be in a live market.

Here we will use data science concepts to explain every step, from understanding the problem to the data visualization and deployment.

But before learning pairs, it is necessary to understand their mathematical principles. So in the section below, we will understand all the basic concepts about pairs trading and statistical arbitrage.

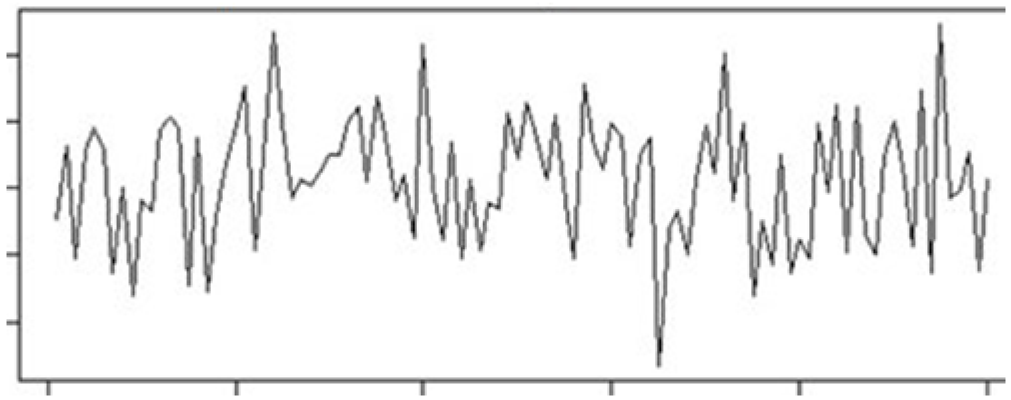
People trade in the stock market with various methods like luck, feeling, chart-reading, fundamental investing, technical trading, market microstructure, systematic trading, statistical arbitrage, machine learning modeling etc. Here we will discuss statistical arbitrage to make a pair trading strategy. Statistical arbitrage refers to a group of trading strategies that utilize mean reversion analyses to invest in diverse portfolios of up to thousands of securities for a very short period of time, often only a few seconds but up to multiple days.

Mathematical Understanding:

Directional Trading:
A strategy is called directional trading if the strategy has only leg long/short open at each point.

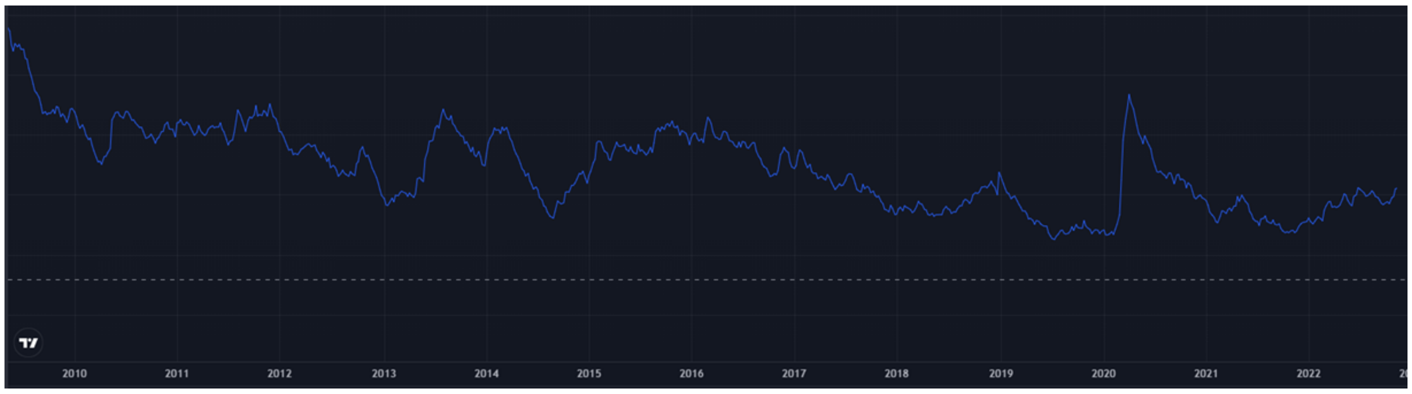
Pairs Trading:
A strategy is called pairs trading if the strategy has both sides, long and short open on the same or a different instrument.

Now we will use the concept of **“Stationarity”.** So a price series is said to be stationary if it doesn’t deviate much but stays around mean. For example, y = sin(x) is a stationary series, but y = root(x) is not stationary.



Above series is an example of a stationary series.

Random walk series are also not stationary. Stationarity is a long series concept, so to say any series is stationary, we have to take sufficiently large sample data. If we consider the price series (let's assume the example of NIFTY 50), it's not stationary; in the long term, the broad stock market tends to go up because of the economy's growth. This was an example, but most price series are not stationary. However, if we still figure out some logical connection between a price series, there can be stationarity. For example, Australian and Canadian dollar prices depend upon the commodity cycle because both nations are highly dependent upon commodity exports. So if we take the currency pair of AUD/CAD, it may be stationary, so let's look at the chart.



.

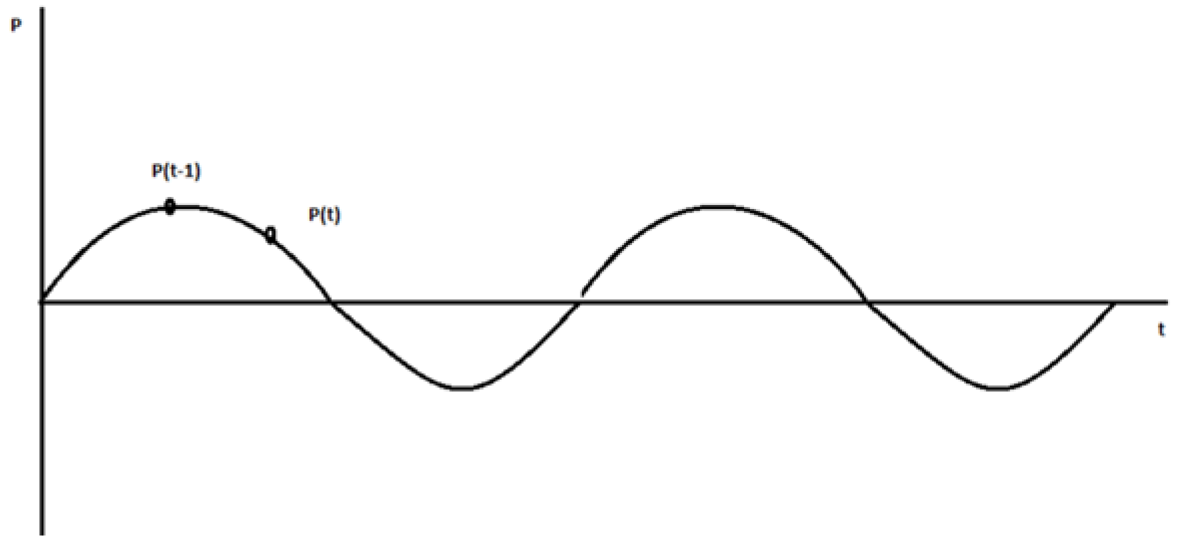
The above chart is a weekly AUDCAD pair from 2010, and we can observe that this series is stationary.

Now let’s devise a strategy for trading this currency pair.

But how to verify it? Is there any method to find whether the given series is stationary?

There is a statistical test called Augmented Dickey-Fuller (ADF) test to find whether a given series is stationary.

So now, let’s understand how the ADF test works.



In the above graph, X-axis represents time, and Y-axis represents price and P(t-1) is the price at time t-1, and P(t) is the price at t then,

P(t) – P(t-1) = Epsilon where the Epsilon is random noise, so the price series is not stationary. For this price series to become stationary, let’s add a constraint lambda such that
P(t) – P(t-1) = Epsilon + Lambda*P(t-1). If Lambda > 0, the price series cannot be stationary (it will become an increasing series). Also, if Lambda = 0, then the above price series is non-stationary, so to be stationary, Lambda < 0.

If we consider the null hypothesis, if Lambda = 0, then the null hypothesis is rejected. But if we can reject the null hypothesis with sufficient confidence, then.
P(t) is inversely dependent upon the P(t-1).

In the above example, we have taken the lag period as one, but after generalization, we can write that.

$$\Delta y(t) = \lambda y(t-1) + \mu + \beta_t + \alpha_1 \Delta y(t-1) + \ldots + \alpha_k \Delta y(t-k) + \epsilon t$$

y = price of the instrument

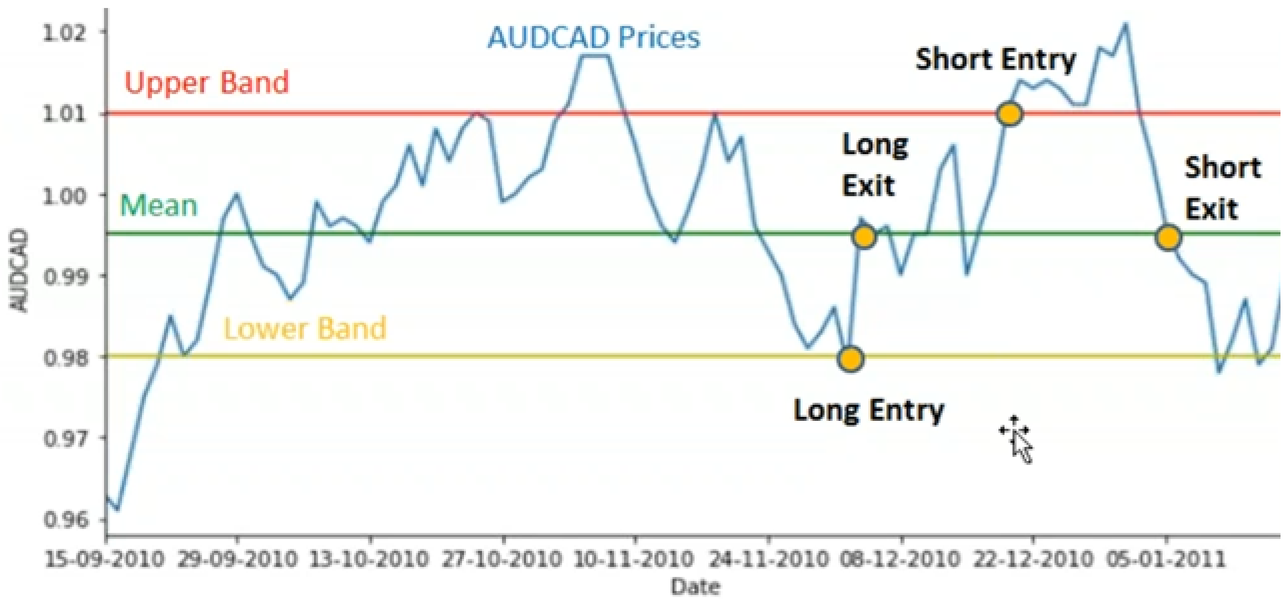
$\Delta y(t) = y(t) - y(t-1)$,

$\Delta y(t-1) = y(t-1) - y(t-2)$, *and so on.*

λ = *regression coefficient*

Intuitively, we can say that if the value of Lambda = -2.43 and the 95% confidence level = -2.42, then we can say with a 95% confidence interval that the given price series is stationary.

Now see the appendix for the calculation of the ADF test in python.



We have defined the hypothesis that the current price is a function of previous prices in a stationary price series, and the sample mean is not changing significantly. So we can define trading logic as follows.
Assumption: Prices are normally distributed around the mean.
So if the price goes high from its mean, then it becomes a shorting opportunity because prices will revert to the mean and vice versa. Let's define high price as a function of standard deviation, so if the prices goes above that (mean + 2*standard deviation), then that will be a good position to short, or if you are aggressive, then you can short at less than that level and cover your position when prices get back to mean. We can also define stop loss and target to make our system robust. But let's keep things simple here.

Now as we have defined earlier, most of the price series are not stationary. So does that mean we can't trade those instruments? Here comes the idea of cointegration.

Co-integration: A portfolio contains two or more instruments such that the portfolio is stationary, then the instrument in the portfolio is said to be co-integrated, and pairs trading is a particular example of cointegration trading strategy which contains only two stocks in a portfolio in this strategy We are doing a mean reversion strategy, so our logic will be selling when the value is high and buying when the value is low.

To understand this concept, let's take an example of India's two biggest IT companies. TCS and INFY have the same business in the same industry and sector. Both these companies are sufficiently large, so there is lot less opportunity for exponential growth. So prices of these companies depend upon the performance of the whole sector, so we can use statistical arbitrage to find which of the two is overvalued concerning each other. So we can short overvalued security and make a long position in under-valued security. So now, how will we use statistical arbitrage here?

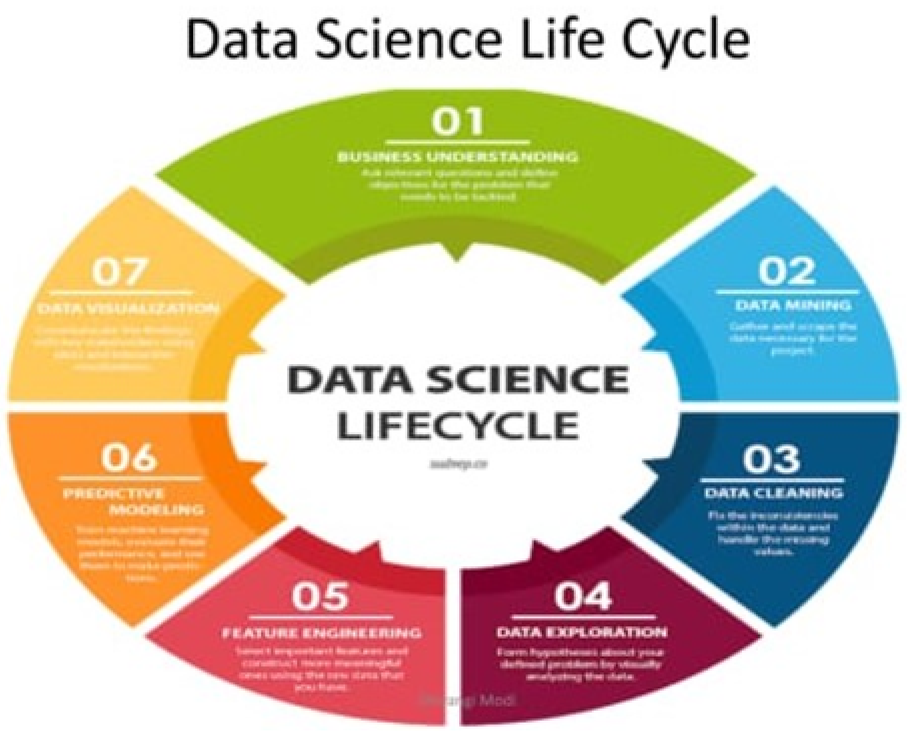
As we have discussed above, the prices of both companies depend upon the whole sector, so the price behavior of these two companies will be more or less similar. So can we take the spread of these two price series and check whether this spread is stationary or not? So we can use the ADF test, and if we can get a high confidence level, then we can apply the mean reversion strategy here.

<u>The strategy of Pair Trading:</u>

Find the spread of two price series, assuming its stationary, rolling moving average and rolling standard deviation of this price series. If the spread value is less than (moving average – 2* standard deviation), take a long position on one stock and short another and exit the position both when the spread crosses the value of the moving average. Do vice versa when the spread value is more significant than (moving average + 2*standard deviation)

So far, we have discussed the mathematics behind the pairs trading and have defined the strategy, but more is needed. We must assume that this will work in the real market. So to check the system's validity, we have to check on historical data and see whether this logic has worked in the past. This process is called backtesting, a whole field of study in itself. It is the strategy simulation in historical data, so we can get an idea about the actual return of strategy, average profit per trade, number of trades, maximum drawdown and expectancy ratio, Sharpe ratio etc. After checking the matrix, we can decide whether to trade this strategy on the live market.

Now we will see the backtesting process regarding the data science life cycle. Doing past data simulations is very similar to training the machine learning model. See the actual code implementation in APPENDIX II.


Data Science Life Cycle
DATA SCIENCE LIFECYCLE
01
BUSINESS UNDERSTANDING
Ask relevant questions and define objectives for the problem that needs to be tackled
02
DATA MINING
Gather and scrape the data necessary for the project.
03
DATA CLEANING
Fix the inconsistencies within the data and handle the missing values.
04
DATA EXPLORATION
Form hypotheses about your defined problem by visually analyzing the data.
05
FEATURE ENGINEERING
Select important features and construct more meaningful ones using the raw data that you have.
06
PREDICTIVE MODELING
07
DATA VISUALIZATION

Business Understanding:
So to do pair trading, it is vital to understand how the market works, how trading works, and what steps we can follow to backtest this strategy. So, domain knowledge is a must.

Data Mining:
We need to collect stock prices to do the historical data simulation. We can do this by downloading from NSE, S&P, and NASDAQ websites, etc. We can also use third-party data vendors like EODHistorical data, Capital-IQ, Quandl, Alphaventage or yahoo finance. Most of those above sources are paid, but we can use yahoo-finance python API to collect daily data. Here we have used yahoo finance.

Data Cleaning:
Data cleaning is a necessary part while working with financial modeling. We have to consider the data which are adjusted from the corporate actions. We have to deal with null values etc.

Data Exploration:
We need to choose those tickers only for co-integrated pair trading to reject the null hypothesis with high confidence intervals. We will see this part in actual code implementation.

Feature Engineering:
In this step, we will decide which tickers we want to trade and which pair is highly co-integrated from the given universe to get optimal results in the live market while trading this strategy.

Predictive Modeling:

Here we will use linear regression to find the relation between the two given pairs, so we can trade while keeping the market position neutral.

Data visualization:
We will analyze the trading strategy with different matrices and time series analysis.

Deployment:
Suppose the results are sounding and highly profitable. In that case, we can automate this strategy with brokers API and deploy it on third-party platforms like tradetron, algobulls or write the whole strategy code logic in a programming language and deploy it to cloud platforms like AWS.

Further plans:

Using optimization techniques, including stop loss and target to make the strategy robust.
Deploying this strategy with broker's API in aws or gcp.
Using options data in the backtesting and also using leverage in calculation.

Results:

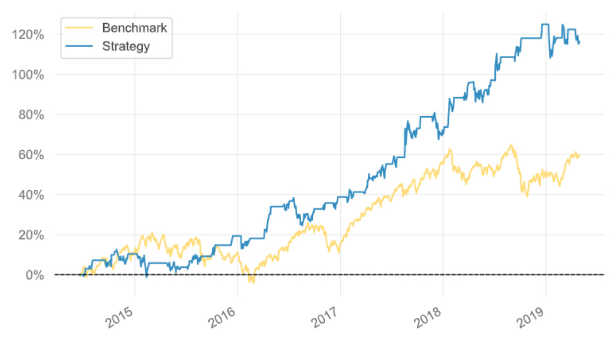
Our strategy has given 17.19% CAGR as compared to NIFTY 500 which has given only 10.05% CAGR. Which is an alpha of 7.14%. Further visualization is given below which includes all the key metrics, equity curve, yearly returns, monthly returns, maximum drawdown and volatility analysis.

Now we will see the full backtesting report, code of ADF test and Code of backtesting which are APPENDIX I, II and III.

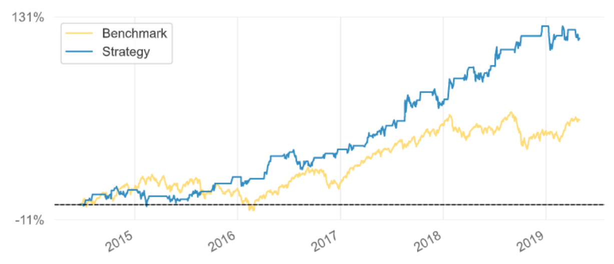
Pairs Trading
19 Jun, 2014 - 30 Apr, 2019

Benchmark is ^CRSLDX | Generated by QuantStats (v. 0.0.56)

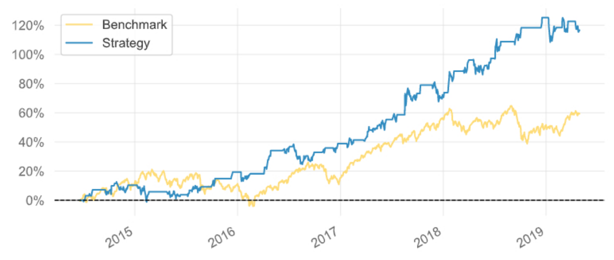
Cumulative Returns vs Benchmark



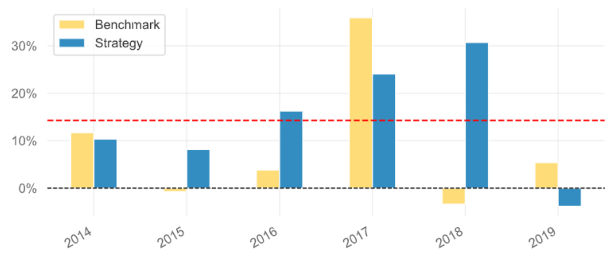
Cumulative Returns vs Benchmark (Log Scaled)



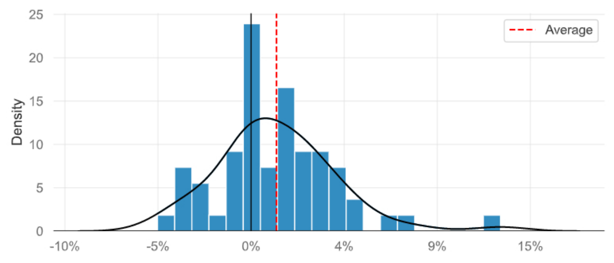
Cumulative Returns vs Benchmark (Volatility Matched)



EOY Returns vs Benchmark



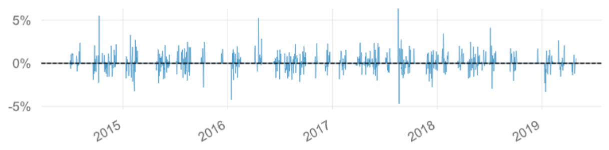
Distribution of Monthly Returns



Daily Returns

Key Performance Metrics

Metric	Strategy	Benchmark
Risk-Free Rate	0.0%	0.0%
Time in Market	47.0%	100.0%
Cumulative Return	116.36%	59.35%
CAGR﹪	17.19%	10.05%
Sharpe	1.26	0.79
Prob. Sharpe Ratio	99.51%	95.96%
Smart Sharpe	1.2	0.75
Sortino	2.07	1.08
Smart Sortino	1.98	1.03
Sortino/√2	1.46	0.76
Smart Sortino/√2	1.4	0.73
Omega	1.36	1.36
Max Drawdown	-12.23%	-20.82%
Longest DD Days	342	520
Volatility (ann.)	13.63%	13.65%
R^2	0.0	0.0
Information Ratio	0.02	0.02
Calmar	1.41	0.48
Skew	1.3	-0.73
Kurtosis	15.08	3.84
Expected Daily	0.06%	0.04%
Expected Monthly	1.32%	0.79%
Expected Yearly	13.73%	8.08%
Kelly Criterion	10.07%	9.74%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-1.34%	-1.37%
Expected Shortfall (cVaR)	-1.34%	-1.37%
Max Consecutive Wins	6	15
Max Consecutive Losses	5	9
Gain/Pain Ratio	0.36	0.14
Gain/Pain (1M)	2.55	0.75
Payoff Ratio	1.03	0.97
Profit Factor	1.36	1.14
Common Sense Ratio	1.64	1.06
CPC Index	0.76	0.62
Tail Ratio	1.2	0.93
Outlier Win Ratio	6.74	3.56
Outlier Loss Ratio	2.49	3.24
MTD	-2.71%	0.65%
3M	-1.06%	8.89%
6M	-0.77%	12.24%
YTD	-3.83%	5.39%
1Y	13.34%	2.34%
3Y (ann.)	17.85%	12.82%
5Y (ann.)	17.19%	10.05%
10Y (ann.)	17.19%	10.05%
All-time (ann.)	17.19%	10.05%
Best Day	9.05%	3.26%
Worst Day	-4.69%	-6.71%
Best Month	13.4%	10.67%
Worst Month	-5.02%	-8.77%
Best Year	30.7%	35.91%
Worst Year	-3.83%	-3.38%
Avg. Drawdown	-2.96%	-2.22%
Avg. Drawdown Days	33	30
Recovery Factor	9.51	2.85
Ulcer Index	0.03	0.07
Serenity Index	4.69	0.59
Avg. Up Month	3.7%	2.97%
Avg. Down Month	-2.92%	-3.61%
Win Days	54.37%	55.45%
Win Month	69.23%	61.02%
Win Quarter	61.9%	71.43%
Win Year	83.33%	66.67%



Beta	0.01	-
Alpha	0.17	-
Correlation	0.85%	-
Treynor Ratio	13686.62%	-

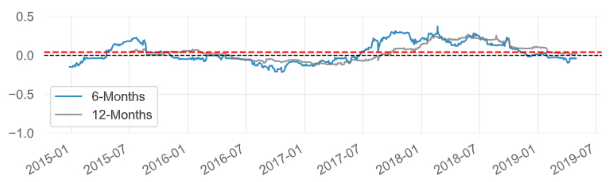
EOY Returns vs Benchmark

Year	Benchmark	Strategy	Multiplier	Won
2014	11.69%	10.35%	0.89	-
2015	-0.72%	8.16%	-11.30	+
2016	3.84%	16.25%	4.23	+
2017	35.91%	24.06%	0.67	-
2018	-3.38%	30.70%	-9.09	+
2019	5.39%	-3.83%	-0.71	-

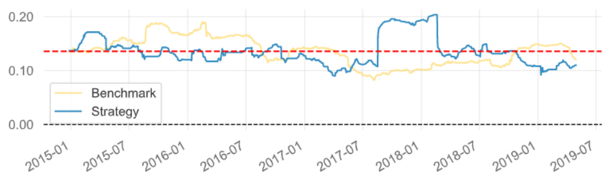
Worst 10 Drawdowns

Started	Recovered	Drawdown	Days
2014-10-29	2015-10-06	-12.23%	342
2016-07-19	2016-12-22	-9.96%	156
2019-01-10	2019-04-30	-7.46%	110
2017-12-04	2018-01-22	-7.42%	49
2016-01-14	2016-04-05	-5.30%	82
2018-04-19	2018-06-19	-5.28%	61
2017-08-29	2017-10-12	-4.89%	44
2017-08-21	2017-08-28	-4.69%	7
2017-05-31	2017-06-12	-4.19%	12
2018-07-11	2018-09-17	-4.05%	68

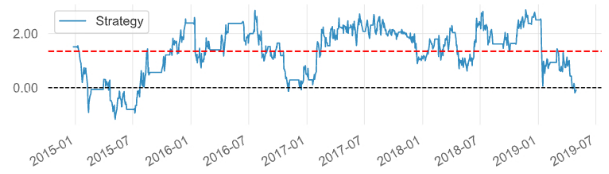
Rolling Beta to Benchmark



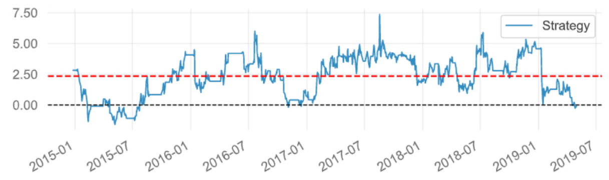
Rolling Volatility (6-Months)



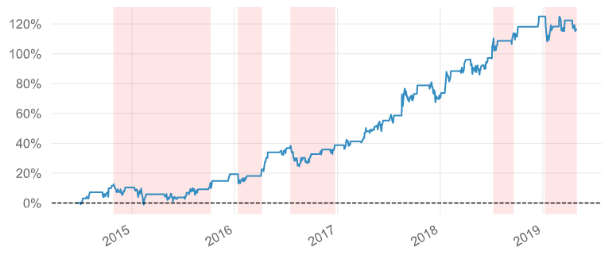
Rolling Sharpe (6-Months)



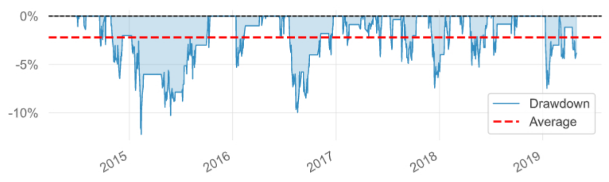
Rolling Sortino (6-Months)



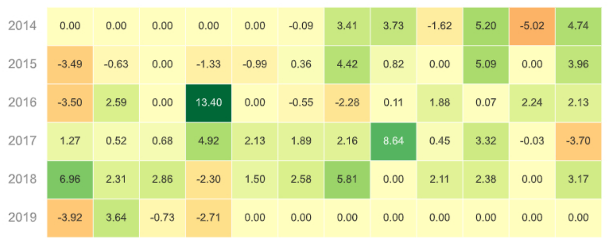
Worst 5 Drawdown Periods



Underwater Plot

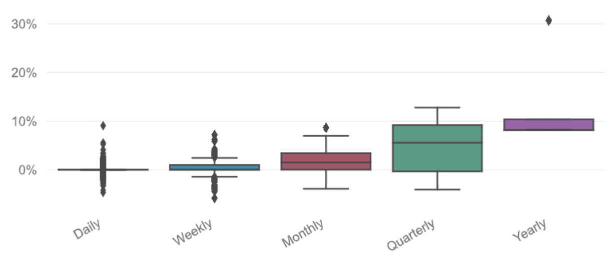


Monthly Returns (%)



2014	0.00	0.00	0.00	0.00	0.00	-0.09	3.41	3.73	-1.62	5.20	-5.02	4.74
2015	-3.49	-0.63	0.00	-1.33	-0.99	0.36	4.42	0.82	0.00	5.09	0.00	3.96
2016	-3.50	2.59	0.00	13.40	0.00	-0.55	-2.28	0.11	1.88	0.07	2.24	2.13
2017	1.27	0.52	0.68	4.92	2.13	1.89	2.16	8.64	0.45	3.32	-0.03	-3.70
2018	6.96	2.31	2.86	-2.30	1.50	2.58	5.81	0.00	2.11	2.38	0.00	3.17
2019	-3.92	3.64	-0.73	-2.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Return Quantiles



ADF Test

November 17, 2022

1 Notebook Instructions

You can run the notebook document sequentially (one cell a time) by pressing shift + enter. While a cell is running, a [*] will display on the left. When it has been run, a number will display indicating the order in which it was run in the notebook [8].

Enter edit mode by pressing `Enter` or using the mouse to click on a cell's editor area. Edit mode is indicated by a green cell border and a prompt showing in the editor area.

1.1 Read CSV file

[22]:
```
# To read csv file
import pandas as pd

# Read CSV into data frame prices_df
prices_df = pd.read_csv('AUDCAD.csv',index_col=0)
prices_df
```

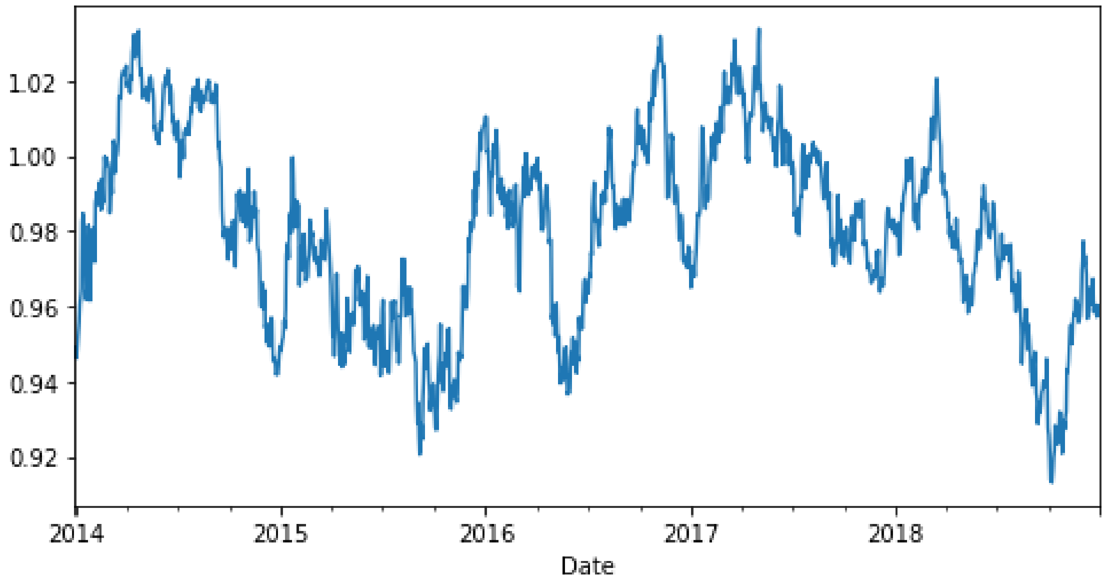
[22]:
```
            prices
Date
15-09-2010   0.963
16-09-2010   0.961
17-09-2010   0.968
20-09-2010   0.975
21-09-2010   0.979
…            …
11-09-2017   0.972
12-09-2017   0.977
13-09-2017   0.972
14-09-2017   0.973
15-09-2017   0.973

[1831 rows x 1 columns]
```

[23]:
```
import yfinance as yf
prices_df = pd.DataFrame(yf.download('AUDCAD=X',start='2014-01-01', end =␣
 ↪'2019-01-01')['Adj Close'])
prices_df.columns = ['prices']
```

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

1.2 Plot the price series

[24]:
```python
# Import matplotlib to plot the graph
import matplotlib.pyplot as plt
%matplotlib inline

# Plot the prices series
prices_df.prices.plot(figsize=(8,4))

# Show the plot
plt.show()
```



1.3 Perform ADF Test

[25]:
```python
# To perform ADF Test
from statsmodels.tsa.stattools import adfuller
```

[26]:
```python
# Compute ADF test statistics
adf = adfuller(prices_df.prices, maxlag = 1)
```

1.4 ADF Test statistic

[27]:
```python
adf[0]
```

[27]:
```
-3.3784669997678645
```

1.5 Critical values

```
[28]: adf[4]
```

```
[28]: {'1%': -3.435378572037035, '5%': -2.863760700696655, '10%': -2.56795231450063}
```

Data Science Mini Project

November 17, 2022

1 Pairs Trading - Data Science Project

[396]:
```
#importing libraries
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
pd.set_option('display.max_rows', 50)
%matplotlib inline
```

1.1 Fetching Daily data using yahoo-finance

[285]:
```
#downloading TCS data from yahoo finance API
df_1 = yf.download('TCS.NS',start='2014-05-01', end = '2019-05-01')['Adj Close']
#downloading INFY data from yahoo finance API
df_2 = yf.download('INFY.NS',start='2014-05-01',end = '2019-05-01')['Adj Close']
```

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

[359]:
```
#merging two dataframe using the datetime index
df = pd.merge(df_1,df_2,left_index=True, right_index=True)
```

[360]:
```
#renaming the columns
df.columns = ['TCS','INFY']
```

[361]:
```
#improrting statsmodel to import linear regession with the error as ordinary␣
 ↪least square method
from statsmodels.api import OLS
```

[362]:
```
#using linear regression to calculate the relation of price between TCS and INFY
model = OLS(df.TCS.iloc[:100], df.INFY.iloc[:100])
model = model.fit()
```

[363]:
```
#slope of linear regression which will be the coefficient
model.params[0]
```

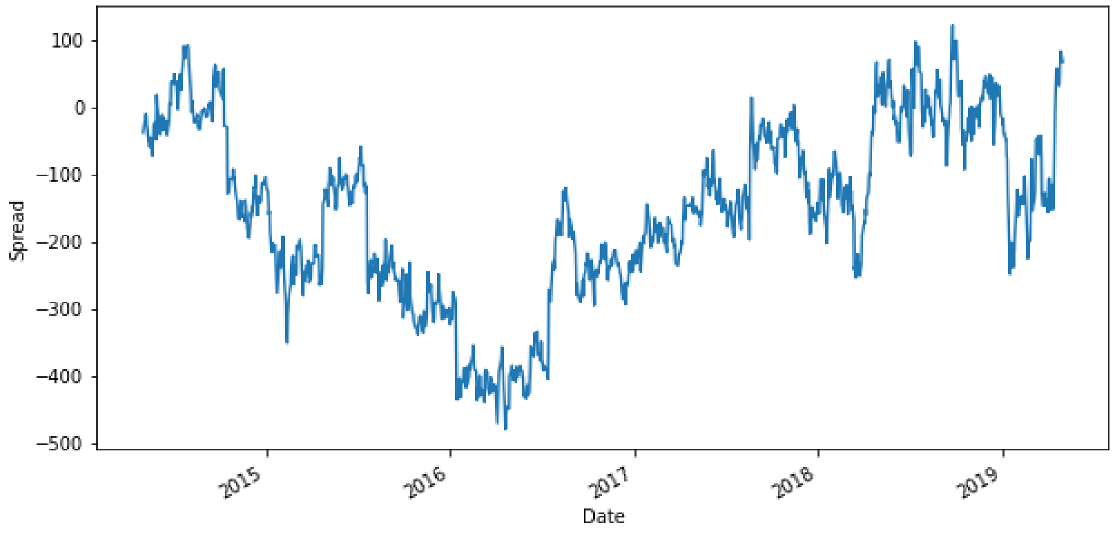
[363]: 2.997149491460207

[364]:
```
#finding the value of spread between TCS and INFY
df['spread'] = df.TCS - model.params[0] * df.INFY
```

[365]:
```
#Now we will check that this spread is a stationary time-series or not?
#We will use ADF-Fuller test to check this hypothesis
```

[366]:
```
#plotting the time-series
print(model.params)
# Plot the spread
df.spread.plot(figsize=(10,5))
plt.ylabel("Spread")
plt.show()
```

```
INFY    2.997149
dtype: float64
```



[367]:
```
#Importing stats model to calculate adffuller
from statsmodels.tsa.stattools import adfuller
```

[368]:
```
#Putting maximum lag as 1
adf = adfuller(df.spread, maxlag=1)
```

[369]:
```
#Value of ADF fuller test
adf[0]
```

[369]: -2.749474770831932

[370]:
```
adf[4]
```

[370]:
```
{'1%': -3.435690695421723,
 '5%': -2.863898416697677,
 '10%': -2.5680256555204184}
```

[371]:
```
#Here we can see that -2.86 < -2.7494 < -2.568
#So we are more than 90% confidence interval that our price is stationary
```

[372]:
```
#Calculating moving averages and Standard deviation
# Moving Average
df['moving_average'] = df.spread.rolling(30).mean()
# Moving Standard Deviation
df['moving_std_dev'] = df.spread.rolling(30).std()
```

[373]:
```
#calculating moving average of upper bound and lower bound
df['upper_band'] = df.moving_average + 2*df.moving_std_dev
df['lower_band'] = df.moving_average - 2*df.moving_std_dev
```

[374]:
```
#finding long entries and Long exits
df['long_entry'] = df.spread < df.lower_band
df['long_exit'] = df.spread >= df.moving_average
```

[375]:
```
#Segragating long entries and exits in dataframe
df['positions_long'] = np.nan
df.loc[df.long_entry,'positions_long']= 1
df.loc[df.long_exit,'positions_long']= 0

df.positions_long = df.positions_long.fillna(method='ffill')
```

[376]:
```
#Finding short entries
df['short_entry'] = df.spread > df.upper_band
df['short_exit'] = df.spread <= df.moving_average
```

[377]:
```
#Finding short entries and exits in datframe
df['positions_short'] = np.nan
df.loc[df.short_entry,'positions_short']= -1
df.loc[df.short_exit,'positions_short']= 0

df.positions_short = df.positions_short.fillna(method='ffill')
```

[378]:
```
#Net positions
df['positions'] = df.positions_long + df.positions_short
```

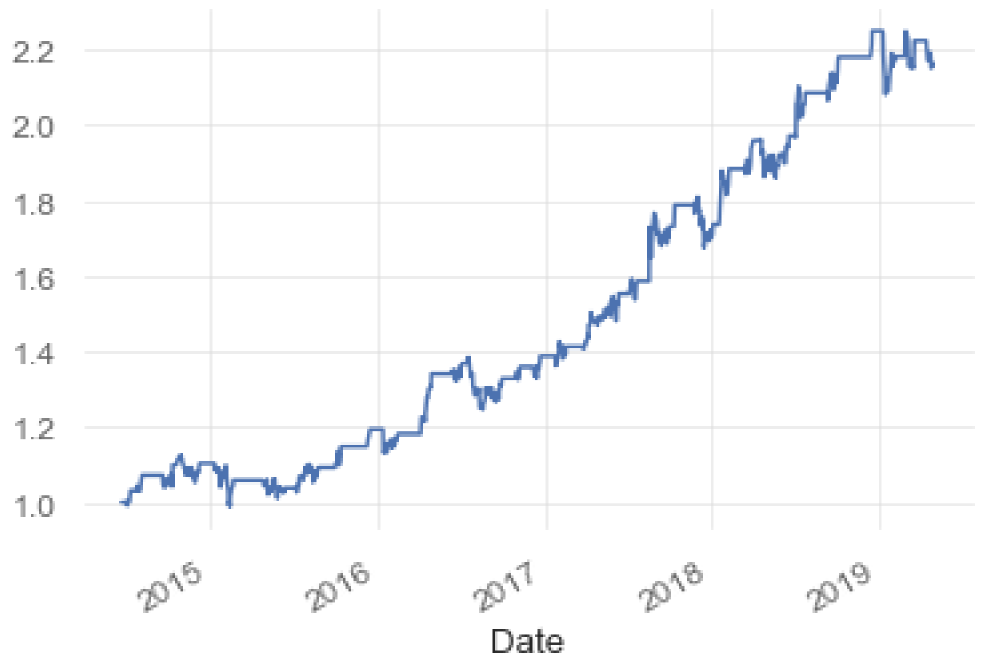
[407]:
```
#Finding pnl and spread analysis
#df['spread_difference']= df.spread - df.spread.shift(1)
```

```
df['spread_difference_pct_change'] = (df.spread - df.spread.shift(1))/(df.TCS +␣
↪df.INFY)
df['pnl_daily_pct'] = df.positions.shift(1) * df.spread_difference_pct_change
#df['cumpnl'] = df.pnl.cumsum()
df

df = df.dropna(subset=['pnl_daily_pct'])

df['cummulative'] = (1 + df['pnl_daily_pct']).cumprod()
```

```
C:\Users\kanos\AppData\Local\Temp\ipykernel_2492\1019591912.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['spread_difference_pct_change'] = (df.spread - df.spread.shift(1))/(df.TCS
+ df.INFY)
C:\Users\kanos\AppData\Local\Temp\ipykernel_2492\1019591912.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['pnl_daily_pct'] = df.positions.shift(1) * df.spread_difference_pct_change
C:\Users\kanos\AppData\Local\Temp\ipykernel_2492\1019591912.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['cummulative'] = (1 + df['pnl_daily_pct']).cumprod()
```

```
[408]: df['cummulative'].plot()
```

```
[408]: <AxesSubplot:xlabel='Date'>
```



[402]:
```
import quantstats as qs
df_returns = df[['pnl_daily_pct']]
df_returns
```

[402]:
```
            pnl_daily_pct
Date
2014-06-19      -0.000000
2014-06-20       0.000000
2014-06-23       0.000000
2014-06-24       0.000000
2014-06-25      -0.000000
…                     …
2019-04-23       0.009511
2019-04-24      -0.001507
2019-04-25      -0.011905
2019-04-26      -0.005421
2019-04-30       0.005714

[1194 rows x 1 columns]
```

[403]:
```
tickers = {
        'pnl_daily_pct':1
    }

portfolio = qs.utils.make_index(tickers,rebalance=None,returns=df_returns)
```

```
qs.reports.html(portfolio,benchmark='^CRSLDX',title='Pairs␣
 ↪Trading',download_filename='Pairs Trading.html')
```

[ ]:
```

```

[180]:
```
from statsmodels.tsa.stattools import adfuller
adf = adfuller(df.spread, maxlag=1)
```

[181]:
```
adf[0]
```

[181]:
```
-2.7494798121641586
```

[182]:
```
adf[4]
```

[182]:
```
{'1%': -3.435690695421723,
 '5%': -2.863898416697677,
 '10%': -2.5680256555204184}
```

[ ]:
```

```