Intraday Strategies Case Study

Tanmay Gupta

Introduction

Day Trading Strategies are designed to take advantage of short term price movement and this can be done using quantitative and technical techniques. Following is a list of the various different techniques, how they work and some downside:

ICT Methods (Inner Circle Trader):

How it works: ICT methods focus on market manipulation by large institutions. They use concepts like market structure, order blocks, liquidity pools, and imbalance areas. Traders look for price action signals in these areas to enter and exit trades.

Downsides: Requires deep understanding and experience. Not suitable for beginners due to complexity. Markets may not always behave as predicted by ICT concepts.

Gap Hunts:

How it works: Traders exploit price gaps that occur when the market opens significantly higher or lower than the previous close. They anticipate that the gap will fill (the price will move back to the previous close level).

Downsides: Not all gaps fill, and trying to trade every gap can lead to significant losses. Market conditions need to be right for gap trading to be effective.

Momentum/Trend Following:

How it works: Traders follow the prevailing trend, buying in an uptrend and selling in a downtrend. They use indicators like Moving Averages, Relative Strength Index (RSI), and MACD to identify trends and confirm momentum.

Downsides: Trends can reverse suddenly, leading to losses. Trend-following can also lead to late entries and exits, reducing profit margins. It can be challenging in sideways or choppy markets.

Pullbacks:

How it works: Traders look for temporary price retracements within a larger trend. They enter trades in the direction of the trend after the price pulls back to a support or resistance level.

Downsides: Determining the end of a pullback can be difficult. Pullbacks can turn into full trend reversals, leading to losses. Requires accurate timing and market analysis.

Mean Reversions:

How it works: Traders assume that prices will revert to their historical average or mean. They buy when prices are below the mean and sell when prices are above the mean, using indicators like Bollinger Bands and moving averages (Can use different rolling and estimated moving averages over different time frames).

Downsides: Prices may stay overbought or oversold for extended periods, leading to significant drawdowns. Mean reversion strategies can fail in trending markets where prices move far from the mean.

From an initial look on the list and from my own experience as a trader, I would be implementing gap hunts, momentum/trend following and pullbacks. ICT methods and information regarding them is rather

difficult to obtain and mean reversion are not particularly suitable for time series which are non-stationary and therefore might lead to unprecedented losses.

The holding period of the strategies employed is primarily in 5 min intervals and till EOD for some strategies. The latter limitation is due to the fact that yfinance offers 5 min interval data for only the last 60 days; however , to establish a continuous trend and profitability of the models over long periods, the Open and Close prices were used as entry and exit prices. More extensive data would only increase the profitability of these techniques as more sophisticated stop loss and take target prices could be set according to the model.

To add an extension, I would be using the Long Short Term Memory (LSTM/ ML) to train a model and predict the price the following day, and resultantly use that to take intraday long/short positions.

Developing, Coding, and Backtesting Strategies using Python

```
import numpy as np
import pandas as pd
import yfinance as yf
from math import sqrt
import matplotlib.pyplot as plt
from datetime import timedelta
from matplotlib import rcParams
from statsmodels.tsa.stattools import adfuller
rcParams['figure.figsize'] = 8,6
rcParams['font.family'] = 'serif'
import seaborn as sb
pd.set_option('min_rows',800)
sb.set()
```

✓ 0.0s Python

Importing different packages: Pandas, Numpy, Math, MatPlotLib, ADF Test, DateTime, YFinance

a. Strategy 1: Gap Hunting

```
## ticker = yf.download('SPY','2010-1-1') # The ticker can be changed based on what stock you want to implement the strategy on
ticker_day = yf.download(['SPY'],period='1mo',interval='1d')
vix = yf.download(['^VIX'],period='1mo',interval='1d')
# Important to note that this is expected annualized volatility for the next 30 days,
# to convert it into daily volatility I will divide it by sqrt 252
ticker_day['VIX'] = vix['Close']/sqrt(252)
ticker_day
```

✓ 0.5s Python

Firstly, I downloaded the data for the past one month for S&P 500 (SPY) along with the volatility index (VIX). VIX presents the expected annualized volatility for the next 30 days so I divided the amount by the square root of the active trading days to find the expected daily volatility. This value would then further be used to calculate the stop loss and target price for our trades.

```
ticker_day['Gap'] = ticker_day['Open']-ticker_day['Close'].shift(1)
ticker_day
```
✓ 0.0s Python

The gap was calculated by subtracting the previous day's closing price by the current day's opening price.

```
one_std = np.arange(ticker_day['Gap'].mean().round(4)-ticker_day['Gap'].std().round(4),ticker_day['Gap'].mean().round(4)+ticker_day['Gap']
two_std = np.arange(ticker_day['Gap'].mean().round(4)-2*ticker_day['Gap'].std().round(4),ticker_day['Gap'].mean().round(4)+2*ticker_day['G
```
✓ 0.0s Python

First, I will try to test the strategy with one_std and if that yields lower accuracy, we will try the two_std, just because the frequency of the former would be higher therefore more number of trades

```
ticker_day['Gap_Up'] = np.where(ticker_day['Gap']>one_std[-1],1,0)
ticker_day['Gap_Down'] = np.where(ticker_day['Gap']<one_std[0],1,0)
ticker_day['Gap_Up2'] = np.where(ticker_day['Gap']>two_std[-1],1,0)
ticker_day['Gap_Down2'] = np.where(ticker_day['Gap']<two_std[0],1,0)
ticker_day['Position'] = 0
ticker_day['PnL'] = 0
ticker_day
#ticker_day[ticker_day['Gap_Up']==1].index
```
✓ 0.0s Python

I used the standard deviation of all the gaps to identify abnormal gaps and when we could actually take advantage of a gap up or a gap down; two standard deviations suggest that the gap is larger than 90% of the data given and therefore more chances that it will lead to more profits. I then created a column where Gap Up and Gap Down columns were 1 when the gap was outside one standard deviation and Gap Up 2 and Gap Down 2 where 1 when the gap was outside two standard deviations.

After this, I iterated through my dataframe finding places where there was a Gap Up (Down) and further checked if there was a Gap Up 2 (Gap Down 2). After this, I took a short (long) position in the market and the number of shares depended on the nature of the opening (1 or 2). This value for stored in the Positions Column and the entry price was recorded. Using the .index feature of Pandas, I collected information regarding the date in regard, this in turn was inputted into a new dataframe which contained data regarding the 5 minute intervals. It is because a 1 day interval only reflects the open and close price whereas to make more profitable trades, it is essential to know the open and close for shorter time frames such as 5 minutes. I then proceeded on a strategy to deduce the exit. I thought it was best to use the volatility index for this purpose. For this I added the volatility to 100 and multiplied it to the entry price to find the upper limit and subtracted volatility from 100 and multiplied it to the entry price to find the lower limits. These limits acted as stoploss and target for the different scenarios. If price failed to reach any of these limits, the closing price of the day was set the exit price however, in a more extensive model, more sophisticated approach to deducing the exit pricing could be used. The code and the output for the same are presented on the next page.

```
print("Amount of Cumulative Positions Taken:", ticker_day['Position'].abs().sum())
print("Amount of Trades taken:" , ticker_day[ticker_day['Position']!=0]['Position'].count())
print("Average Profit and Loss per Trade: $" , (ticker_day['Cumulative_PnL'][-1]/ticker_day[ticker_day['Position']!=0]['Position'].count()
```
✓ 0.0s Python

```
Amount of Cumulative Positions Taken: 7
Amount of Trades taken: 6
Average Profit and Loss per Trade: $ 0.882
```

In 6 trades, out of which the majority were of 1 position, I was able to scalp 0.882$ profit per trade. I believe this is quite promising because considering a stable ETF like the SPY, utilizing the gaps we were able to generate profits healthily which could be amplified by trading more volume or increasing the number of positions taken per trade.

```
for i in range(0,len(ticker_day)):
    if ticker_day['Gap_Up'].iloc[i] ==1:
        if ticker_day['Gap_Up2'].iloc[i] ==1:
            entry_price = ticker_day['Open'].iloc[i]
            ticker_day['Position'].iloc[i] = -2
        else:
            entry_price = ticker_day['Open'].iloc[i]
            ticker_day['Position'].iloc[i] = -1
        Date = ticker_day[ticker_day['Open']==entry_price].index
        for j in Date:
            ticker = pd.DataFrame()
            ticker = yf.download('SPY',j,j+timedelta(days=0,hours=23),interval='5m')
            print(ticker)
            for k in range(0,len(ticker)):
                if ticker['Close'].iloc[k]>(100+ticker_day['VIX'].iloc[i]/2)*entry_price: # Trigger Stoploss if it moves above half of its expected volatility
                    exit_price = ticker['Close'].iloc[k]
                if ticker['Close'].iloc[k]<(100-ticker_day['VIX'].iloc[i]/2)*entry_price: # Target if it moves below half of its expected movement
                    exit_price = ticker['Close'].iloc[k]
                else:
                    exit_price = ticker_day['Close'].iloc[i]
        if ticker_day['Position'].iloc[i]==-2:
            ticker_day['PnL'].iloc[i] = 2*(entry_price - exit_price)
        else:
            ticker_day['PnL'].iloc[i] = entry_price - exit_price
    elif ticker_day['Gap_Down'].iloc[i] ==1:
        if ticker_day['Gap_Down2'].iloc[i] ==1:
            entry_price = ticker_day['Open'].iloc[i]
            ticker_day['Position'].iloc[i] = 2
        else:
            entry_price = ticker_day['Open'].iloc[i]
            ticker_day['Position'].iloc[i] = -1
        Date = ticker_day[ticker_day['Open']==entry_price].index
        for j in Date:
            print(j)
            ticker = pd.DataFrame()
            ticker = yf.download('SPY',j,j+timedelta(days=0,hours=23),interval='5m')
            for k in range(0,len(ticker)):
                if ticker['Close'].iloc[k]>(100+ticker_day['VIX'].iloc[i]/2)*entry_price: # Target if it moves below half of its expected movement
                    exit_price = ticker['Close'].iloc[k]
                if ticker['Close'].iloc[k]<(100-ticker_day['VIX'].iloc[i]/2)*entry_price: # Trigger Stoploss if it moves above half of its expected volatility
                    exit_price = ticker['Close'].iloc[k]
                else:
                    exit_price = ticker_day['Close'].iloc[i]
        if ticker_day['Position'].iloc[i]==-2:
            ticker_day['PnL'].iloc[i] = 2*(exit_price - entry_price)
        else:
            ticker_day['PnL'].iloc[i] = exit_price-entry_price
count = 0
for i in range(0,len(ticker_day)):
    if ticker_day['Position'].iloc[i] == 0:
        continue
    else:
        count +=1
ticker_day['Cumulative_PnL'] = ticker_day['PnL'].cumsum()
ticker_day

✓ 0.6s
```

Code for Gap Hunting

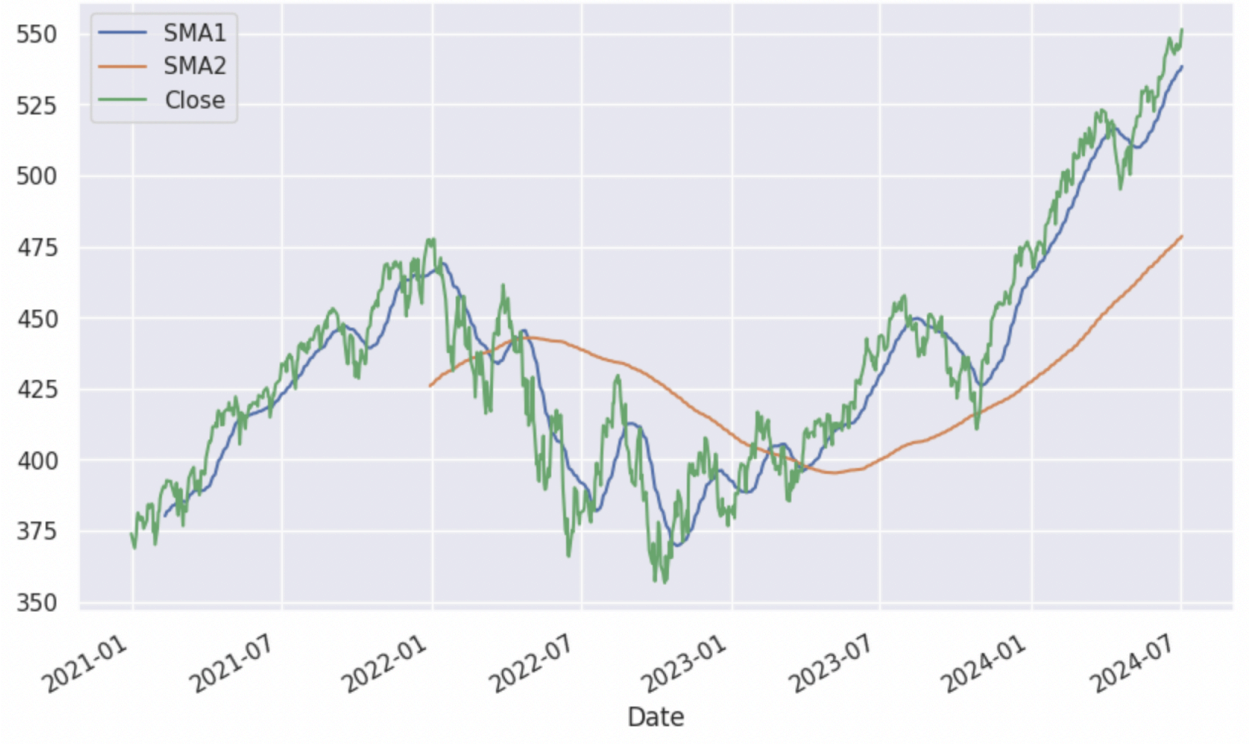
Date	Open	High	Low	Close	Adj Close	Volume	VIX	Gap	Gap_Up	Gap_Down	Gap_Up2	Gap_Down2	Position	PnL	Cumulative_PnL
2024-06-04	526.460022	529.150024	524.960022	528.390015	526.690857	34632700	0.829002	NaN	0	0	0	0	0	0.000000	0.000000
2024-06-05	530.770020	534.690002	528.729980	534.669983	532.950623	47610400	0.795615	2.380005	1	0	0	0	-1	-3.880005	-3.880005
2024-06-06	534.979980	535.419983	532.679993	534.659973	532.940674	30808500	0.792466	0.309998	0	0	0	0	0	0.000000	-3.880005
2024-06-07	533.659973	536.890015	532.539978	534.010010	532.292786	43224500	0.769788	-1.000000	0	0	0	0	0	0.000000	-3.880005
2024-06-10	533.179993	535.989990	532.570007	535.659973	533.937439	35729300	0.802545	-0.830017	0	0	0	0	0	0.000000	-3.880005
2024-06-11	534.070007	537.010010	532.049988	536.950012	535.223328	36383400	0.809474	-1.589966	0	1	0	0	-1	2.839966	-1.040039
2024-06-12	541.630005	544.119995	540.299988	541.359985	539.619141	63251300	0.758449	4.679993	1	0	1	0	-2	0.589966	-0.450073
2024-06-13	543.150024	543.330017	539.590027	542.450012	540.705627	44760900	0.752149	1.790039	1	0	0	0	-1	0.740051	0.289978
2024-06-14	540.880005	542.809998	539.849976	542.780029	541.034607	40089900	0.797505	-1.570007	0	0	0	0	0	0.000000	0.289978
2024-06-17	542.080017	548.530029	541.609985	547.099976	545.340637	55839500	0.803175	-0.700012	0	0	0	0	0	0.000000	0.289978
2024-06-18	547.159973	548.619995	546.729980	548.489990	546.726196	41376400	0.774827	0.059998	0	0	0	0	0	0.000000	0.289978
2024-06-20	549.440002	550.119995	545.179993	547.000000	545.240967	70328200	0.836561	0.950012	0	0	0	0	0	0.000000	0.289978
2024-06-21	544.400024	545.650024	543.020020	544.510010	544.510010	64513900	0.831522	-2.599976	0	1	0	0	-1	-0.270020	0.019958
2024-06-24	544.330017	546.950012	542.619995	542.739990	542.739990	45528700	0.839711	-0.179993	0	0	0	0	0	0.000000	0.019958
2024-06-25	543.989990	545.200012	542.440002	544.830017	544.830017	38273300	0.808844	1.250000	0	0	0	0	0	0.000000	0.019958
2024-06-26	543.690002	546.239990	543.030029	545.510010	545.510010	38550600	0.790576	-1.140015	0	0	0	0	0	0.000000	0.019958
2024-06-27	545.369995	546.960022	544.609985	546.369995	546.369995	35041500	0.771048	-0.140015	0	0	0	0	0	0.000000	0.019958
2024-06-28	547.159973	550.280029	542.950012	544.219971	544.219971	76144500	0.783646	0.789978	0	0	0	0	0	0.000000	0.019958
2024-07-01	545.630005	545.880005	542.520020	545.340027	545.340027	40297800	0.769788	1.410034	0	0	0	0	0	0.000000	0.019958
2024-07-02	543.700012	549.010010	543.650024	549.010010	549.010010	40434800	0.757819	-1.640015	0	1	0	0	-1	5.269958	5.289917
2024-07-03	548.690002	551.830017	548.650024	551.460022	551.460022	32789900	0.761598	-0.320007	0	0	0	0	0	0.000000	5.289917

Final DataFrame Table for Gap Hunting

b. Strategy 2: Momentum Following

```
SMA1 = 28
SMA2 = 252
SPY['SMA1'] = SPY['Close'].rolling(SMA1).mean()
SPY['SMA2'] = SPY['Close'].rolling(SMA2).mean()
SPY[['SMA1','SMA2','Close']].plot(figsize=(10,6));
```

Python



A Plot for the SPY with the Simple Moving Averages over the period of a month and a year

```
SPY_SMA2_Pct_change = SPY['SMA2'].pct_change()*100
SPY["SMA2 % Change"] = SPY_SMA2_Pct_change
SPY_Close_Pct_change = SPY['Close'].pct_change()*100
SPY["Close % Change"] = SPY_Close_Pct_change
SPY['Position'] = 0
SPY['PnL'] = 0
SPY['Cumulative PnL'] = SPY['PnL'].cumsum()
SPY.dropna()
```

✓ 0.0s Python

According to momentum trading, a stock tends to follow its long term trend and setbacks and small corrections present an ideal opportunity to implement day trading strategies. To do this quantitatively, I calculated the percentage change between consecutive closing prices of both the long term simple moving average and the closing price of the SPY.

```
SPY['Gap'] = SPY['Open']-SPY['Close'].shift(1)
one_std = np.arange(SPY['Gap'].mean().round(4)-SPY['Gap'].std().round(4),SPY['Gap'].mean().round(4)+SPY['Gap'].std().round(4),0.0001)
SPY['Gap_Up'] = np.where(SPY['Gap']>one_std[-1],1,0)
SPY['Gap_Down'] = np.where(SPY['Gap']<one_std[0],1,0)
SPY
```

✓ 0.0s Python

It is also necessary to involve the Gap Hunting process. It is because if the trend is upwards, and a stock fell the previous day, there are chances that the mean reversion happened during extended hours which is not reflected on the market data provided and the market opens gap up the following day. Taking a trade without this information might lead to losses. This would ensure that there are no gap ups or gap downs in the direction of our preferred trading direction and we do not enter a trade if there are.

```
SPY = SPY.dropna()
for i in range(1,len(SPY)):
    if SPY['SMA2 % Change'].iloc[i-1] > 0 and SPY['Close % Change'].iloc[i-1]<-0.3 and SPY['Gap_Up'].iloc[i] < 1:
        entry_price = SPY['Open'].iloc[i]
        exit_price = SPY['Close'].iloc[i]
        SPY['Position'].iloc[i] = 1
        SPY['PnL'].iloc[i] = exit_price-entry_price
    elif SPY['SMA2 % Change'].iloc[i-1] < 0 and SPY['Close % Change'].iloc[i-1]>0 and SPY['Gap_Down'].iloc[i] <1:
        entry_price = SPY['Open'].iloc[i]
        exit_price = SPY['Close'].iloc[i]
        SPY['Position'].iloc[i] = -1
        SPY['PnL'].iloc[i] = -exit_price+entry_price
    else:
        SPY['Position'].iloc[i] = 0
        SPY['PnL'].iloc[i] = 0
SPY['Cumulative PnL'] = SPY['PnL'].cumsum()
SPY
```

✓ 0.6s

This code filters out the data for where the change in SMA is positive, however the SPY had fallen and there was not a gap up opening the following day. It also checks for data where the SMA % Change is negative, and SPY increased the previous day, however there was not a gap down day following it.

Date	Open	Close	SMA1	SMA2	SMA2 % Change	Close % Change	Position	PnL	Cumulative PnL	Gap	Gap_Up	Gap_Down
2024-05-21	529.280029	531.359985	1.299988	512.527144	466.233056	0.096049	0.245253	0	0.000000	8.989777	-0.779968	0	0
2024-05-22	530.650024	529.830017	-1.529968	513.205002	466.673691	0.094510	-0.287934	0	0.000000	8.989777	-0.709961	0	0
2024-05-23	532.960022	525.960022	-3.869995	513.973216	467.117620	0.095126	-0.730422	0	0.000000	8.989777	3.130005	0	0
2024-05-24	527.849976	529.440002	3.479980	514.898574	467.587263	0.100541	0.661644	1	1.590027	10.579803	1.889954	0	0
2024-05-28	530.270020	529.809998	0.369995	515.943574	468.044247	0.097732	0.069884	0	0.000000	10.579803	0.830017	0	0
2024-05-29	525.679993	526.099976	-3.710022	516.892859	468.465199	0.089939	-0.700255	0	0.000000	10.579803	-4.130005	0	1
2024-05-30	524.520020	522.609985	-3.489990	517.873216	468.871667	0.086766	-0.663370	1	-1.910034	8.669769	-1.579956	0	0
2024-05-31	523.590027	527.369995	4.760010	518.860716	469.306271	0.092691	0.910815	1	3.779968	12.449738	0.980042	0	0
2024-06-03	529.020020	527.799988	0.429993	519.651787	469.726826	0.089612	0.081535	0	0.000000	12.449738	1.650024	0	0
2024-06-04	526.460022	528.390015	0.590027	520.472501	470.125516	0.084877	0.111790	0	0.000000	12.449738	-1.339966	0	0
2024-06-05	530.770020	534.669983	6.279968	521.586073	470.552381	0.090798	1.188510	0	0.000000	12.449738	2.380005	0	0
2024-06-06	534.979980	534.659973	-0.010010	522.528928	470.975516	0.089923	-0.001872	0	0.000000	12.449738	0.309998	0	0
2024-06-07	533.659973	534.010010	-0.649963	523.384286	471.401945	0.090542	-0.121566	0	0.000000	12.449738	-1.000000	0	0
2024-06-10	533.179993	535.659973	1.649963	524.587142	471.824683	0.089677	0.308976	0	0.000000	12.449738	-0.830017	0	0
2024-06-11	534.070007	536.950012	1.290039	525.894285	472.249485	0.090034	0.240832	0	0.000000	12.449738	-1.589966	0	0
2024-06-12	541.630005	541.359985	4.409973	527.191784	472.676310	0.090381	0.821301	0	0.000000	12.449738	4.679993	1	0
2024-06-13	543.150024	542.450012	1.090027	528.304642	473.096112	0.088814	0.201350	0	0.000000	12.449738	1.790039	0	0
2024-06-14	540.880005	542.780029	0.330017	529.240714	473.515159	0.088576	0.060838	0	0.000000	12.449738	-1.570007	0	0
2024-06-17	542.080017	547.099976	4.319946	530.310713	473.929842	0.087575	0.795893	0	0.000000	12.449738	-0.700012	0	0
2024-06-18	547.159973	548.489990	1.390015	531.428569	474.362501	0.091292	0.254070	0	0.000000	12.449738	0.059998	0	0
2024-06-20	549.440002	547.000000	-1.489990	532.386784	474.798294	0.091869	-0.271653	0	0.000000	12.449738	0.950012	0	0
2024-06-21	544.400024	544.510010	-2.489990	533.232141	475.233096	0.091576	-0.455208	0	0.000000	12.449738	-2.599976	0	0
2024-06-24	544.330017	542.739990	-1.770020	534.011784	475.654643	0.088703	-0.325066	1	-1.590027	10.859711	-0.179993	0	0
2024-06-25	543.989990	544.830017	2.090027	534.780714	476.097580	0.093121	0.385088	1	0.840027	11.699738	1.250000	0	0
2024-06-26	543.690002	545.510010	0.679993	535.342499	476.550239	0.095077	0.124808	0	0.000000	11.699738	-1.140015	0	0
2024-06-27	545.369995	546.369995	0.859985	535.973927	476.987540	0.091764	0.157648	0	0.000000	11.699738	-0.140015	0	0
2024-06-28	547.159973	544.219971	-2.150024	536.501426	477.415437	0.089708	-0.393511	0	0.000000	11.699738	0.789978	0	0
2024-07-01	545.630005	545.340027	1.120056	537.047141	477.840953	0.089129	0.205809	1	-0.289978	11.409760	1.410034	0	0
2024-07-02	543.700012	549.010010	3.669983	537.677499	478.260516	0.087804	0.672971	0	0.000000	11.409760	-1.640015	0	0
2024-07-03	548.690002	551.460022	2.450012	538.449999	478.687778	0.089337	0.446260	0	0.000000	11.409760	-0.320007	0	0

The tail of the DataFrame

```
print("Amount of Cumulative Positions Taken:", SPY['Position'].abs().sum())
print("Amount of Trades taken:" , SPY[SPY['Position']!=0]['Position'].count())
print("Average Profit and Loss per Trade: $" , (SPY['Cumulative PnL'][-1]/SPY[SPY['Position']!=0]['Position'].count()).round(3))
```

✓ 0.0s Python

```
Amount of Cumulative Positions Taken: 256
Amount of Trades taken: 256
Average Profit and Loss per Trade: $ 0.045
```

Over a period from 2022, the model took 256 trades, each of a position of 1 and yielded $0.045 in profit per trade. The low profit suggests that there might be something lacking in my model but it is also a signal that in the long run it is a profit yielding strategy if we ignore the transaction fees and slippage. To better the same strategy, I conducted some literature review and found that it is important that we use a test known as ADF test to find out if a series has a unit root, and then we could modify our data to make it stationary so that the historical average stays constant. Implementing the strategy then could yield better results.

c. Strategy 2: Momentum Following (Z-Score)

```
result = adfuller(SPY['Close'].dropna())
print('ADF Statistic:',result[0])
print('p-value:',result[1])
print("Critical Values:",result[4])
```

✓ 0.0s Python

```
ADF Statistic: -0.6504706633799245
p-value: 0.8591651789513037
Critical Values: {'1%': -3.4361864296062166, '5%': -2.864117116658563, '10%': -2.568142129417371 4}
```

The ADF statistic of -0.6504706633799245 is not more negative than any of the critical values. The p-value of 0.8591651789513037 is much higher than the typical significance level of 0.05. Based on these results, you fail to reject the null hypothesis that the time series has a unit root. This implies that the time series is non-stationary, meaning its statistical properties such as mean, variance, and autocorrelation are not constant over time.

Therefore initially without doing this test, I got a Cumulative PnL of $8.21, i would try to make the series stationary and then see if it has any impact on the total PnL

```
SPY['Close_Diff'] = SPY['Close'].diff()
result_diff = adfuller(SPY['Close_Diff'].dropna())
print('ADF Statistic:',result_diff[0])
print('p-value:',result_diff[1])
print("Critical Values:",result_diff[4])
```

✓ 0.0s Python

```
ADF Statistic: -10.210486248323837
p-value: 5.655365193297729e-18
Critical Values: {'1%': -3.4361864296062166, '5%': -2.864117116658563, '10%': -2.568142129417371 4}
```

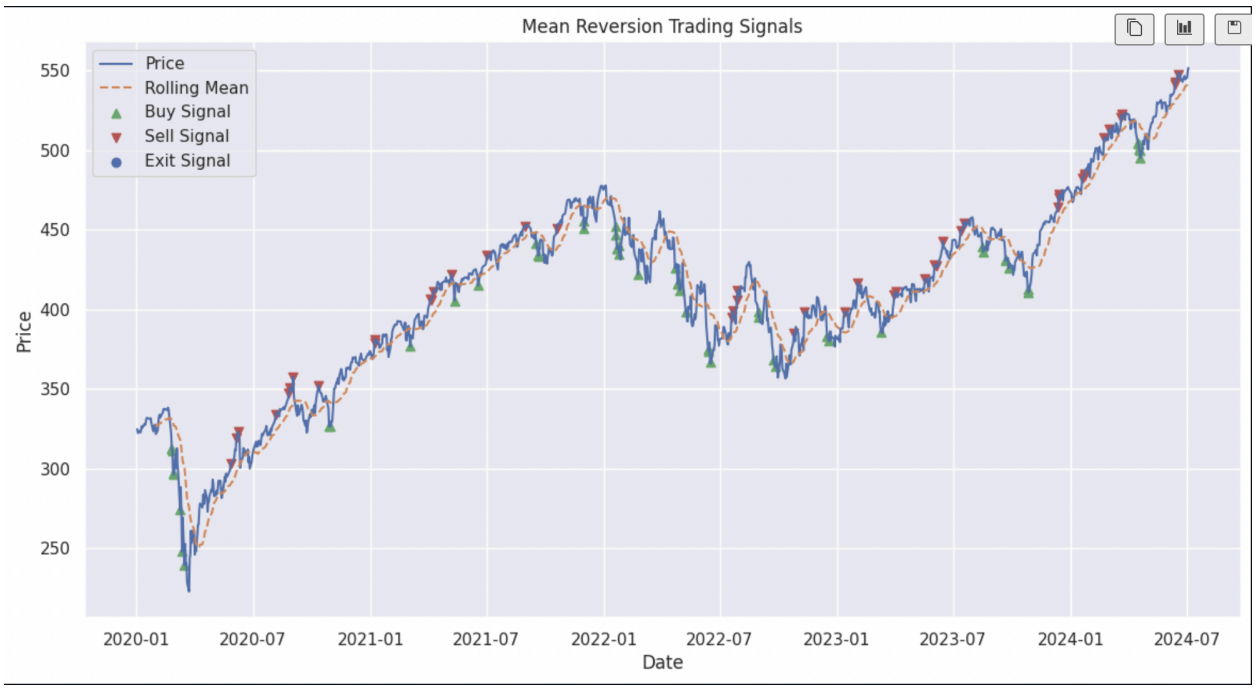
Conducting the ADF test using adfuller, I found out that the Closing price of the SPY was in fact non-stationary. I then found out the closing difference and used that to calculate the ADF Statistic, p-value, critical values for the new series and this indeed came out to be stationary.

```
SPY['Z-Score'] = (SPY['Close'] - SPY['Rolling Mean']) / SPY['Rolling Std']

# Define trading signals
entry_threshold = 2
exit_threshold = 0

SPY['Buy Signal'] = SPY['Z-Score'] < -entry_threshold
SPY['Sell Signal'] = SPY['Z-Score'] > entry_threshold
SPY['Exit Signal'] = SPY['Z-Score'].abs() < exit_threshold

# Plot the time series and signals
plt.figure(figsize=(14, 7))
plt.plot(SPY.index, SPY['Close'], label='Price')
plt.plot(SPY.index, SPY['Rolling Mean'], label='Rolling Mean', linestyle='--')
plt.scatter(SPY.index[SPY['Buy Signal']], SPY['Close'][SPY['Buy Signal']], label='Buy Signal', marker='^', color='g')
plt.scatter(SPY.index[SPY['Sell Signal']], SPY['Close'][SPY['Sell Signal']], label='Sell Signal', marker='v', color='r')
plt.scatter(SPY.index[SPY['Exit Signal']], SPY['Close'][SPY['Exit Signal']], label='Exit Signal', marker='o', color='b')
plt.legend()
plt.title('Mean Reversion Trading Signals')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```

✓ 0.4s Python

In mean reversion trading, the Z-score helps to determine how far the current price is from its historical average (mean). When the Z-score is high (positive), it indicates that the price is significantly above the mean, suggesting that it may revert back down towards the mean. Conversely, when the Z-score is low

(negative), it indicates that the price is significantly below the mean, suggesting that it may revert back up towards the mean. I used the entry threshold as 2 and the exit threshold as 0. Plotting the graph, I derived this. To make this meaningful I had to quantify the data.



We need to quantitatively calculate the PnL for this method

+ Code + Markdown

```
TradeBook = pd.DataFrame(index=SPY.index)
TradeBook['Position'] = 0
TradeBook['Price Bought'] = 0.0
TradeBook['Price Sold'] = 0.0
for i in range(1,len(SPY)):
    if SPY['Buy Signal'].iloc[i-1] == True:
        TradeBook['Position'].iloc[i-1] += 1
        TradeBook['Price Bought'].iloc[i-1] += SPY['Close'].iloc[i-1]
    if SPY['Sell Signal'].iloc[i-1] == True:
        TradeBook['Position'].iloc[i-1] -=  1
        TradeBook['Price Sold'].iloc[i-1] -= SPY["Close"].iloc[i-1]
TradeBook['Total Position Left'] = TradeBook['Position'].cumsum()
TradeBook['Total Bought'] = TradeBook['Price Bought'].cumsum()
TradeBook['Total Sold'] = TradeBook['Price Sold'].cumsum()
print(f"Positions Left: {TradeBook['Total Position Left'][-1]}, with PnL {TradeBook['Total Bought'][-1]+TradeBook['Total Sold'][-1]}")
```

✓ 0.0s Python

```
Positions Left: 4, with PnL 393.94007873535156
```

This strategy is more like swing trading where the holding periods are typically longer than a day therefore they might not fall directly under the purview of day trading strategies but the z score threshold might be decreased to fit intraday levels as well. Note there were some positions left uncatered to because there was not an appropriate exit signal to exit the trade.

```
print(f"Positions Left: {TradeBook['Total Position Left'][-1]}, with PnL {TradeBook['Total Bought'][-1]+TradeBook['Total Sold'][-1]}")
print(f"Total number of trades taken: " ,TradeBook[TradeBook['Position']!=0]['Position'].count())
print(f"Profitability per trade:",((TradeBook['Total Bought'][-1]+TradeBook['Total Sold'][-1])/TradeBook[TradeBook['Position']!=0]['Positi
```

✓ 0.1s Python

```
Positions Left: 4, with PnL 393.94007873535156
Total number of trades taken:  96
Profitability per trade: 4.104
```

There were 96 trades taken over the span of 5 years and the profitability was an impressive $4.104 dollars. This is with 4 positions left which puts in a really good positions and speaks up on the effectiveness of this strategy.

d. Strategy 3: Long Short Term Memory Approach

Upon reading various articles on Medium and QuantInsider, I learned about this ML RNN which is very effective with sequential datasets, exactly what we are using.

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import math
```

Python

```
data_spy = yf.download('SPY','2010-1-1')
data_spy
```

Python

To start with the LSTM Model, the MinMaxScaler was imported to scale the data such that it can be fit into the model. Keras Sequential Model: The Sequential model is a linear stack of layers. LSTM Layer: The LSTM layer is used to create an LSTM network, which is suitable for sequence prediction problems. It can maintain long-term dependencies through its cell state and gates. Dense Layer: The Dense layer is a regular fully connected layer, which is used for outputting predictions or creating complex feature representations. After fetching the data for SPY, I plotted the data as show below.

I then trained my model on around 80% of the given data and used the 20% of the remaining to test the predictions of the model. If the model accurately predicts the data, we can then use it to give us insights into the market movement on those days and use that to take advantage of day trading.

```python
data = data_spy.filter(['Close'])
#Convert the data frame to numpy array
dataset = data.values
#Get the number of rows to train the model on
training_data_len = math.ceil(len(dataset)*0.8) # Training about 80% of the data
training_data_len
```

Python

```
2919
```

```python
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
scaled_data
```

Python

```python
#Create the training data set, create the scaled training data set
train_data = scaled_data[0:training_data_len,:]
#Split the data into x_train and y_train data sets
x_train = []
y_train = []
for i in range(60,len(train_data)):
    x_train.append(train_data[i-60:i,0]) # will contain 60 values
    y_train.append(train_data[i,0]) # will contain 61st value
    if i <= 61:
        print(x_train)
        print(y_train)
        print()
```

Python

To do this, first I set the training data set length to 0.8 of the entire data set and then I transformed the data using the MinMaxScaler to be in values between 0 and 1. The scaled data was then trained from 0 to the training length for all the columns.

```python
#Convert the x_train and y_train to numpy arrays
x_train,y_train = np.array(x_train),np.array(y_train)
```

Python

```python
#Reshape the data, LSTM expects the data to be 3dimensional: number of samples, number of time steps, number of features
x_train = np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))
x_train.shape
```

Python

```
(2859, 60, 1)
```

```python
#Build the LSTM Model
model = Sequential()
model.add(LSTM(50,return_sequences = True, input_shape= (x_train.shape[1],1)))
model.add(LSTM(50,return_sequences = False))
model.add(Dense(25))
model.add(Dense(1))
```

Python

The first 60 values were added to the x train data and the 61st value was added to the y train data. This were then converted to numpy arrays so that the model can handle the data and the x was reshaped to be 3 dimensional because LSTM models utilize 3 dimensions.

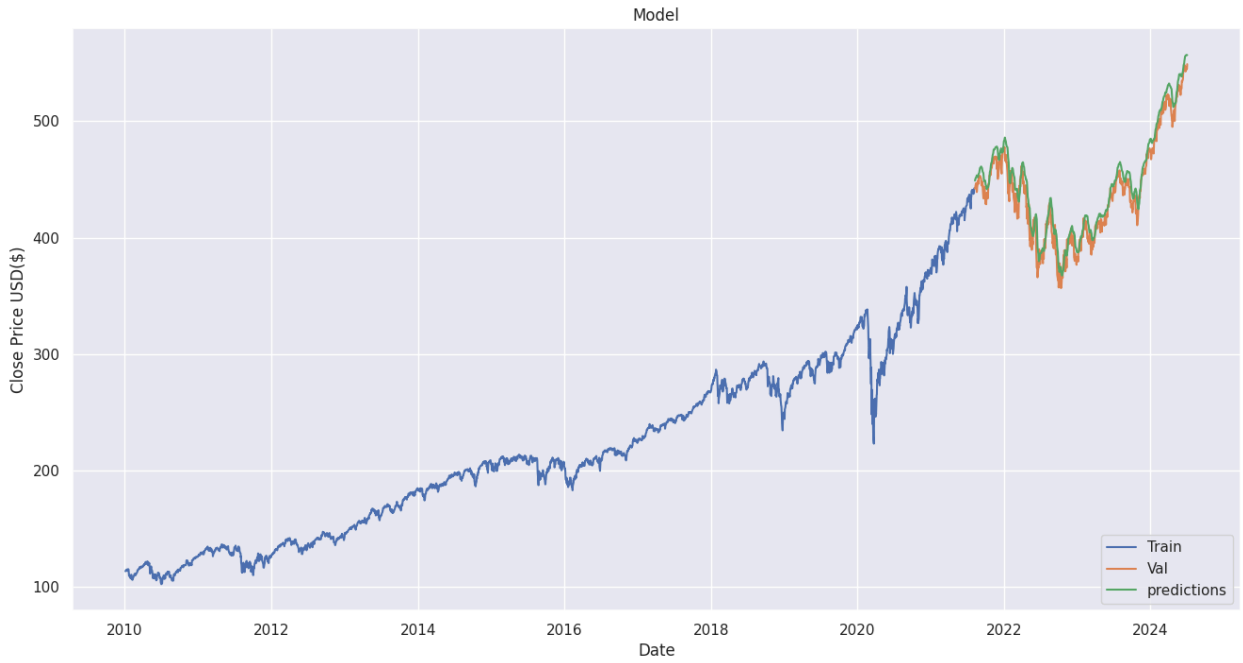
```python
#Compile the Model
model.compile(optimizer='adam',loss='mean_squared_error')
```
Python

```python
#Train the model
model.fit(x_train,y_train,batch_size=1,epochs=1)
```
Python

```python
#Create the testing data set
#Create a new array containing new scaled values from index end of training set to full testing data set
test_data = scaled_data[training_data_len-60:,:]
#Create the data sets x_test, y_test
x_test = []
y_test = dataset[training_data_len:,:]
for i in range(60,len(test_data)):
    x_test.append(test_data[i-60:i,0])
```
Python

```python
#Get the model's predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```
Python

```python
valid = data[training_data_len:]
valid['predictions'] = predictions
```



Upon plotting the data, we can see that the predicted values are indeed very close to the actual values with a Root Mean Square Error of about 0.10. This suggests that this model could be effectively used to predict the price movements and trade. I then proceed to code further to quantify this.

Using a similar approach I had used in the previous methods, I loop through the dataframe finding places where the value of the predictions had increased. If the value increased by more than 1 in dollar value I took a position of 2 otherwise a change of 0.2 was followed by a position of 1. The same was done if the predictions indicated a fall in price. The exit price was set as the closing price. This could be updated in

future models to better improve the accuracy and profitability of the model.

```python
count = 0
for i in range(1,len(valid)):
    if valid["predictions"].iloc[i]-valid['predictions'].iloc[i-1]>0.2:
        if valid["predictions"].iloc[i]-valid['predictions'].iloc[i-1]>1:
            entry_price = valid['Open'].iloc[i]
            exit_price = valid['Close'].iloc[i]
            valid['Position'].iloc[i] = 2
            valid['PnL'].iloc[i] = 2*(exit_price-entry_price)
            count+=1
        else:
            entry_price = valid['Open'].iloc[i]
            exit_price = valid['Close'].iloc[i]
            valid['Position'].iloc[i] = 1
            valid['PnL'].iloc[i] = exit_price-entry_price
            count+=1
    elif valid["predictions"].iloc[i]-valid['predictions'].iloc[i-1]<-0.2:
        if valid["predictions"].iloc[i]-valid['predictions'].iloc[i-1]<-1:
            entry_price = valid['Open'].iloc[i]
            exit_price = valid['Close'].iloc[i]
            valid['Position'].iloc[i] = -2
            valid['PnL'].iloc[i] = 2*(-exit_price+entry_price)
            count+=1
        else:
            entry_price = valid['Open'].iloc[i]
            exit_price = valid['Close'].iloc[i]
            valid['Position'].iloc[i] = -1
            valid['PnL'].iloc[i] = -exit_price+entry_price
            count+=1
    else:
        valid['Position'].iloc[i] = 0
        valid['PnL'].iloc[i] = 0
        continue
valid['Cumulative_PnL'] = valid['PnL'].cumsum()
valid
```

Python

Following is the tale of the data when I traded based on this model.

Date	Open	Close	predictions	Position	PnL	Cumulative_PnL
2024-05-30	524.520020	522.609985	515.864685	-1	1.910034	47.999329
2024-05-31	523.590027	527.369995	515.101501	-1	-3.779968	44.219360
2024-06-03	529.020020	527.799988	514.704285	-1	1.220032	45.439392
2024-06-04	526.460022	528.390015	514.533264	0	0.000000	45.439392
2024-06-05	530.770020	534.669983	514.567566	0	0.000000	45.439392
2024-06-06	534.979980	534.659973	515.394348	1	-0.320007	45.119385
2024-06-07	533.659973	534.010010	516.474854	2	0.700073	45.819458
2024-06-10	533.179993	535.659973	517.506165	2	4.959961	50.779419
2024-06-11	534.070007	536.950012	518.589294	2	5.760010	56.539429
2024-06-12	541.630005	541.359985	519.701416	2	-0.540039	55.999390
2024-06-13	543.150024	542.450012	521.176880	2	-1.400024	54.599365
2024-06-14	540.880005	542.780029	522.747009	2	3.800049	58.399414
2024-06-17	542.080017	547.099976	524.219421	2	10.039917	68.439331
2024-06-18	547.159973	548.489990	525.950256	2	2.660034	71.099365
2024-06-20	549.440002	547.000000	527.724304	2	-4.880005	66.219360
2024-06-21	544.400024	544.510010	529.145630	2	0.219971	66.439331
2024-06-24	544.330017	542.739990	529.969421	1	-1.590027	64.849304
2024-06-25	543.989990	544.830017	530.223267	1	0.840027	65.689331
2024-06-26	543.690002	545.510010	530.400146	0	0.000000	65.689331
2024-06-27	545.369995	546.369995	530.564392	0	0.000000	65.689331
2024-06-28	547.159973	544.219971	530.793640	1	-2.940002	62.749329
2024-07-01	545.630005	545.340027	530.787170	0	0.000000	62.749329
2024-07-02	543.700012	549.010010	530.812256	0	0.000000	62.749329
2024-07-03	548.690002	551.460022	531.242432	1	2.770020	65.519348

```
print("Amount of Cumulative Positions Taken:", valid['Position'].abs().sum())
print("Amount of Trades taken:" , valid[valid['Position']!=0]['Position'].count())
print("Average Profit and Loss per Trade: $" , (valid['Cumulative_PnL'][-1]/valid[valid['Position']!=0]['Position'].count()).round(3))
```

✓ 0.0s Python

```
Amount of Cumulative Positions Taken: 1008
Amount of Trades taken: 653
Average Profit and Loss per Trade: $ 0.1
```

Out of 653 trades taken of multiple positions, this model generated approximately $0.1 per trade. The whole code for all of the models can be found in my github repository here for more reference on each code!

Proposals for improvement in each strategy

Gap Hunting

1. Volume Considerations: Ensuring there is enough trading volume to avoid slippage
2. Trend Analysis: Determine the market trend before taking a trade and considering SMA, EMA and the Relative Strength Index
3. Try Gap Hunting across different stocks to see where this strategy is the most effective
4. Use optimization techniques such as Walk-Forward Optimization and Stochastic Optimization
5. Position Sizing: Better develop position sizing algorithms to manage potential losses and exploit further gains
6. Market Sentiment: Use market sentiment and indicators such as Net Social Sentiment and Net Institutional Flow to deduce the nature of the Gaps

Mean Reversion

Although I had implemented the gap hunting strategy into the mean reversion (SMA) trading strategy, the suggestions for the previous strategy follow along on this too along with some additional suggestions

1. Technical Indicators: Include indicators like RSI, Bollinger Bands, Moving Averages etc
2. Conduct rigorous Volume Analysis: This includes volume based indicators such as VWAP
3. Fundamental Analysis: Metrics like earnings, P/E Ratios etc.
4. Techniques like Kelly Criterion (used to determine the optimal size of an investment, based on the probability and expected size of a win or loss) or volatility based sizing

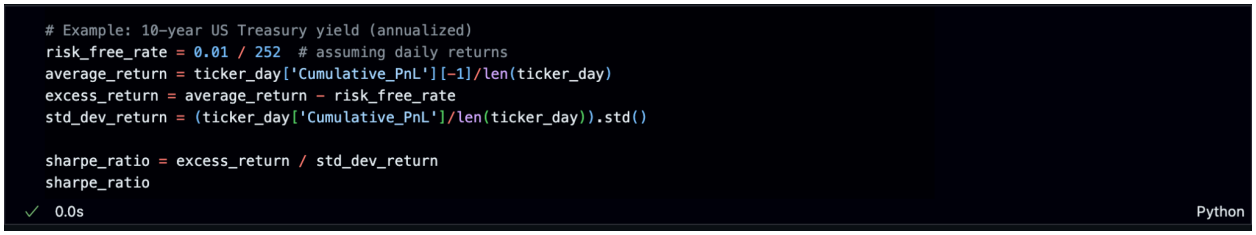
. **LSTM Model**

1. Include the various indicators mentioned above
2. Use lagged values of features to help the model understand temporal dependencies
3. Experiment with different model architectures to find the best one.
4. Add Dropout layers to prevent overfitting
5. Use techniques like Grid Search or Random Search to find the optimal hyperparameters.
6. **Early Stopping**: Use early stopping to prevent overfitting by monitoring validation loss.

Risk Adjusted Returns

It is important to realise what risks we are bearing and are accepting with each investment. To do so I am going to use the Sharpe Ratio which is a widely used metric in finance helping investors and traders evaluate the risk-adjusted return. To do this I will measure the excess return (return above the risk-free rate) per unit of volatility (standard deviation of the returns). It would quantify the amount of additional return received for the extra volatility endured.

To understand the data presented in the tabular format below, a higher sharpe ratio indicates that the asset has provided better risk-adjusted ratios and more return for the same amount of risk or less risk for the same level of return. The Hit Ratio on the other hand would tell us the number of profitable trades taken as a fraction of all trades taken.



```python
# Example: 10-year US Treasury yield (annualized)
risk_free_rate = 0.01 / 252  # assuming daily returns
average_return = ticker_day['Cumulative_PnL'][-1]/len(ticker_day)
excess_return = average_return - risk_free_rate
std_dev_return = (ticker_day['Cumulative_PnL']/len(ticker_day)).std()

sharpe_ratio = excess_return / std_dev_return
sharpe_ratio
```

✓ 0.0s Python

```python
Hit = (ticker_day['PnL'] > 0).sum()/((ticker_day['Position']!=0).sum())
Hit
```

✓ 0.0s Python

	Gap Hunting	Momentum Following	Momentum Following, ADF and Z-Score	LSTM
Profit per Trade	$0.882	$0.045	$4.104	$0.1
Sharpe Ratio	3.100	1.030	0.178	2.310
Hit Ratio	0.75	0.527	NA	0.497

Source:

Python For Finance by Yves Hilpisch
Investopedia - Technical Analysis
Medium - Understanding the Basics of Technical Indicators
Towards Data Science - Sentiment Analysis in Trading
Scikit-Optimize Documentation - Bayesian Optimization
Investopedia - Position Sizing Methods
QuantStart - The Importance of Transaction Costs in Trading
Quantitative Trading - Strategy Diversification
Towards Data Science - Machine Learning in Finance
Towards Data Science - Reinforcement Learning in Trading
Investopedia's guide on the Sharpe Ratio
Github Link: https://github.com/lemontang3/Python_Finance/tree/main/Intraday_Strategies