



HARVARD UNIVERSITY

Vol-arb trading in the Options Market

Final Assignment in assessing the arbitrage between a basket of options on the S&P 500 presented to Harvard University for Applied Quantitative Finance and Machine Learning CSCI S-278

by

Aniket Manoj Shroff

Under the supervision of

Professor: MarcAntonio Awada, Matthew Nemesure
Teaching Assistant: Conor Mackey

HARVARD UNIVERSITY AUGUST 2024

Table of Contents

Introduction	2
Chapter 1	4
Data Description and Methodology	4
Chapter 2	9
Analysis of Code and Data	9
Chapter 3	12
Model Building – Multi Linear Regression	12
Multi Linear Regression - NVDA 6-Month Data	12
Multi Linear Regression - NVDA 6 Years Data	15
Multi Linear Regression - TSLA 6-Month Data	17
Multi Linear Regression - TSLA 6 Years Data	19
Model Building – Support Vector Machine	21
Support Vector Machine – NVDA	21
Support Vector Machine - TSLA	22
Chapter 4	23
Purging Data	23
Expanding Window	23
Sliding Window	23
Chapter 5	25
Buy and Sell Signals	25
NVDA Stock Vol-Arb Trading	25
TSLA Stock Vol-Arb Trading	26
Chapter 6	27
Scope Limitation	27
Conclusion	30
Appendix	32
References	33
CODE	33

Introduction

The options market is a fascinating side of finance that gained momentum in the last 50 years. The famous Black and Scholes formula allowed investors to take exposure to the market at a fraction of the price. Their invention gave rise to an entirely new segment of trading that allowed users to trade rights to own products rather than purchasing the product itself. Options seem to be very confusing to the average trader. The following section will give a quick understanding of Options followed by a volatility arbitrage strategy to trade them efficiently.

Options are derivatives that give an investor the right to buy or sell a product; hence, they are essentially contracts or pieces of paper that define a right to own or sell. There are two types of options namely call and put options. Call options are contracts that give an investor the right to buy a stock while put options give the investor a right to sell the stock. Every contract has two parties namely the writer of the option and the buyer of the option. Hence, a call writer gives the buyer the right to buy the underlying stock, and put writer gives the buyer the right to sell the underlying stock.

Black and Scholes utilized the volatility of the underlying stock to price the option. However, that volatility is an estimation of historic standard deviation. To get the current market sentiment, one can back out the implied volatility of the underlying stock by using the current price at which the call option transacted. The implied volatility reflects the current market sentiment of the price movement of the underlying stock. The implied volatility is one of the major factors in determining the price movement of the option. According to Doris Dobi, “the market price of an option and its implied volatility are interchangeable; many quote the option price in terms of its implied volatility” (Doris 5). Therefore, predicting the implied volatility is as good as predicting the price of an option. Below is a formula for pricing the call option:

$$C = N(d_1)S_t - N(d_2)Ke^{-rt}$$

$$\text{where } d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}$$

C = call option price

N = CDF of the normal distribution

S_t = spot price of an asset

K = strike price

r = risk-free interest rate

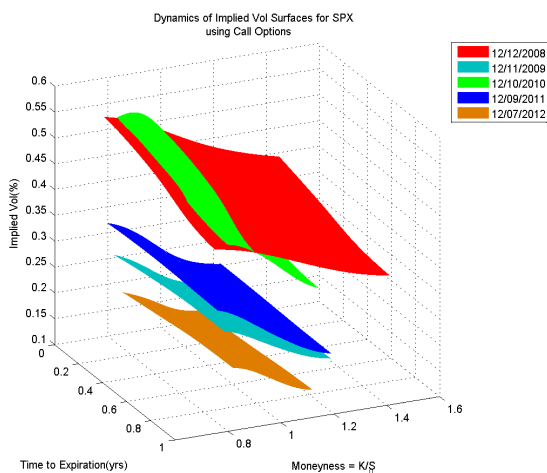
t = time to maturity

σ = volatility of the asset

The option contract has a strike price(S_t) and expiry date after time (t). The strike price of a call option is the price at which the contract allows the buyer to buy the stock. The expiry date is the time frame in which the option needs to be exercised or else will be deemed worthless. The

strike price can have options be in the money or out of the money. These terms are mere comparisons of the strike price and the stock price. For example, if AAPL is trading at \$155 and the option to buy Apple is at \$150 with an expiry of 7 days, then the option is in-the-money as the strike price (\$150) is less than the stock price (\$155). Again, volatility can quickly turn an in-the-money option into an out-of-the-money option. Therefore, predicting the implied volatility and trading it strategically can enable one to take advantage of the inherent arbitrage that can exist in the market.

Implied volatility is not a constant like the volatility in the option pricing. Dobi's paper, modeling volatility risk in equity options; a cross-sectional approach, graphed the implied volatility against time to expire and moneyness of the option. The graph below illustrates how volatility increases when the option reaches closer to expiration. Furthermore, in-the-money options have a higher implied volatility than out-of-the-money options. The research by Dobi helps in understanding the relationship implied volatility has with moneyness and time to expire. Her research validated the importance of implied volatility in the options market.



Modeling volatility risk in equity options; a cross-sectional approach (Doris Dobi 5)

This paper is specifically identifying the arbitrage in the implied volatility and trading on it. Volatility arbitrage trading is called Vol-arb in the finance industry. The concept mainly identifies option volatilities that are highly correlated. For example, the volatility of AAPL, NVDA, MEE, and MSFT are similar due to sector and their volatility correlations can be tested. If there is a high correlation between them and their volatility is stationary; then the concept of stationarity can be applied to infer mean reverting features of the dataset. Therefore, the temporary volatility mispricing could be used to take advantage of the delayed market effect on the option pricing. For example, if AAPL volatility is 1.3 and MSFT is at 1.5; then the assumption can be made that there is a mispricing and the volatility will converge. The fact that AAPL is under volatile than its basket means its option is cheaper than MSFT. Therefore, the investor should buy the AAPL option at the cheaper price and short the MSFT option at the higher price. Their mean reverting feature will eventually kick in and the AAPL volatility will increase while the MSFT volatility will decrease. Therefore, the investor will end up making a profit on the arbitrage that existed due to the misalignment of volatility between the highly correlated stock options.

Chapter 1

Data Description and Methodology

My research involved utilizing Wharton Research Data Services (WRDS) to retrieve option pricing data from Option Metrics. The data bank has petabytes of option pricing data and is extremely costly to retrieve. Harvard University’s collaboration with WRDS enabled me to get access to the dataset. My research started by pulling data from January 2020 to August 2023 for every option traded on the S&P 500. The data was over 30 GB and required sophisticated computing power to be operated on. Therefore, the scope of the paper was reduced to 6 months of pricing data focused on the top 100 hedge fund stocks according to hedge follows (the scope limitation is outlined in detail in Chapter 6 of this paper). This allowed me to compute the correlation on the stock option volatility and generate heatmaps. For this research, I focused on pricing NVDA (Nvidia) and TSLA (Tesla) option volatility as they are high-volume-traded stocks. The high volume of trades ensures liquidity; thus, making them good stock picks for a vol-arb trading strategy. The 6-month dataset was used to develop the heatmap and basket of stock options that correlate to NVDA and TSLA. Once the basket was determined, I utilized 6 years of data from January 2018 to August 2023 to develop the vol-arb trading model. The dataset has option data up to August 2023; hence, my research focuses on data up to that point only. Furthermore, pricing data was focused on the last 7 days before the option expired. The dataset has 7 days, 30 days, 60 days, 90 days, and 365 days option expiry terms. However, the action of most options happens around the final week of the options expiry; hence, for this paper, I limited the scope to the final 7-day pricing data. The paper is also focusing on American options rather than European options. Furthermore, the data is limited to the call options and any missing data columns were omitted from the research. The scope limitation section of this paper highlights different results obtained on the larger datasets to provide an insight into the future possibilities.

The data had over 30 columns that identified different aspects of the option prices. The following diagram gives a quick understanding of what the data entails. The appendix has a detailed explanation followed by a link to get further insight into the column headers:

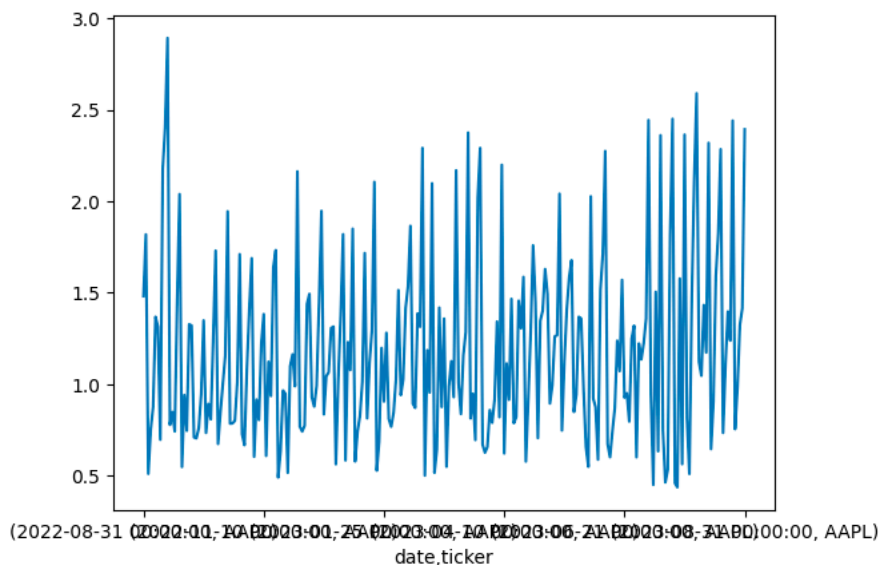
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
1	secid	date	symbol	exdate	last_date	strike_pri	best_bid	best_offe	volume	open_inte	impl_volat	delta	gamma	vega	theta	optionid	contract_j	cusip	ticker	sic	issuer	
44	101594	9/1/22	AAPL 220902C185000	9/2/22	9/1/22	185000	0	0.01	20	4241	1.040076	0.002021	0.000745	0.05306	-10.0664	148533364	100	3783310	AAPL		3571	APPLE INC
45	101594	9/1/22	AAPL 220902C187500	9/2/22	9/1/22	187500	0	0.01	10	2067	1.119537	0.00189	0.000652	0.049891	-10.19563	148720102	100	3783310	AAPL		3571	APPLE INC
46	101594	9/1/22	AAPL 220902C190000	9/2/22	9/1/22	190000	0	0.01	110	2189	1.197399	0.001778	0.000576	0.04715	-10.31472	148533365	100	3783310	AAPL		3571	APPLE INC
47	101594	9/1/22	AAPL 220902C192500	9/2/22	8/30/22	192500	0	0.01	0	783	1.273751	0.001681	0.000515	0.044772	-10.42509	148720103	100	3783310	AAPL		3571	APPLE INC
48	101594	9/1/22	AAPL 220902C195000	9/2/22	9/1/22	195000	0	0.01	50	572	1.348674	0.001596	0.000464	0.042697	-10.52794	148533366	100	3783310	AAPL		3571	APPLE INC
49	101594	9/1/22	AAPL 220902C197500	9/2/22	8/30/22	197500	0	0.01	0	354	1.422242	0.001521	0.000421	0.040875	-10.62426	148720104	100	3783310	AAPL		3571	APPLE INC
50	101594	9/1/22	AAPL 220902C200000	9/2/22	8/26/22	200000	0	0.01	0	1933	1.494521	0.001454	0.000384	0.039259	-10.71493	148533367	100	3783310	AAPL		3571	APPLE INC
51	101594	9/1/22	AAPL 220902C205000	9/2/22	8/24/22	205000	0	0.01	0	248	1.635438	0.00134	0.000326	0.036498	-10.88198	148533368	100	3783310	AAPL		3571	APPLE INC
52	101594	9/1/22	AAPL 220902C210000	9/2/22	8/19/22	210000	0	0.01	0	65	1.771824	0.001246	0.000282	0.034194	-11.03339	148533369	100	3783310	AAPL		3571	APPLE INC
53	101594	9/1/22	AAPL 220902C215000	9/2/22	8/19/22	215000	0	0.01	0	283	1.904	0.001168	0.000247	0.032218	-11.17201	148533370	100	3783310	AAPL		3571	APPLE INC
54	101594	9/1/22	AAPL 220902C220000	9/2/22	8/29/22	220000	0	0.01	0	57	2.032244	0.001101	0.000219	0.0305	-11.29979	148533371	100	3783310	AAPL		3571	APPLE INC
55	101594	9/1/22	AAPL 220902C225000	9/2/22	8/11/22	225000	0	0.01	0	11	2.156801	0.001044	0.000197	0.028995	-11.41816	148533372	100	3783310	AAPL		3571	APPLE INC
56	101594	9/1/22	AAPL 220902C230000	9/2/22		230000	0	0.01	0	0	2.277893	0.000994	0.000178	0.027676	-11.52828	148596218	100	3783310	AAPL		3571	APPLE INC
57	101594	9/1/22	AAPL 220902C235000	9/2/22		235000	0	0.01	0	0	2.395723	0.000995	0.000162	0.026515	-11.63111	148596219	100	3783310	AAPL		3571	APPLE INC
58	101594	9/1/22	AAPL 220902C240000	9/2/22	8/29/22	240000	0	0.01	0	12	2.510479	0.000911	0.000149	0.025492	-11.7275	148596220	100	3783310	AAPL		3571	APPLE INC
59	101594	9/1/22	AAPL 220902C245000	9/2/22		245000	0	0.01	0	0	2.62233	0.000876	0.000138	0.024584	-11.81811	148596221	100	3783310	AAPL		3571	APPLE INC
60	101594	9/1/22	AAPL 220902C250000	9/2/22	9/1/22	250000	0	0.01	2	277	2.731431	0.000845	0.000128	0.023773	-11.90367	148596222	100	3783310	AAPL		3571	APPLE INC
61																						
62																						
63		9/1/22					\$ 17.17	\$ 17.25	359,403.00	247,364.00		1.82										
64		8/31/22					\$ 16.75	\$ 17.24	262,118.00	230,694.00		1.48										
65																						

The data above has a quick snapshot of AAPL option data. Cell A44 is showing an option price on 1 September 2022 for an option that is expiring on 2 September 2022. The option bid price is \$0 and the offer price is \$0.01. This is a classic example of an out-of-the-money option.

The strike price was \$185 when the stock was trading around \$150 on that day. Therefore, the option was \$35 out of the money (OOM). The volume traded was 20 and the open interest was 4241. The open interest highlights the number of options still available in the market. The implied volatility, cell K44 is the main point of highlight for our research. The implied volatility is 1.04. The remaining columns highlight the Greeks. These are different metrics that can help a trader understand how the stock price movement can affect the option price. We have limited our scope to focus on the implied volatility column.

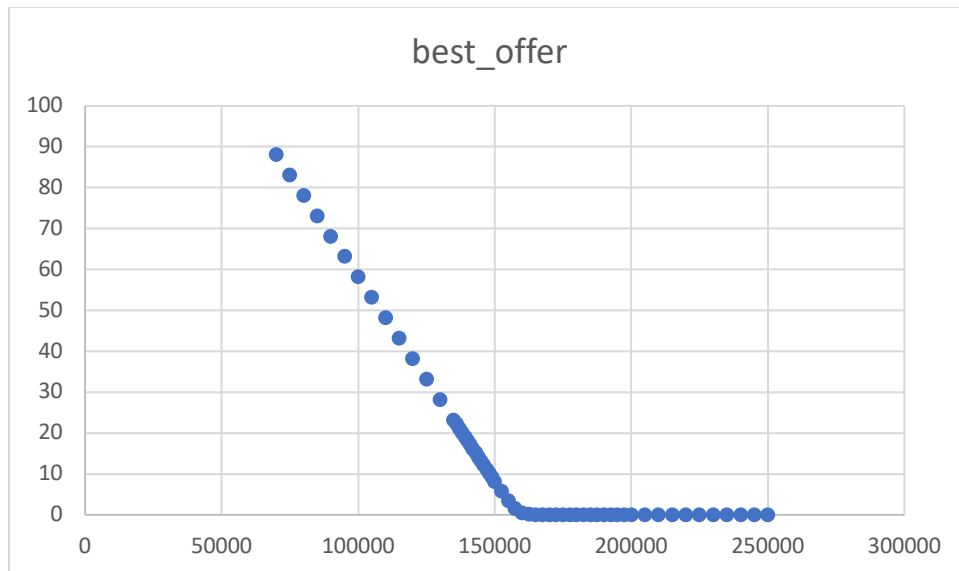
The data set has over 20 options with different strike prices for a single stock on a single expiry date and that makes the option dataset gigantic compared to a stock dataset. To consolidate the data and get a better understanding of the volatility, an average of the daily volatility of all options for a particular stock was taken. The Python file attached to this paper illustrates that. Cell K63 illustrates the average of 1 September 2022 volatility at 1.82. The bid and offer price were also averaged to get an average option price on that particular date. Daniel’s research paper reduced the size of the dataset using a similar average; however, his paper focused on the options whose moneyness was greater than 0.8 (Daniels 44). His paper highlighted that “trading volume tends to concentrate on options close to being at-the-money, [hence] deleted 55,647 options with moneyness (i.e. St/Kt) less than 0.8 (55,647 observations) and greater than 1.2 (832,892 observations) (Daniels 44).”

The averaging of the data allowed me to draw a time series of implied volatility movements for each stock option and made the data more manageable. Below is AAPL stock option volatility graphed over 1 year from 2022 August to 2023 August. The data identifies stationarity. The analysis of the graph and further explanation of stationarity tests is explained in the analysis section of the paper.

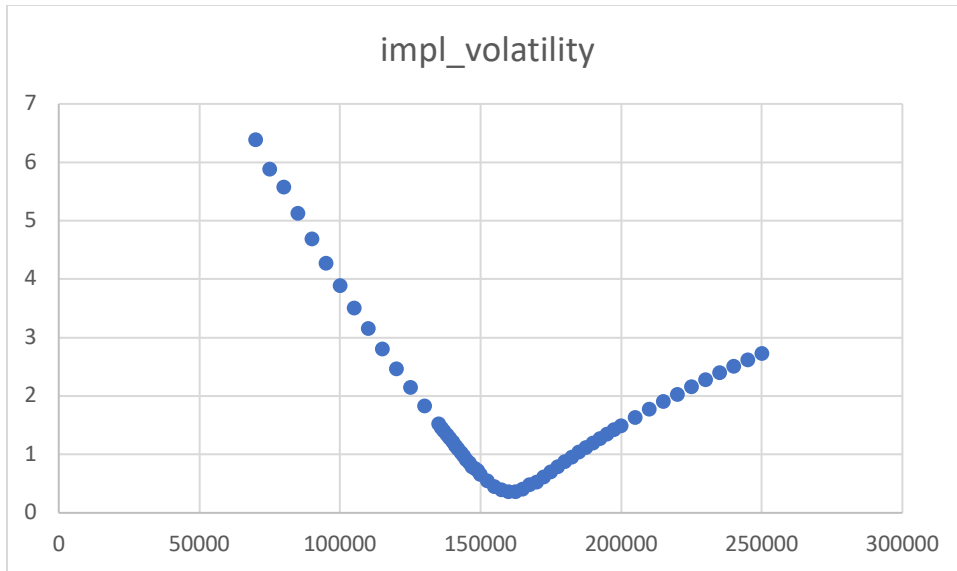


There were further data observations that were made on the dataset before averaging the volatility. The chart below highlights the option bid price against the strike price of the option.

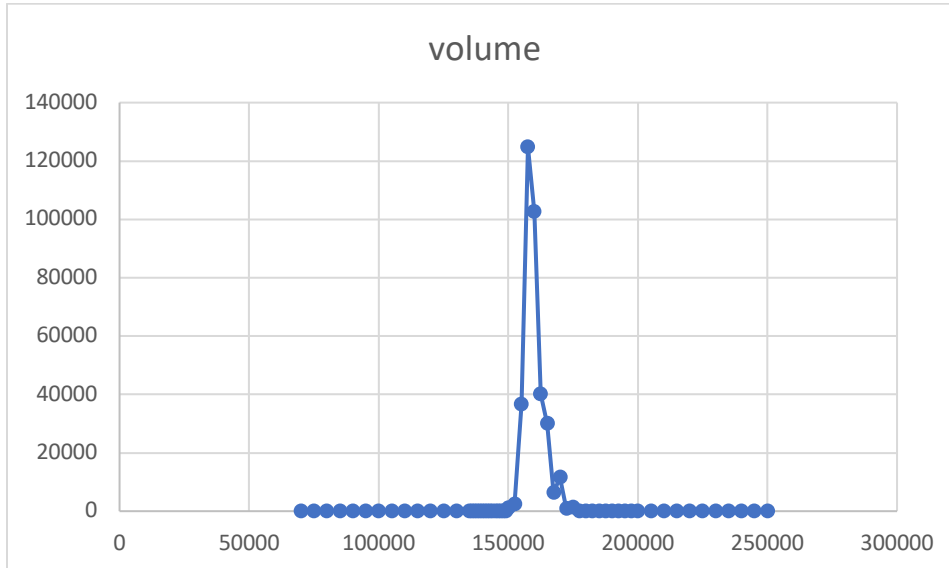
The graph follows the classic hockey stick graph. The options that are in and at the money are valuable while the options that are out of the money hold no value. The straight line on the x-axis where $y=0$ illustrates the option strike prices where the option itself holds no value. The right to buy a contract becomes valuable as the option strike price is lower than the market stock price. The hockey stick graph makes sense because in-the-money options will lead to an immediate profit; hence, the added value of the contract.



Further data analysis was done to get a sense of the data. The implied volatility was plotted against the strike price of the option. The graph follows the “u-shape” implied volatility curve wherein the volatility is lowest at-the-money and increases as the strike prices are further away from the underlying stock price. The graph also gives an understanding of the options that are sold in the market. There are more options sold with a lower strike price difference when the option is trading at the money. For example, in the case of AAPL, there are more options near \$150 as the incremental strike price difference between the options is lowered to \$5. Therefore, the options are trading in increments of \$5 with strike prices of \$125, \$130, 135, 140, 145 etc. However, the options that are trading away from the at-the-money option have a jump of \$10 where one option would trade at \$100 and the next would trade at \$110, and so on. The graph of the volatility below illustrates this point clearly as the dots are more clustered together when they are closer to at-the-money options. The data graph below helps in allowing averages to be taken of the implied volatility to get a single volatility that represents the option that expires on a single day. This helps in getting one implied volatility that reflects the option for a particular stock on a specific day (Daniels).

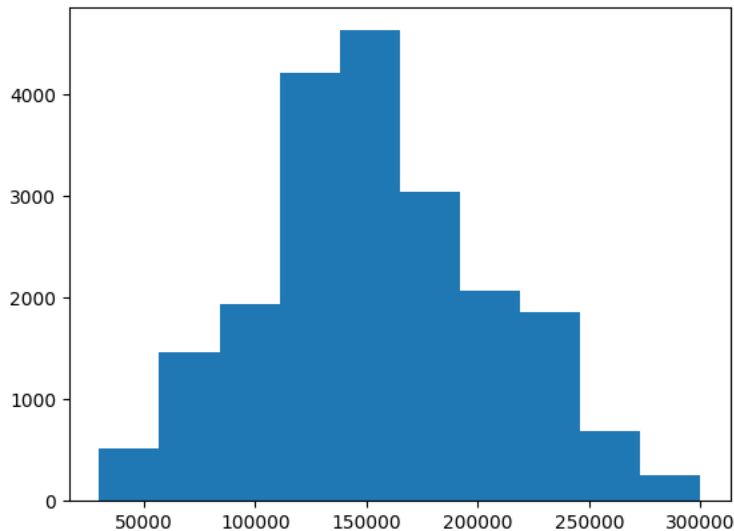


The data was further analyzed to get a sense of how the volume of an option is relative to the strike price. The volume below is for the AAPL call options expiring on 2 September 2022. The visualization illustrates that the majority of trading happens near at-the-money options. Hence, liquidity for the options market happens closer to at-the-money options while OOM options have far less liquidity in trading volumes.



The major point to note in the data section is that only 7 days before option expiry data was analyzed since the majority of trade happens during that time frame. Furthermore, the option data is 100 times bigger than the stock market; therefore, the effective way to model implied volatility and build a vol-arb strategy is by averaging the implied volatility for a stock option with the same expiry date.

The data analytics was carried out in Python to get a sense of the number of options issued at different strike prices. The plotted histogram for AAPL shows that the majority of options are issued around the strike price. This again helps to better understand the raw data:

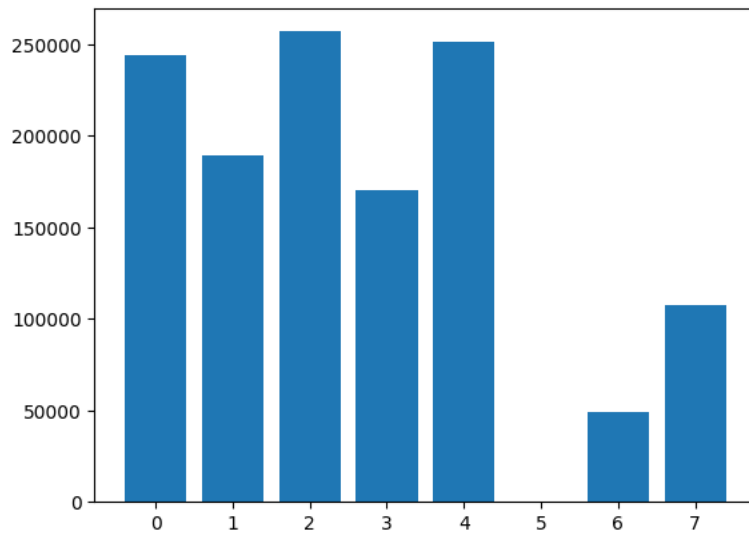


The bid-ask spread for different strike prices for AAPL is also plotted before an average implied volatility is taken. The AAPL options market seems very liquid as the spread is very small; however, the research could be furthered in creating trading strategies around the bid-ask spread.



The days to expire were calculated by subtracting the expiry date from the current date of the data and a new column day to expire was created. This was used to visualize the volume trade around days to expire for AAPL. The data tells a story where the trading volume is closer to the final day of an option expiry. The volume on 5 days before expiry was 0. This is because options expire on Friday and 5 days before Friday is a Sunday; hence, no trading activities occur on Sunday. The 6 days before expiry had some trading volume. This is strange as trading doesn't

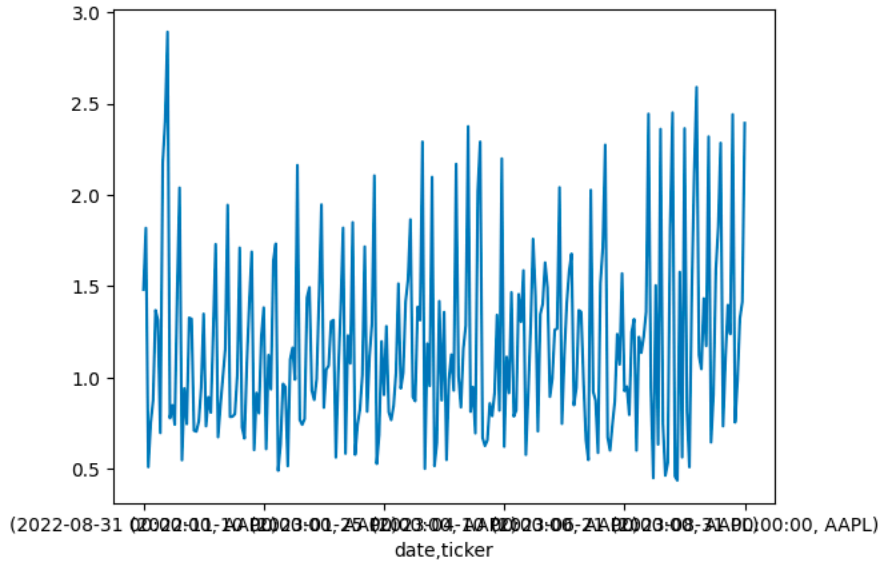
occur on Saturday. The irregularities of the data on Saturday were out of the scope of this project and were incorporated in drawing analysis on the vol-arb trading strategy.



Chapter 2

Analysis of Code and Data

The first step after averaging the volatility was to plot it. The plotted volatility for AAPL stock gave a story of the data's stationarity. Data is required to be stationary when running forecasting models because it improves predictability and allows for fewer forecasting errors. Data stationarity can be tested through Augmented-Dickey Fuller (ADF) statistics or p-value tests. The stationarity of the data means that it has a constant mean and constant standard deviation. Furthermore, it implies that the data has a mean reverting signal which helps in the forecasting ability of the data. Below is the plotted implied volatility for AAPL



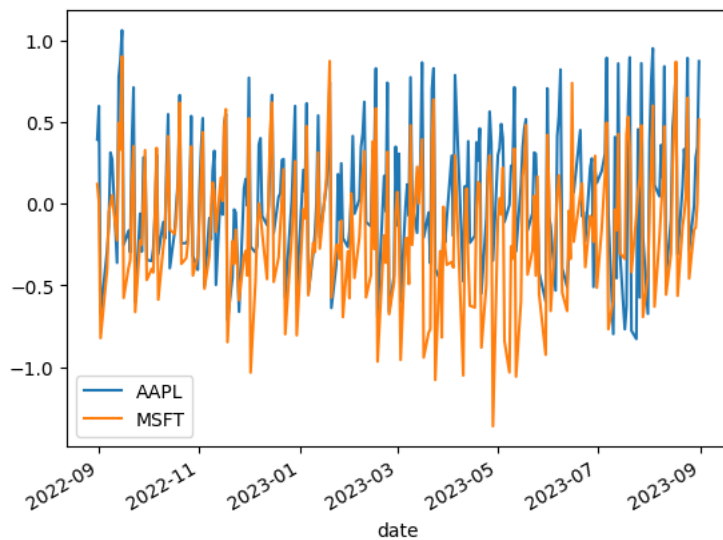
The ADF statistics is -3.86 and the p-value is 0.002. Since the ADF is negative and the p-value is less than 0.05, the data is stationary. The graph can visually illustrate this point as the data is reverting around the mean.

```

ADF Statistic: -3.8669317300500508
p-value: 0.0022921487399941787
Critical Values:
  1%, -3.4582467982399105
Critical Values:
  5%, -2.8738137461081323
Critical Values:
 10%, -2.5733111490323846

```

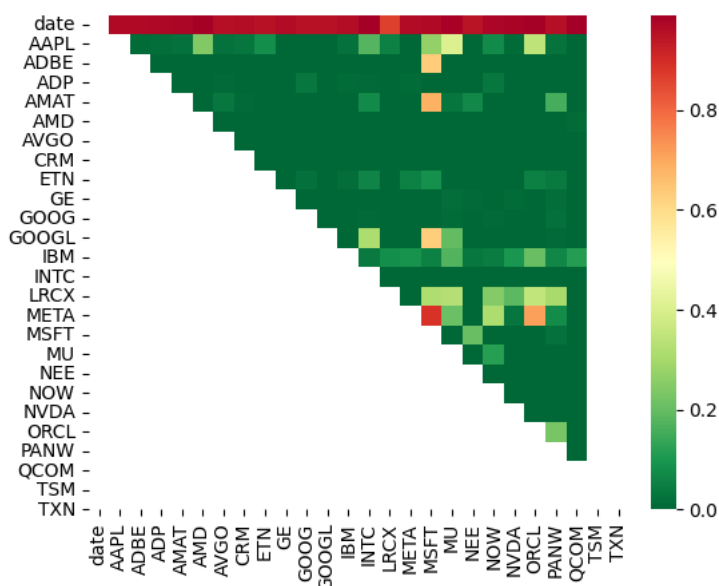
The plotted AAPL implied volatility illustrated stationarity. Therefore, I started to plot similar nature stocks over AAPL’s implied volatility. I plotted Microsoft’s (MSFT) implied volatility over AAPL’s. The graph showed a lot of similarities:



Therefore, I started analyzing the cointegration for the tech sector stock option volatility. I dropped all the infinite and not available values. Then I filtered the data on the following stocks

```
tickers_to_filter = ['date', 'AAPL', 'ADBE', 'ADP', 'AMAT', 'AMD', 'AVGO',
                    'CRM', 'ETN', 'GE', 'GOOG', 'GOOGL', 'IBM', 'INTC', 'LRCX',
                    'META', 'MSFT', 'MU', 'NEE', 'NOW', 'NVDA', 'ORCL', 'PANW', 'QCOM', 'TSM',
                    'TXN']
```

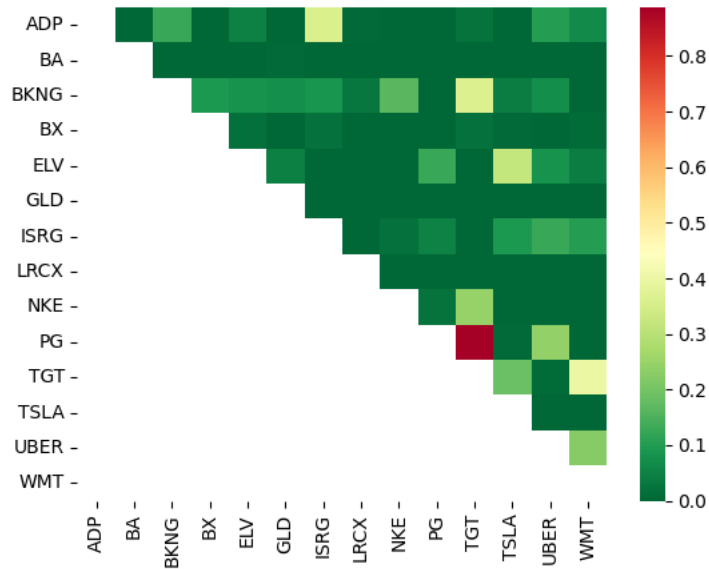
The filtered stocks were tested for cointegration and the resulting p-values were plotted on a heatmap. The correlation in the tech sector is highlighted below:



The dark green illustrates high cointegration between the tech sectors' implied volatility. In the tech sector, NVDA has been a volatile stock of late. Hence, I picked a basket of stocks that cointegrated with NVDA and could mimic the NVDA implied volatility. The stock options I picked for the NVDA basket are:

```
['GOOG', 'NOW', 'MSFT', 'AMD', 'ADP']
```

Furthermore, I was interested in analyzing TSLA stock option volatility against cross-sector stock option volatility. Therefore, I ran a cointegration test on different sectors' stock options along with TSLA. The result of the cointegration test is below:



The above dataset helped in identifying stock option volatility that cointegrated with TSLA. The stock basket selected for the vol-arb trading for TSLA is :

```
[ 'PG', 'NKE', 'LRCX', 'GLD', 'BA' ]
```

The heatmap portrays a high cointegration of the stock option volatility of the above stocks and TSLA

Chapter 3

Model Building – Multi Linear Regression

Multi Linear Regression - NVDA 6-Month Data

The cointegrated tech sector basket was analyzed for vol-arb trading. To forecast the volatility of the basket, a training and test set split was done. The data was 6 months of options data from 3/01/2023 to 08/31/2023. The data was split on 30 June 2023. Therefore, the training set had 4 months of data and the test set had 2 months of data.

Thereafter, a multi-linear regression model was fit on the logged implied volatility of NVDA and the basket of stocks that mimicked NVDA’s implied volatility. The basket of stocks selected for this research are:

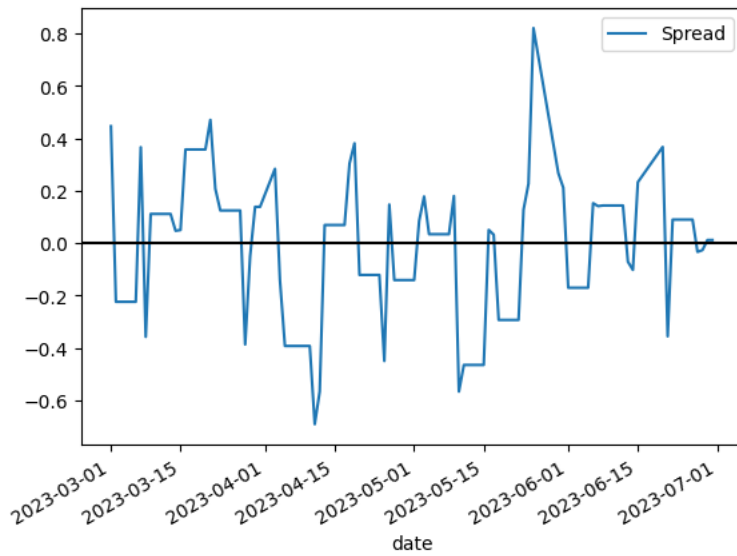
```
[ 'GOOG', 'NOW', 'MSFT', 'AMD', 'ADP' ]
```

The regression was then used to predict the NVDA option volatility. The predicted NVDA volatility was compared against the actual NVDA volatility to see how the model performed. The difference in the prediction was tested for stationarity and it gave a result of

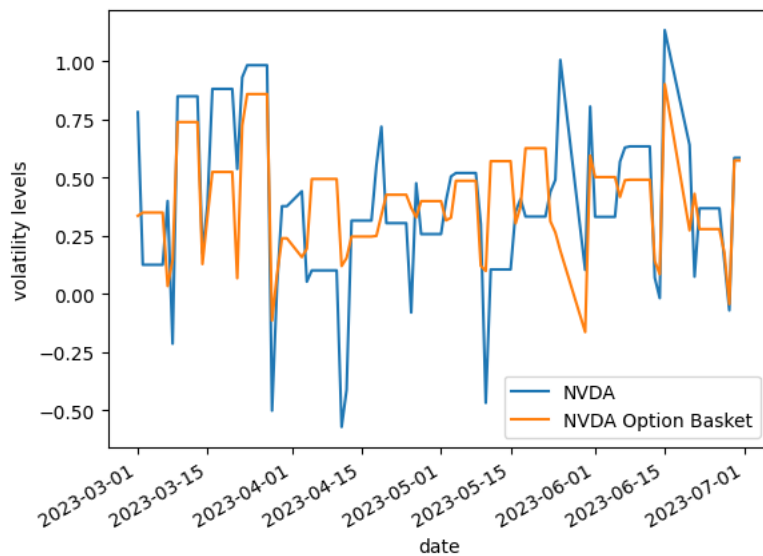
```
MLR RSquared: 0.3598336734532721
p-value for in-sample stationarity: 4.365541441587516e-07
```

t statistics for in sample stationarity: -5.812623181796803

The result shows that the p-value was less than 0.05; hence, the spread is stationary. However, the MLR r square was 0.36; therefore, the model is not a good predictor of NVDA volatility. However, we are not trying to predict the NVDA volatility in the vol-arb strategy. We are instead trading the mean reverting signal. Therefore, since there is high cointegration an investor can easily trade the arbitrage that exists between the predicted NVDA basket volatility and the actual NVDA volatility. Below is a graph illustrating the spread of the difference :



The graph showing the predicted value against the actual implied volatility is below:

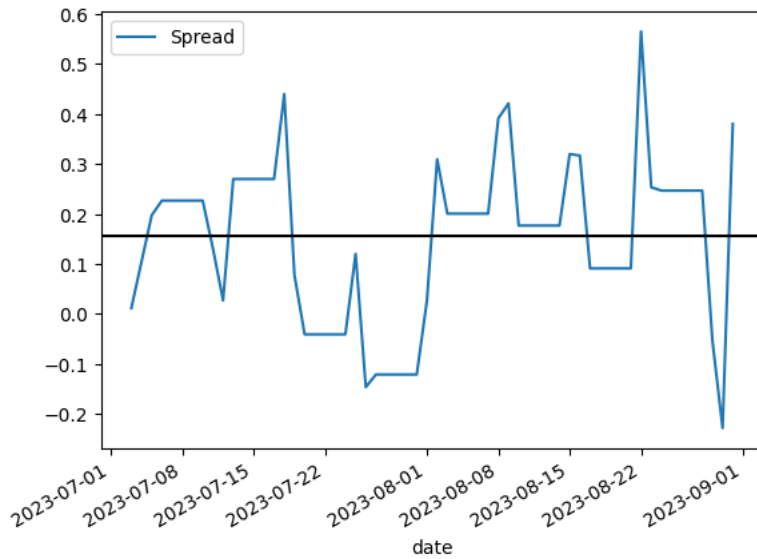


The NVDA volatility reverts to the mean; therefore, the investor can long the call option when the NVDA basket's implied volatility is less than the NVDA basket and can short the call option

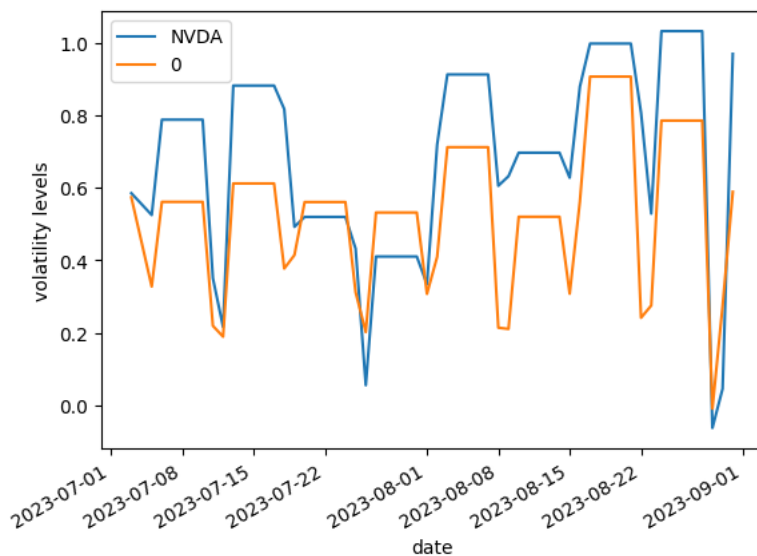
when the NVDA implied volatility is higher than the NVDA basket's implied volatility. This is a classic vol-arb strategy that takes advantage of the volatility difference in the market.

A similar test is done on the test data set and the results are as follows:

MLR RSquared: 0.6611205978855109
p-value for in-sample stationarity: 0.0015252056724458247
t statistics for in sample stationarity: -3.978884362058182

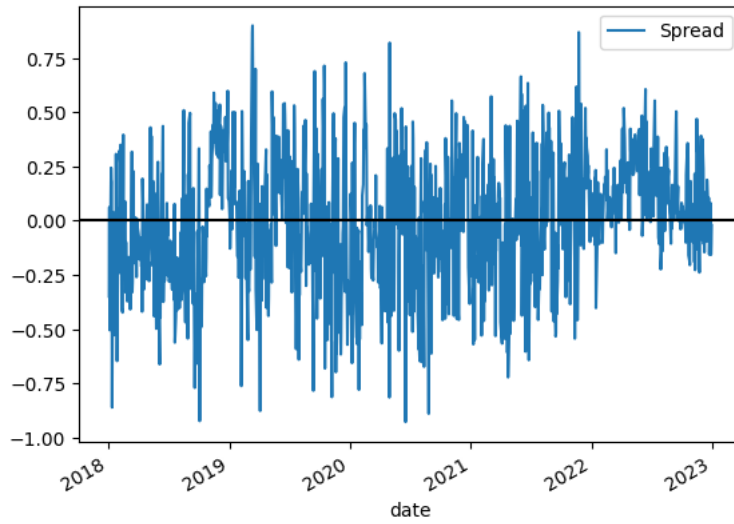


The spread is stationary as the p-value is less than 0.05 and the above graph illustrates a mean reverting signal. Furthermore, the graph below shows the basket plotted against the NVDA actual logged volatility. The signal is mean reverting and the investor can trade the volatility arbitrage in the following situation.



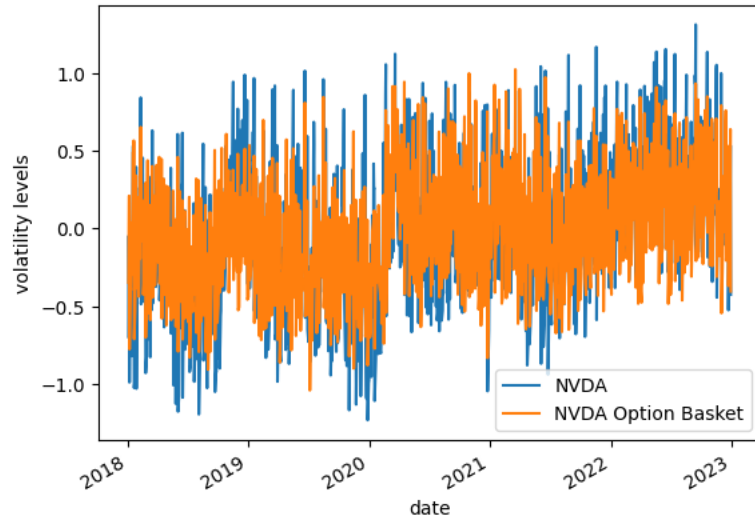
Multi Linear Regression - NVDA 6 Years Data

The model above had very few data points to build a volatility arbitrage model. Hence, 6 years of data was imported from the WRDS dataset. The data range is from 1 Jan 2018 to 31 August 2023. The data was trained on the first 5 years from 1 Jan 2018 to 31 December 2022. The test set was the remaining 8 months of 2023 from Jan 2023 to August 2023. The MLR mode is built on the training dataset and the forecasted values are compared against the actual NVDA logged volatility. The spread of the dataset is graphed below:



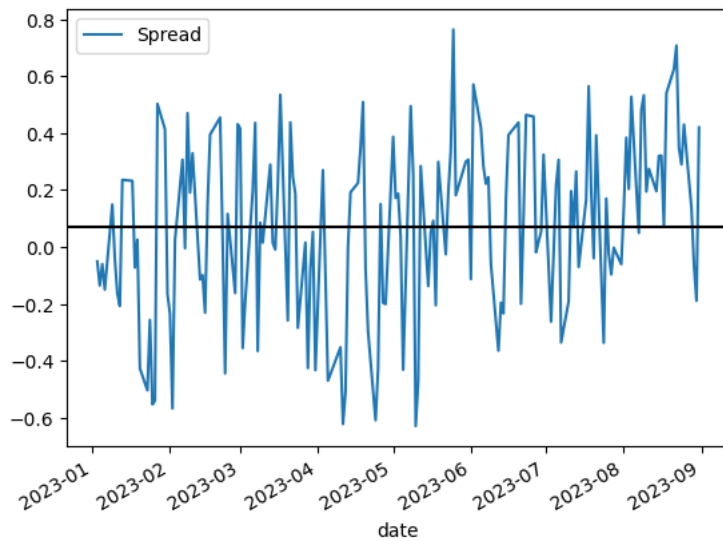
```
MLR RSquared: 0.615256481962784
p-value for in-sample stationarity: 6.119061823123265e-05
t statistics for in sample stationarity: -4.77317208063876
```

The MLR has a r square of 0.62; this is a really good number as 62% of the NVDA logged volatility is explained by the basket stock's coefficients. Furthermore, the p-value is $6.12e-05$ which is less than 0.05; hence, proving the spread to be stationary. This model is powerful for predicting volatility arbitrage. The predicted value is plotted against the actual volatility value for the training set:



The graph above illustrates the cointegration and the predictability power of the MLR. The mean reverting signal can also be observed from the graph above.

The MLR model was kept to work on the test data set for the remaining 8 months of 2023. The result of the spread is below:

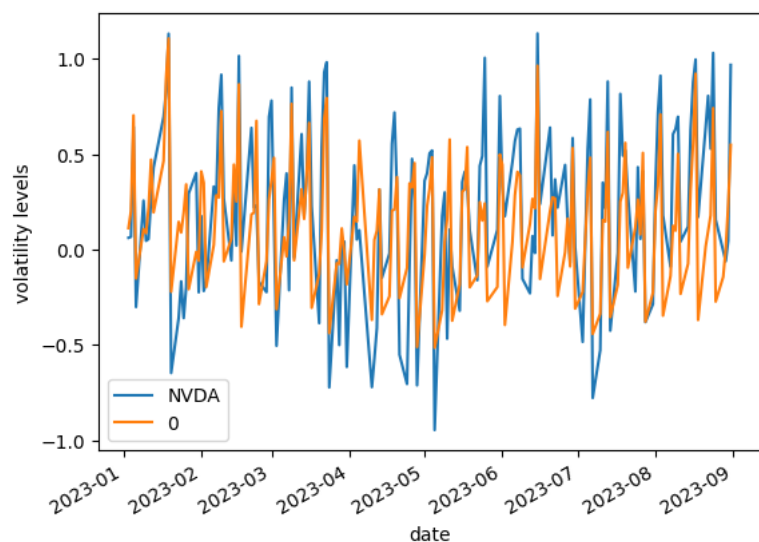


p-value for in-sample stationarity: 1.427881295338533e-15

t statistics for in-sample stationarity: -9.258519261999913

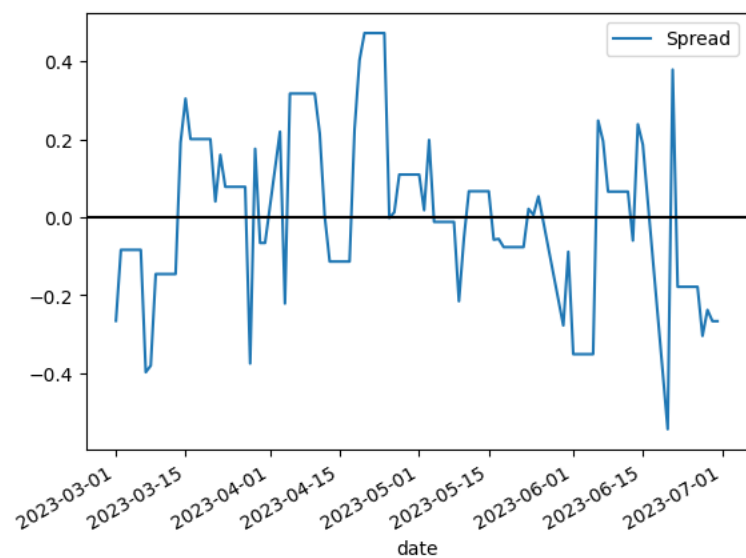
The p-value is 1.43e-15 which is lower than 0.05; hence proving the stationarity of the spread.

The actual vs forecasted basket is plotted below for the test set. The investor can use this to trade the volatility arbitrage that exists between NVDA's option volatility and its basket. Whenever, the NVDA option volatility is lower than the basket, the investor can go long on the call option and vice versa.



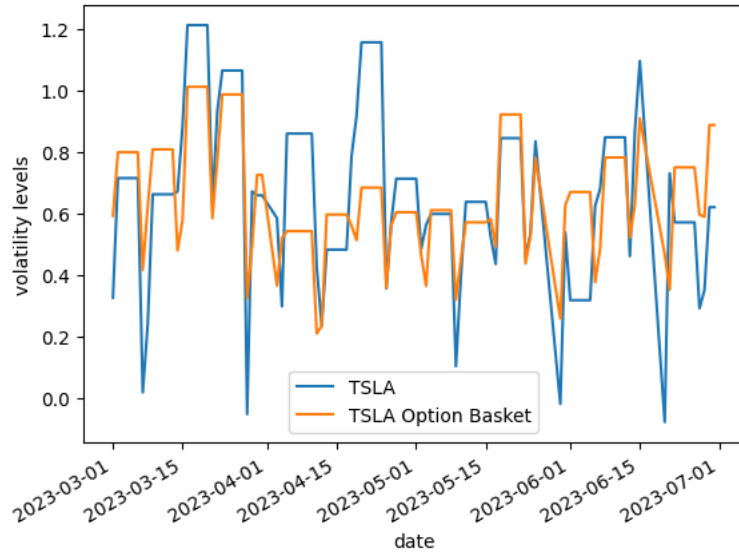
Multi Linear Regression - TSLA 6-Month Data

The same test between TSLA and its basket of stock options was done to get an understanding of the volatility. The training and test dataset was split into 4 months and 2 months respectively. The training set result of the spread for TSLA and its basket is below:

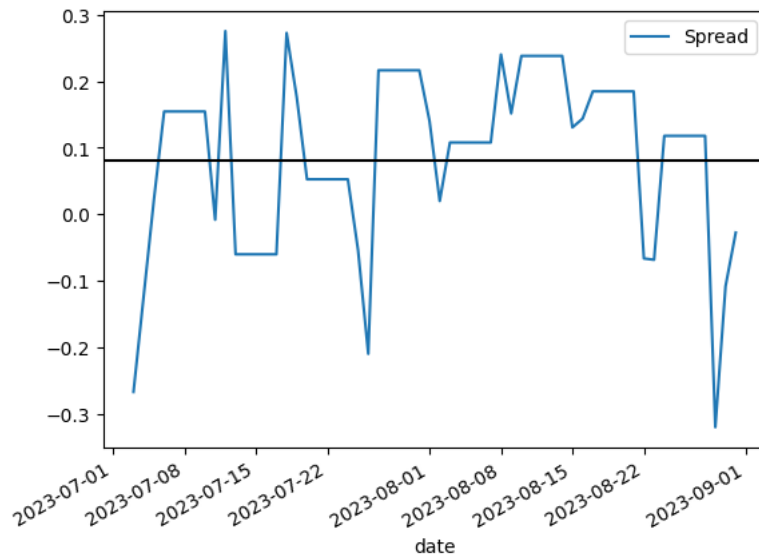


MLR RSquared: 0.4021363553545346
 p-value for in-sample stationarity: 0.003250233657528494
 t statistics for in sample stationarity: -3.768175161496668

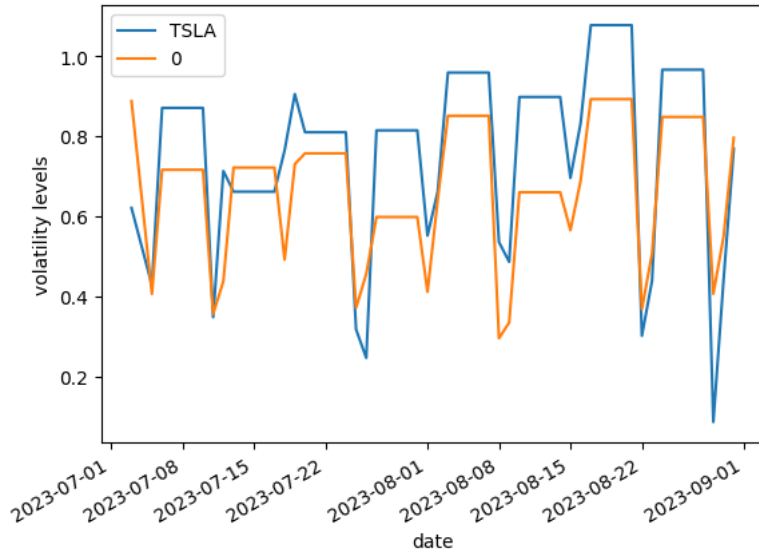
The rsquare value is 0.40 which is low; however, the spread is stationary as the p-value is less than 0.05. Therefore, an investor can trade the arbitrage as the volatility has a mean reverting signal. The forecasted value is plotted against the actual logged volatility below:



The MLR Model was applied to the test set. The spread and the forecasted values are graphed as follows:



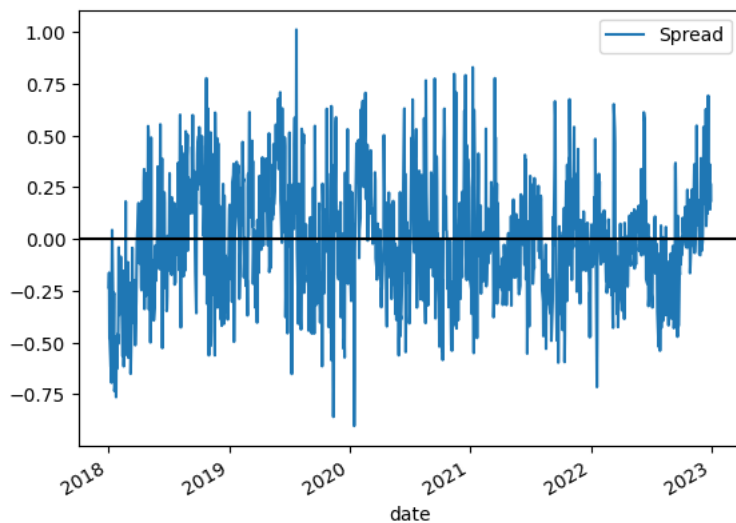
p-value for in-sample stationarity: 1.7701771584970987e-05
t statistics for in-sample stationarity: -5.049422220927951



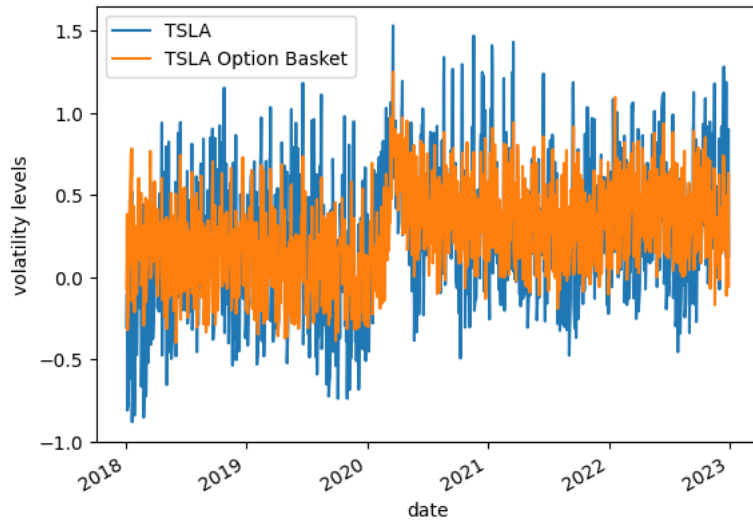
The p-value is $1.77e-05$ which indicates the stationarity of the spread. The graph above illustrates points where the volatility differs and an investor can long or short the call option based on the spread graph above. For example, on 2023-07-08, the implied volatility of TSLA is around 0.82 while the basket's implied volatility is 0.7; hence, an investor could short TSLA call options and wait for the mean reverting signal to converge. Once the implied volatility drops to converge with the basket, the investor would end up making a profit from the trade.

Multi Linear Regression - TSLA 6 Years Data

The dataset on TSLA was too small to get an understanding of the MLR Model. Therefore, similar to NVDA, a 6-year dataset was imported from WRDS. The first 5-year dataset was used to train the model and the remaining 8 months' data was used to test the model. The results of the trained model are as follows:

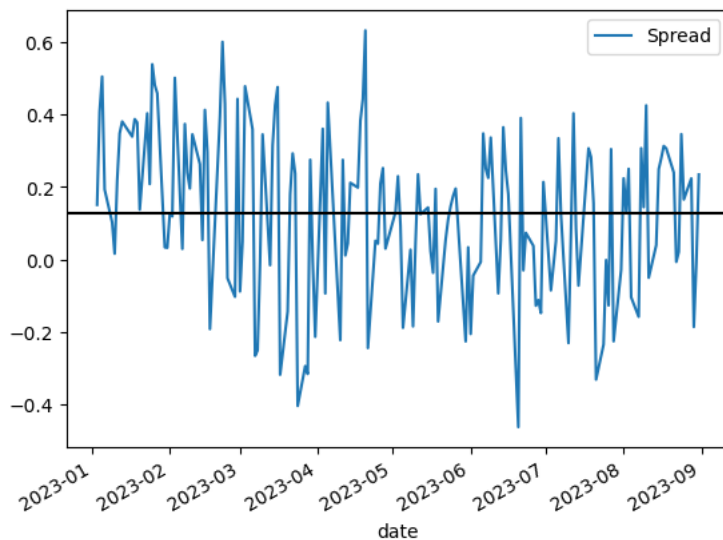


MLR RSquared: 0.48095040906746633
p-value for in-sample stationarity: 5.2268533347294655e-05
t statistics for in sample stationarity: -4.809008280440021

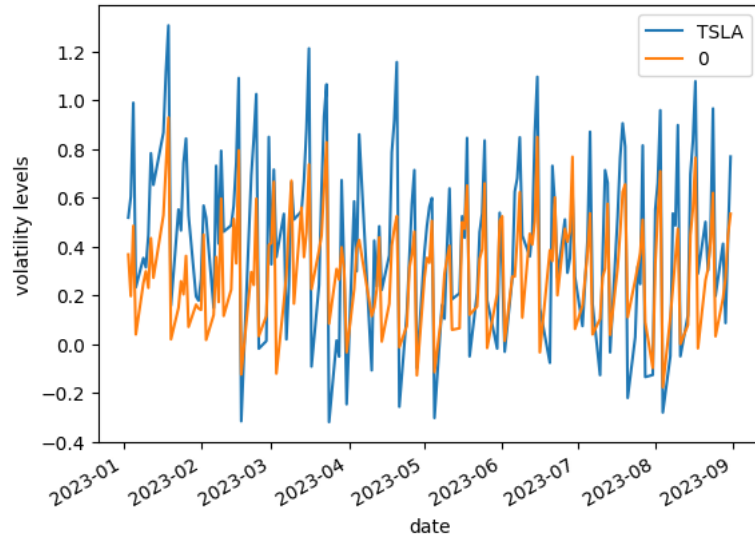


The p-value is less than 0.05; therefore, the spread is stationary. The r square is 0.48 which in this case is lower than the r square obtained for the NVDA prediction. It's safe to conclude that the basket of stocks can be reshuffled to get a better r-square value. However, for a vol-arb strategy 0.48 r-square can be still used to produce a decent strategy.

The test set results for the MLR spread of the basket and TSLA's implied volatility are as follows:



p-value for in-sample stationarity: 0.02167477289427217
t statistics for in sample stationarity: -3.171822036085348



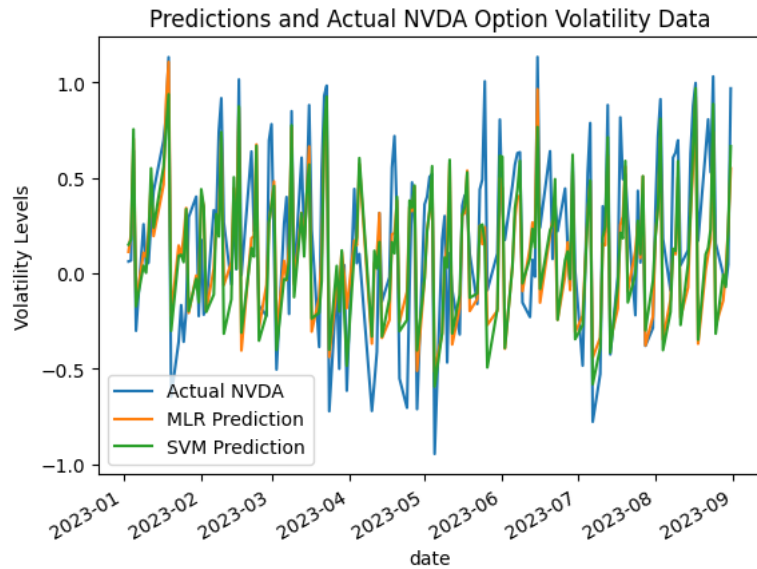
The p-value of the data is less than 0.05; therefore, the spread is stationary. The graph above illustrates how closely the volatility of the basket mimics TSLA implied volatility. This graph can be used to build a vol-arb strategy as portrayed in the trading module of this paper below.

Model Building – Support Vector Machine

Support Vector Machine – NVDA

Support Vector Machine modeling was used to predict the volatility on the basket of the NVDA stock options. The graph below shows the model results to be very similar to the MLR model. This proves that the basket is a good representation of NVDA’s volatility. The spread is stationary as the p-value is less than 0.05 at 2.01e-16.

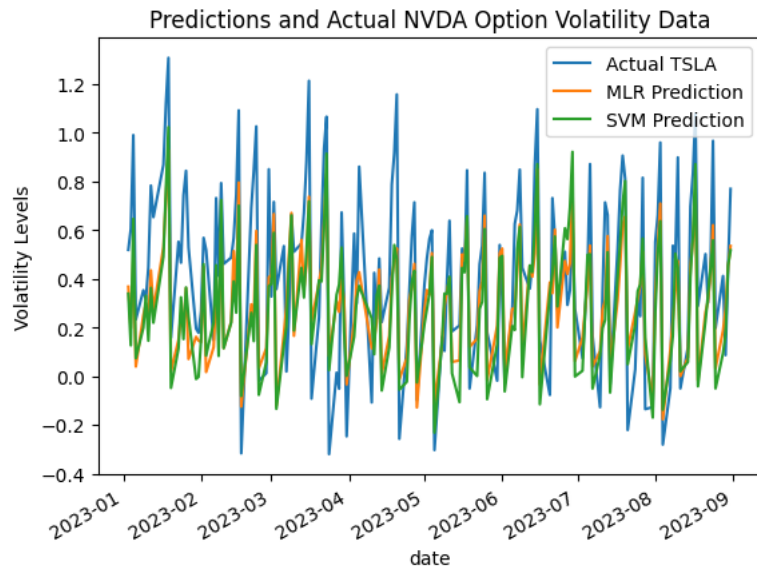
p-value for in-sample stationarity: 2.0107562721764302e-16
 t-statistics for in-sample stationarity: -9.593156268018701



Support Vector Machine - TSLA

The SVM modeling for TSLA paints a similar picture to the MLR prediction. Both models are good predictors of the TSLA stock option implied volatility. The p-value of the spread is $8.47e-20$ which indicates stationarity in the spread. Therefore, either model can be used for the vol-arb trading.

p-value for in-sample stationarity: $8.469519838772011e-20$
t-statistics for in-sample stationarity: -10.957917422363442



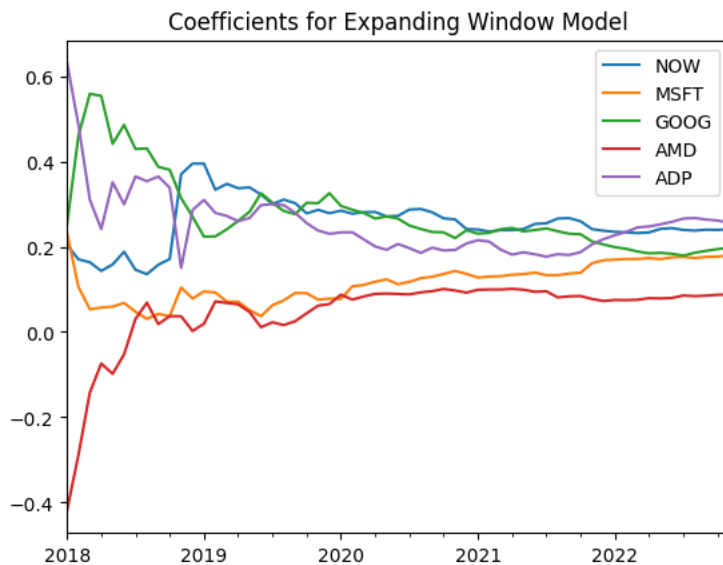
Chapter 4

Purging Data

The dataset is purged to avoid overfitting. Time series data needs to be split in a series and the order of time needs to be respected for creating a sophisticated model. The training and test splits conducted for building the MLR model above respected the continuity of time. The purging of the data omits certain data points in the training set to avoid overfitting. There are 2 visualizations done in this research to understand how the coefficients of the baskets are developed over time. The 2 methods are expanding window and sliding window.

Expanding Window

In the expanding window, the training set is expanded month-wise after every training. Therefore, towards the end of the training, the entire data set will be trained. The coefficients of the basket option stock volatility are noted after every expanding window and are plotted below:

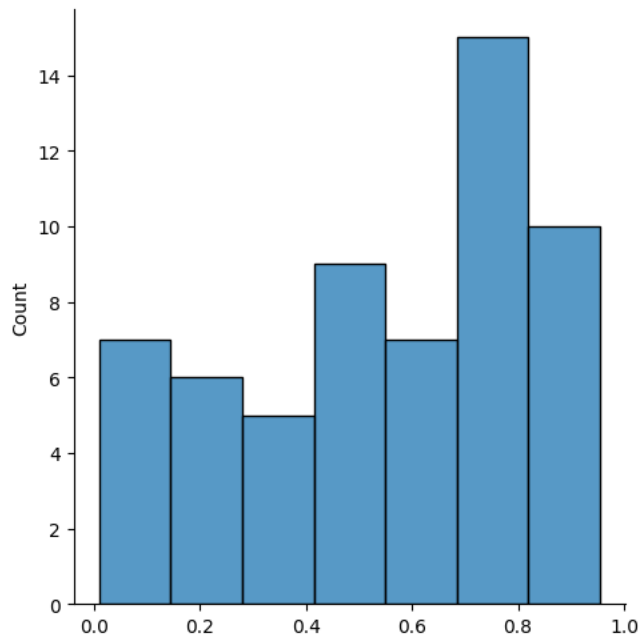


After 1 year of training, the model coefficients come very close to the final coefficient value that is derived after training the model on the 5 year dataset. The above graph helps to understand the behavior of the coefficient over the training set. It also indicates that a minimum of 1 year of training data is required to build an ideal model that could be used for vol-arb trading.

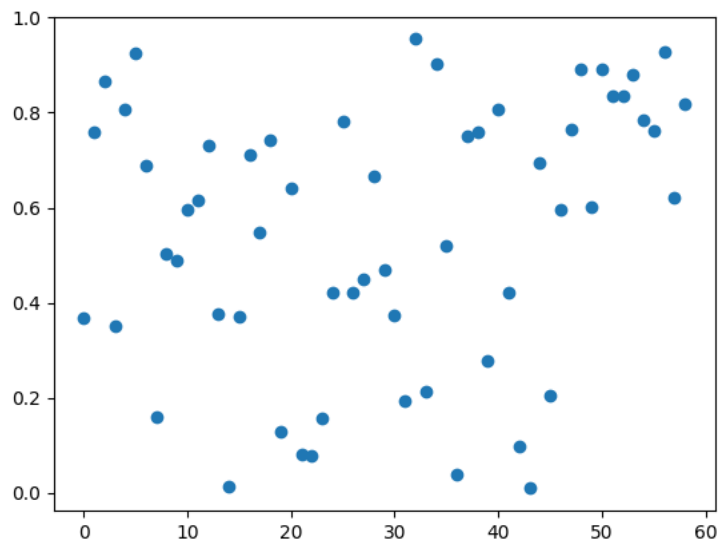
Sliding Window

The second way of training the data is by sliding the window of training data on each month along the dataset. The model is then built on the sliding window dataset and the model is validated on the validation set. Here the validation set data is purged by 2 days and then the next 15 days data is taken to predict the volatility. This helps in avoiding overfitting as certain data is

omitted for prediction. The r-square for each model on the sliding month is added to an array and finally plotted on a histogram:

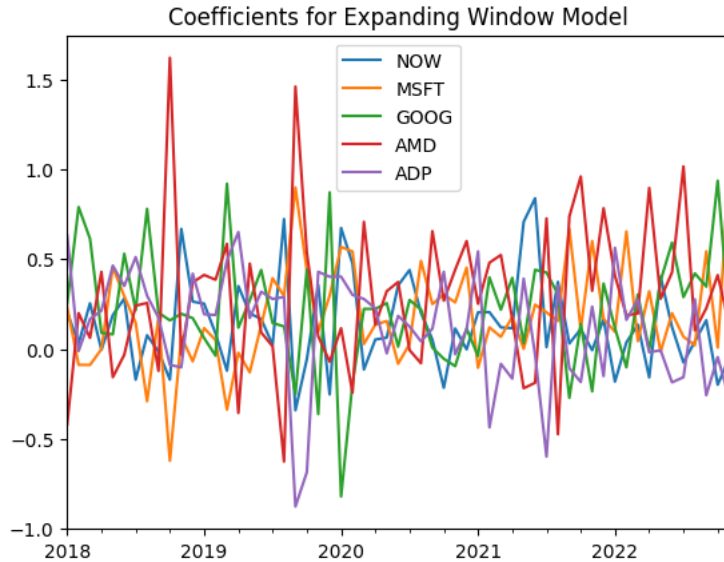


The graph above shows the model results for each training set that was trained and tested on. Most of the r square values for every model were above 0.5; hence, the sliding window does produce multiple models that can be used to predict the volatility. Furthermore, the scatter plot is developed to show how the r square varies for different training set models.



The randomness of the scatter plot shows how some training sets have a high percent of explanatory power towards the validation set while on the other hand, some training data cannot be utilized in the model building.

The coefficients for the expanding window are plotted and the noise in the graph indicates the variability of each training sets predicting power. The sliding model can be used for model building as it helps with the issue of overfitting; however, for vol-arb trading, the traditional training and test set split is sufficient to build a strong trading model.

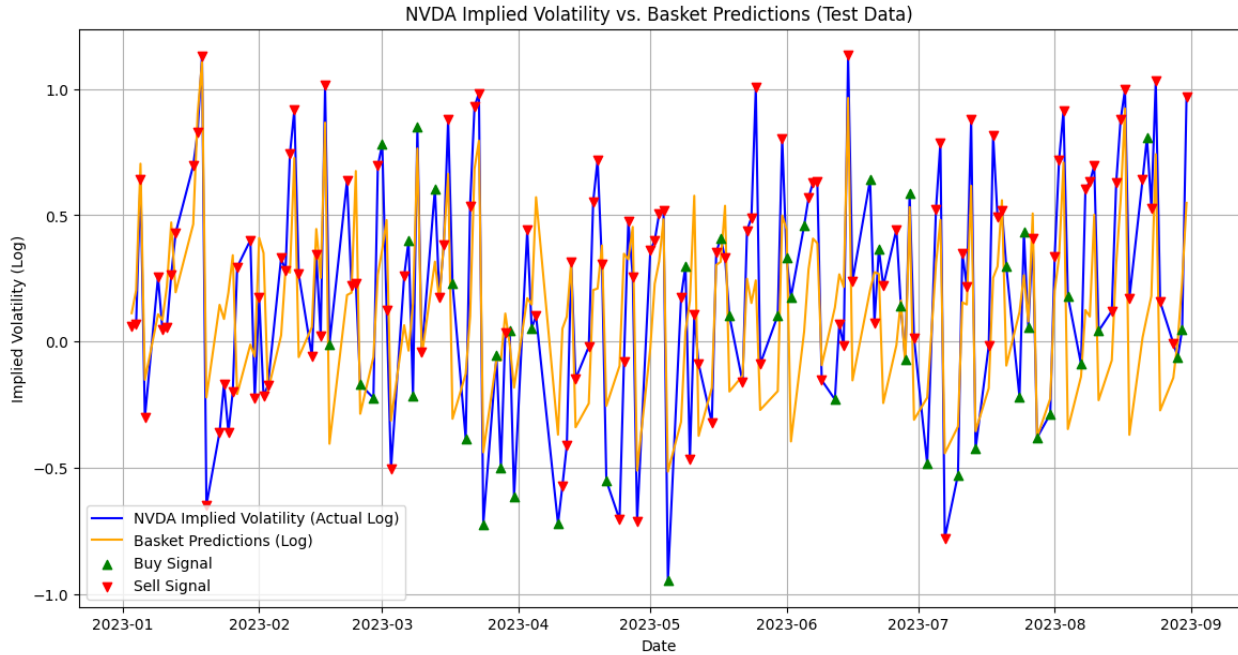


Chapter 5

Buy and Sell Signals

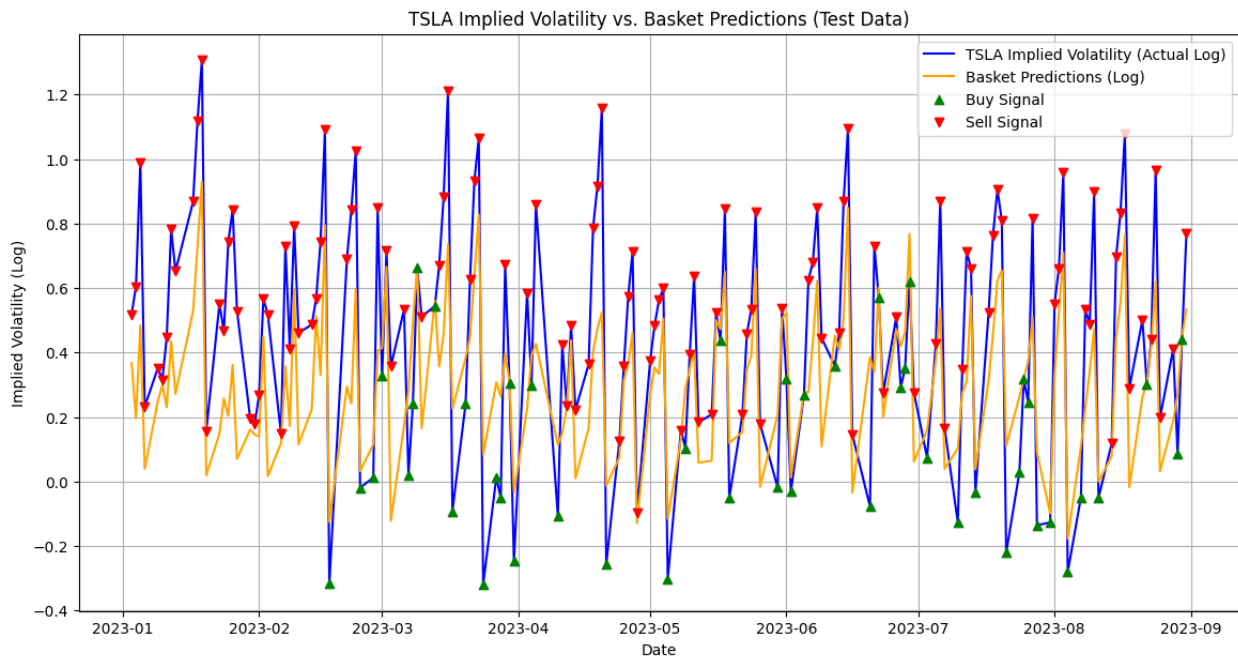
NVDA Stock Vol-Arb Trading

The final part of this research included developing buy and sell signals. Whenever the implied volatility of NVDA was less than the basket's implied volatility, the buy signal was indicated with a green arrow. This meant going long on the call option for NVDA. On the contrary, whenever the implied volatility of NVDA was higher than the basket of NVDA a sell signal was raised with a red arrow. This meant shorting the NVDA call option as the price of the call option was expected to fall due to the mean reverting nature of the implied volatility. This strategy is graphed below for 2023. An investor can use this signal to trade the implied volatility arbitrage.



TSLA Stock Vol-Arb Trading

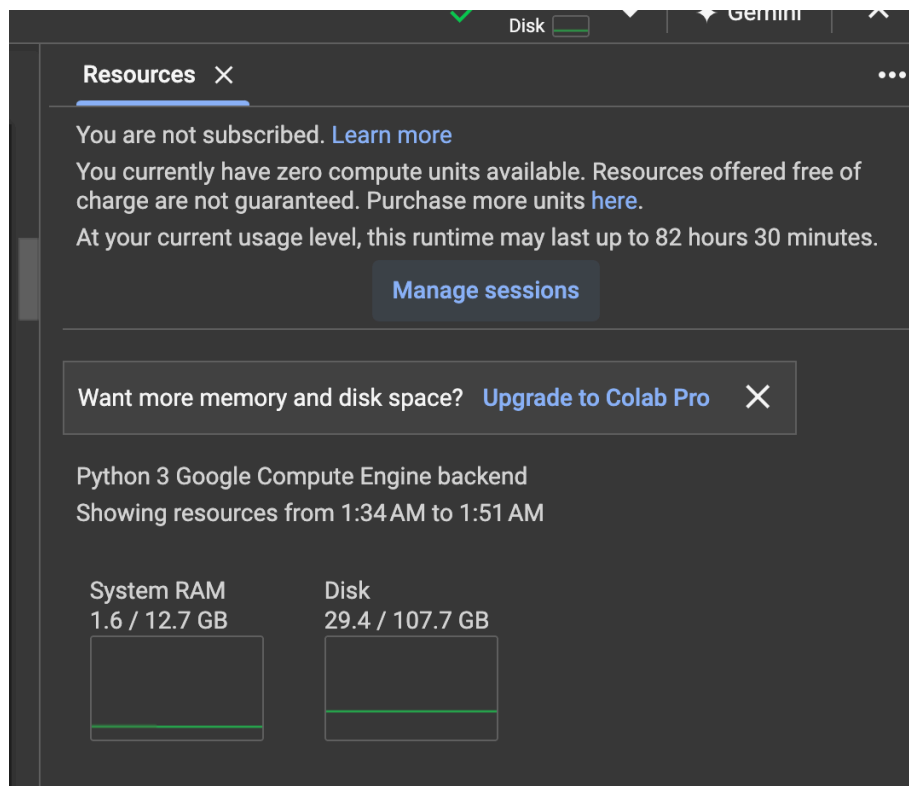
A Similar implied volatility arbitrage strategy is created for TSLA and its basket prediction. The graph below illustrates the buy and sell points for the call option on TSLA. An investor can go long when the implied volatility of TSLA is lower than its basket stock options. The mean reverting characteristic will pull up the implied volatility; hence, it will in turn increase the call option price and yield a profit.



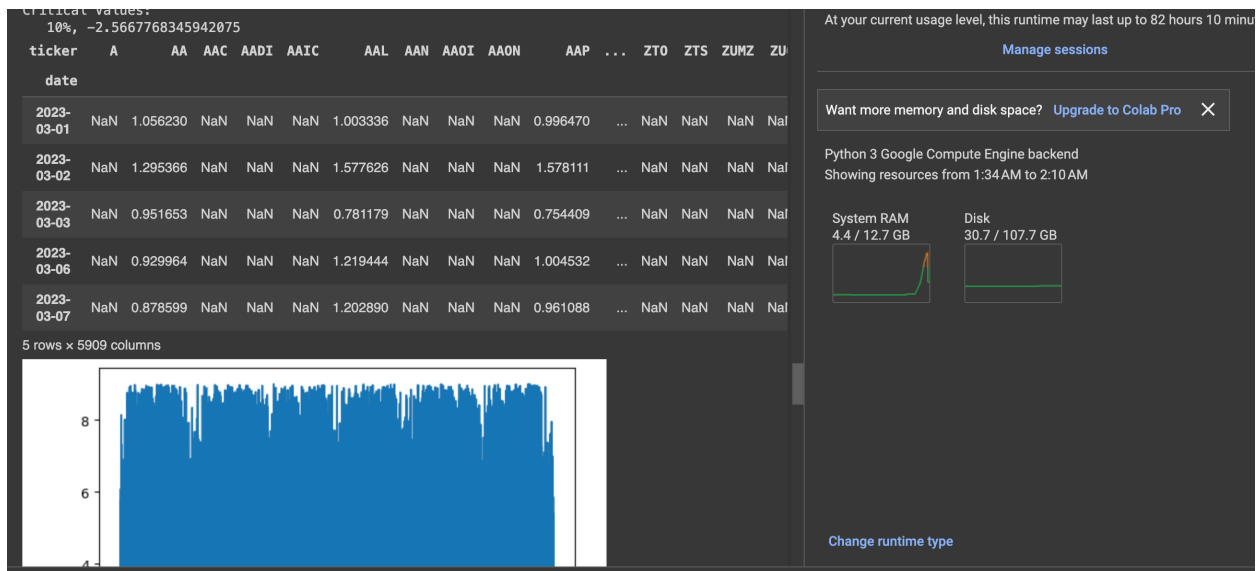
Chapter 6

Scope Limitation

The project led to a lot of challenges with the data processing. The major challenge was the data size. The first dataset that I downloaded from WRDS had the entire options price data for 3 years. The data size was over 30 GB and Google Colab crashed as the data was getting ingested into the application. Therefore, I reduced the scope to 1 year of options data for all stocks. The data was 13 GB in size and brought the Google Colab application to its edge of operation. The only option left was to upgrade to Colab pro for further analysis.



Therefore, the scope was further reduced to focus on option data that was 7 days to expiry as most of the volatility happened closer to the option expiry. Furthermore, the number of years was reduced to 6 months. The compute power still struggled and the RAM spike usage can be observed below:

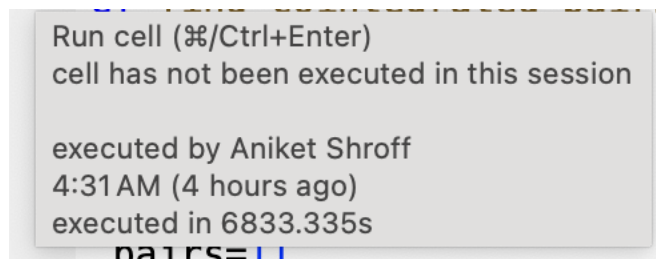


The 6 month data was prepared for doing a cointegration test which took over 4 hours to execute. Below are the stocks observed followed by the cointegration results for all the stocks for 6 months of option data.

['AA', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABNB', 'ABT', 'ACB', 'ACN', 'ADBE', 'ADI', 'ADM', 'ADP', 'ADSK', 'AEO', 'AFL', 'AFRM', 'AG', 'AGNC', 'AI', 'AIG', 'ALB', 'ALGN', 'ALLY', 'AMAT', 'AMBA', 'AMC', 'AMD', 'AMGN', 'AMPL', 'AMPX', 'AMRSQ', 'AMZN', 'ANET', 'ANF', 'APA', 'APO', 'APPHQ', 'APPS', 'APRN', 'APT', 'AR', 'ARDX', 'ARKG', 'ARKK', 'ASAN', 'ASHR', 'ASML', 'ASO', 'ASTS', 'ATER', 'ATVI', 'AUY', 'AVGO', 'AVXL', 'AXP', 'AXSM', 'AZN', 'AZO', 'BA', 'BABA', 'BAC', 'BAX', 'BB', 'BBAI', 'BBBYQ', 'BBIG', 'BBWI', 'BBY', 'BDX', 'BEKE', 'BHC', 'BIDU', 'BIIB', 'BILI', 'BILL', 'BITO', 'BK', 'BKKT', 'BKLN', 'BKNG', 'BLK', 'BLNK', 'BMY', 'BNTX', 'BOIL', 'BP', 'BRK', 'BROS', 'BTU', 'BUD', 'BURL', 'BX', 'BYND', 'BYON', 'BZFD', 'C', 'CAG', 'CAH', 'CANO', 'CAR', 'CAT', 'CBOE', 'CC', 'CCJ', 'CCL', 'CEI', 'CELH', 'CF', 'CGC', 'CHPT', 'CHTR', 'CHWY', 'CI', 'CLF', 'CLOV', 'CLX', 'CMCSA', 'CME', 'CMG', 'CNC', 'CODX', 'COF', 'COIN', 'COP', 'COR', 'COST', 'COTY', 'CPB', 'CPNG', 'CRM', 'CRON', 'CROX', 'CRSP', 'CRWD', 'CS', 'CSCO', 'CSIQ', 'CSTM', 'CSX', 'CTRA', 'CVNA', 'CVS', 'CVX', 'CWH', 'CZR', 'DAL', 'DASH', 'DB', 'DBX', 'DD', 'DDD', 'DDOG', 'DE', 'DELL', 'DFS', 'DG', 'DHI', 'DHR', 'DIA', 'DIS', 'DISH', 'DKNG', 'DKS', 'DLO', 'DLTR', 'DNMR', 'DOCU', 'DOW', 'DPZ', 'DT', 'DUST', 'DVN', 'DWAC', 'EA', 'EBAY', 'EDIT', 'EDU', 'EEM', 'EFA', 'ELV', 'EMB', 'EMR', 'ENPH', 'ENVX', 'EOG', 'EPD', 'EQT', 'ERX', 'ET', 'ETSY', 'EVTL', 'EW', 'EWC', 'EWG', 'EWI', 'EWU', 'EWY', 'EWZ', 'EXAS', 'EXPE', 'EXPR', 'F', 'FAS', 'FAZ', 'FAZE', 'FCEL', 'FCX', 'FDX', 'FEZ', 'FFIE', 'FI', 'FIVE', 'FL', 'FOXA', 'FSLR', 'FSLY', 'FSR', 'FTCHF', 'FUBO', 'FUTU', 'FXE', 'FXI', 'GD', 'GDX', 'GDXJ', 'GE', 'GEHC', 'GH', 'GILD', 'GLD', 'GLW', 'GM', 'GME', 'GNRC', 'GNS', 'GOEV', 'GOLD', 'GOOG', 'GOOGL', 'GOOS', 'GOTU', 'GPRO', 'GPS', 'GRWG', 'GS', 'GSAT', 'GSK', 'GT', 'HAL', 'HBI', 'HD', 'HES', 'HIG', 'HL', 'HLF', 'HOG', 'HON', 'HOOD', 'HPQ', 'HRL', 'HSBC', 'HSY', 'HUBS', 'HUM', 'HUT', 'HYG', 'HZNP', 'IBB', 'IBM', 'ICLN', 'IEF', 'ILMN', 'IMPP', 'INDA', 'INO', 'INTC', 'INTU', 'IP', 'IQ', 'ISRG', 'ITB', 'ITW', 'IVR', 'IVV', 'IWM', 'TYR', 'JBLU', 'JD', 'JDST', 'JETS', 'JKS', 'JMIA', 'JNJ', 'JNPR', 'JNUG', 'JPM', 'JWN', 'KGC', 'KHC', 'KKR', 'KLAC', 'KMB', 'KMI', 'KMX', 'KO', 'KOLD', 'KR', 'KRE', 'KSS', 'KWEB', 'LAAC', 'LABD', 'LABU', 'LAZR', 'LCID', 'LEN', 'LI', 'LITE', 'LL', 'LLY', 'LMND', 'LMT', 'LNG', 'LOW', 'LQD', 'LRCX', 'LULU', 'LUMN', 'LUV', 'LVS', 'LYFT', 'M', 'MA', 'MANU', 'MAR', 'MARA', 'MCD', 'MCK', 'MDB', 'MDGL', 'MDLZ', 'MDT', 'MELI', 'MET', 'META', 'MGM', 'MJ', 'MLCO', 'MMAT', 'MMM', 'MNKD', 'MO', 'MOMO', 'MOS', 'MPC', 'MPW', 'MRK', 'MRNA', 'MRO', 'MRVL', 'MS', 'MSFT', 'MSOS', 'MSTR', 'MT', 'MTCH', 'MU', 'MULN', 'MVIS', 'NCLH', 'NEGG', 'NEM', 'NEOG', 'NET', 'NFLX', 'NIO', 'NKE', 'NKLA', 'NLY', 'NNDM', 'NNOX', 'NOC', 'NOK', 'NOV', 'NOW', 'NRGV', 'NSC', 'NTES', 'NTR', 'NU', 'NUE', 'NUGT', 'NVAX', 'NVDA', 'NXPI', 'OCGN', 'OIH', 'OKTA', 'OLED', 'OLN', 'ON', 'OPEN', 'ORCL', 'OXY', 'PAA', 'PACB', 'PANW', 'PARA', 'PBR', 'PCG', 'PDD', 'PENN', 'PEP', 'PFE', 'PG', 'PHUN', 'PINS', 'PLTR',

'PLUG', 'PM', 'PNC', 'PPG', 'PRVB', 'PSFE', 'PSNY', 'PSX', 'PTON', 'PXD', 'PYPL', 'QCOM', 'QQQ', 'QS', 'RACE', 'RADCQ', 'RBLX', 'RCL', 'RDFN', 'RDW', 'REGN', 'RH', 'RIDEQ', 'RIG', 'RIOT', 'RIVN', 'RKT', 'RNA', 'RNG', 'ROKU', 'ROST', 'RRC', 'RTX', 'RUM', 'RUN', 'SABR', 'SARK', 'SAVA', 'SAVE', 'SBLK', 'SBUX', 'SCHW', 'SDCCQ', 'SDOW', 'SDS', 'SE', 'SEDG', 'SFIX', 'SHAK', 'SHEL', 'SHOP', 'SICP', 'SIG', 'SILJ', 'SIRI', 'SKLZ', 'SKX', 'SLB', 'SLV', 'SMH', 'SMMT', 'SNAP', 'SNDL', 'SNOW', 'SNV', 'SO', 'SOFI', 'SOLO', 'SONO', 'SONY', 'SOUN', 'SOXL', 'SOXS', 'SPCE', 'SPGI', 'SPLK', 'SPOT', 'SPWR', 'SPXL', 'SPXS', 'SPXU', 'SPY', 'SQ', 'SQQQ', 'SRPT', 'SSO', 'STEM', 'STNE', 'STNG', 'STX', 'STZ', 'SU', 'SVXY', 'SWKS', 'SWN', 'SYF', 'SYY', 'T', 'TAL', 'TAN', 'TBT', 'TDOC', 'TEAM', 'TECK', 'TELL', 'TEVA', 'TGT', 'TGTX', 'THC', 'TJX', 'TLRY', 'TLT', 'TME', 'TMF', 'TMO', 'TMUS', 'TNA', 'TOL', 'TOON', 'TOST', 'TPR', 'TQQQ', 'TRIP', 'TROW', 'TSCO', 'TSLA', 'TSLI', 'TSM', 'TTCFQ', 'TTD', 'TTWO', 'TWLO', 'TXN', 'TZA', 'U', 'UAA', 'UAL', 'UBER', 'UCO', 'ULTA', 'UNG', 'UNH', 'UNP', 'UPRO', 'UPS', 'UPST', 'URA', 'URBN', 'URI', 'USB', 'USO', 'UUP', 'UVIX', 'UVXY', 'UWMC', 'V', 'VALE', 'VERU', 'VFC', 'VIXY', 'VLO', 'VOD', 'VRTX', 'VTRS', 'VXRT', 'VXX', 'VYX', 'VZ', 'W', 'WB', 'WBA', 'WBD', 'WDAY', 'WDC', 'WEAT', 'WEWKQ', 'WFC', 'WHR', 'WISH', 'WKHS', 'WM', 'WMB', 'WMT', 'WPM', 'WSM', 'WYNN', 'X', 'XBI', 'XHB', 'XLB', 'XLC', 'XLE', 'XLF', 'XLI', 'XLK', 'XLP', 'XLU', 'XLV', 'XLY', 'XME', 'XOM', 'XOP', 'XPEV', 'XRT', 'YETI', 'YINN', 'YPF', 'YY', 'Z', 'ZIM', 'ZM', 'ZS']

The stock option cointegration heat map led to a permutation and combination of 630! Which took the machine 4 hours to process:

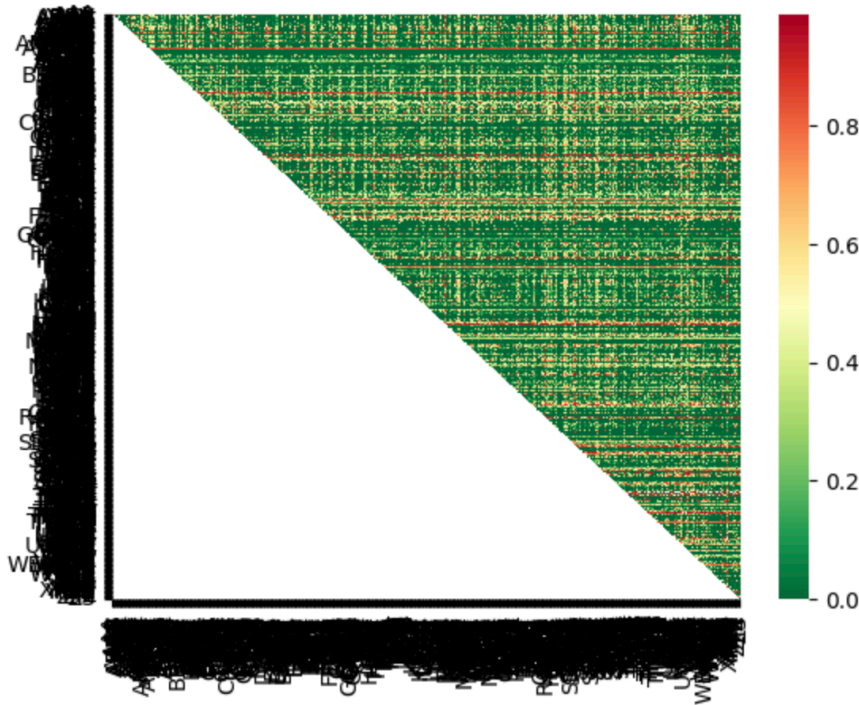


The heatmap was a complete pair trading strategy for a team to work on. The scope of this analysis could be useful for a hedge fund that has multiple team members focusing on different sectors. Each team could be given 100 pairs of options to focus their efforts on. Furthermore, stronger computing power and cloud storage will be required to have the processing be done quicker. The cointegrated pairs for the entire stock options are below:

```

[('AA', 'AAL'), ('AA', 'AAPL'), ('AA', 'ABBV'), ('AA', 'ABNB'), ('AA', 'ABT'), ('AA', 'A
[('AA', 'AAL'), ('AA', 'AAPL'), ('AA', 'ABBV'), ('AA', 'ABNB'), ('AA', 'ABT'), ('AA', 'A

```



There are over 4000 pairs that can be traded. Below are some options that the computer printed out to have a p-value less than 0.05, deeming the pair to be cointegrated with significance.

```

[('AA', 'AAL'), ('AA', 'AAPL'), ('AA', 'ABBV'), ('AA', 'ABNB'), ('AA', 'ABT'), ('AA', 'ACB')

```

The above dataset was out of scope for our research; hence, a query of 100 top hedge fund stock picks was taken from <https://hedgefollow.com/stocks>. The 100 stock option data was then retrieved from WRDS for a 6-month time frame. This allowed the dataset to be manageable for further processing. A cointegration test was conducted on certain sectors within the top 100 stock options. Chapter 2 of this paper highlights the results of these tests.

Finally, this paper doesn't take into consideration the transaction cost of trading options. Continuous buy and sell signals could increase the transaction cost significantly. Therefore, a machine learning algorithm can be developed to account for the transaction cost of each trade.

Conclusion

Volatility arbitrage trading is a great strategy for a portfolio manager. The cointegration of stock options implied volatility can provide a lot of information regarding the movement of a particular stock option volatility. The mismatched volatility with stationarity of data creates a golden opportunity for a trader to trade that arbitrage. An investor can use multilinear regression to generate the prediction of the basket volatility. The volatility arbitrage between the stock option implied volatility and the baskets' predicted volatility can create a lucrative trading opportunity. In our research, we focused on trading the implied volatility for TSLA and NVDA. Numerous

volatility arbitrage trading opportunities can be worked on for future projects. The methodology for developing future vol-arb strategies is very similar to the one employed in this research paper.

Appendix

The OptionMetric manual is available to WRDS account holders and OptionMetric clients. The direct link to the manual is https://wrds-www.wharton.upenn.edu/documents/755/IvyDB_US_Reference_Manual.pdf

1. Option_Price File

The Options Price file from OptionMetrics contains the historical price, implied volatility, and sensitivity information for the options on an underlying security.

Field descriptions

- a) Security ID = The Security ID for the underlying security.
- b) Date = The date of this price.
- c) Symbol = The option symbol.
- d) Strike = The strike price of the option times 1000.
- e) Expiration = The expiration date of the option.
- f) Call/Put = C or P, where C is Call, P is Put.
- g) Best Bid = The best, or highest, 15:59 EST bid price across all exchanges on which the option trades.
- h) Best Offer = The best, or lowest, 15:59 EST ask price across all exchanges on which the option
- i) Special Settlement = 0 or 1 or E.
 - I. 0 - The option has a standard settlement (100 shares of underlying security are to be delivered at exercise; the strike price and premium multipliers are \$ 100 per tick).
 - II. 1 - The option has a non-standard settlement. The number of shares to be delivered may be different from 100 (fractional shares); additional securities and/or cash may be required, and the strike price and premium multipliers may be different than \$ 100 per tick.
 - III. E - The option has a non-standard expiration date. This is usually due to an error in the historical data, which has not yet been researched and fixed.
- j) Option ID = Option ID is a unique integer identifier for the options contract. This identifier can be used to track specific options contracts over time.

References

- ALMEIDA, J. P. D. (2014). *Option-implied information and return prediction* (dissertation).
- Daniels, P. S. (2022). *Machine Learning Techniques for Pricing, Hedging and Statistical Arbitrage in Finance*(dissertation).
- Diavatopoulos, C. (2008). (dissertation). *Two essays on the predictive ability of implied volatility*.
- Dobi, D. (2014). *Modeling Volatility Risk in Equity Options: a Cross-Sectional Approach* (dissertation).
- Fernandes, A. J. F. G. (2020). *Betting Against Beta strategies using option-implied correlations* (dissertation).
- Metrics, O. (2024, July 11). OptionMetrics Option Pricing Database. Retrieved July 27, 2024,.
- Nogueira, V. H. de A. e. (2017). *Momentum strategies using option-implied correlations* (dissertation).
- Ulrich, M., Zimmer, L., & Merbecks, C. (2023). Implied volatility surfaces: A comprehensive analysis using half a billion option prices. *Review of Derivatives Research*, 26(2–3), 135–169. <https://doi.org/10.1007/s11147-023-09195-5>

CODE

✓ Executive Summary

This google colab document was created by Aniket Shroff for his Research on Volatility Arbitrage Trading. The focus is on the tech sector. The document follows a chronological order of data exploration, processing and analyzing. Please refer to the research paper for further clarification

✓ Ingesting Data

✓ Import Package

The panda library, numpy, sklearn and other important libraries are imported in order to provide boiler plate code

```
import yfinance as yf
import pandas as pd
import numpy as np
import csv
from matplotlib import pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import coint, adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm
from statsmodels import regression
from statsmodels.tsa.stattools import adfuller
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR

import seaborn as sns
from IPython.display import display

import scipy
from scipy.cluster import hierarchy
from scipy.spatial import distance

from sklearn.preprocessing import StandardScaler, Normalizer

from google.colab import drive
```

✓ Import Option Pricing Data - Google Drive

There are 3 datasets that have been imported for various analysis in this research. The size of the data limited the research scope to 6 month option trading data from the WRDS dataset. There are some parts of this colab that has data analysis done on 1 year data. The notes will highlight those results and code

```
#Mounting Google Drive
drive.mount('/content/drive')

#Define Data types for the Columns in the dataset
dtype = {
    'secid': 'float64',
    'symbol': 'object',
    'symbol_flag': 'float64',
    'strike': 'float64',
    'cp_flag': 'object',
    'strike_price': 'float64',
    'last_price': 'float64',
    'best_bid': 'float64',
    'best_offer': 'float64',
    'volume': 'float64',
    'open_interest': 'float64',
    'impl_volatility': 'float64',
    'delta': 'float64',
    'gamma': 'float64',
    'vega': 'float64',
    'theta': 'float64',
    'optionid': 'float64',
    'cfadj': 'float64',
    'am_settlement': 'float64',
    'contract_size': 'float64',
    'ss_flag': 'object',
    'forward_price': 'float64',
    'expiry_indicator': 'object',
    'root': 'object',
    'suffix': 'object',
    'cusip': 'object',
    'ticker': 'object',
    'sic': 'float64',
    'index_flag': 'object',
    'exchange_d': 'object',
    'class': 'object',
    'issue_type': 'object',
    'industry_group': 'object',
    'issuer': 'object',
    'div_convention': 'object',
    'exercise_style': 'object',
    'am_set_flag': 'object',
}


#define the date column for parsing
date_columns = ['date', 'exdate', 'last_date']

#Load the CSV file from google drive
df = pd.read_csv('/content/drive/My Drive/fiveyearmlr.csv', parse_dates=date_columns, dtype=dtype, low_memory=False)

#Calculate the number of days left
df['daysLeft'] = (df['exdate'] - df['date']).dt.days

#Set the date column as the index
df.set_index('date', inplace=True)

#Display the first 60 dataset
df.head(60)
```

 Mounted at /content/drive

	secid	symbol	symbol_flag	exdate	last_date	cp_flag	strike_price	best_bid	best_offer	volume	..
date											
2018-01-02	101121.0	AMD 180105C10000	1.0	2018-01-05	2018-01-02	C	10000.0	1.00	1.02	583.0	.
2018-01-02	101121.0	AMD 180105C10500	1.0	2018-01-05	2018-01-02	C	10500.0	0.52	0.54	6660.0	.
2018-01-02	101121.0	AMD 180105C11000	1.0	2018-01-05	2018-01-02	C	11000.0	0.17	0.18	12851.0	.
2018-01-02	101121.0	AMD 180105C11500	1.0	2018-01-05	2018-01-02	C	11500.0	0.03	0.04	2928.0	.
2018-01-02	101121.0	AMD 180105C12000	1.0	2018-01-05	2018-01-02	C	12000.0	0.00	0.02	348.0	.
2018-01-02	101121.0	AMD 180105C12500	1.0	2018-01-05	2018-01-02	C	12500.0	0.00	0.01	274.0	.
2018-01-02	101121.0	AMD 180105C13000	1.0	2018-01-05	2018-01-02	C	13000.0	0.00	0.01	10.0	.
2018-01-02	101121.0	AMD 180105C13500	1.0	2018-01-05	2017-12-07	C	13500.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C14000	1.0	2018-01-05	2017-12-06	C	14000.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C14500	1.0	2018-01-05	2017-12-04	C	14500.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C15000	1.0	2018-01-05	NaT	C	15000.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C15500	1.0	2018-01-05	2017-12-04	C	15500.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C16000	1.0	2018-01-05	NaT	C	16000.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C16500	1.0	2018-01-05	NaT	C	16500.0	0.00	0.02	0.0	.
2018-01-02	101121.0	AMD 180105C5000	1.0	2018-01-05	NaT	C	5000.0	5.95	6.05	0.0	.

2018-01-02	101121.0	AMD 180105C5500	1.0	2018-01-05	NaT	C	5500.0	5.45	5.55	0.0	.
2018-01-02	101121.0	AMD 180105C6500	1.0	2018-01-05	2017-12-04	C	6500.0	4.45	4.55	0.0	.
2018-01-02	101121.0	AMD 180105C7000	1.0	2018-01-05	2018-01-02	C	7000.0	3.95	4.05	2.0	.
2018-01-02	101121.0	AMD 180105C7500	1.0	2018-01-05	2017-12-13	C	7500.0	3.45	3.55	0.0	.
2018-01-02	101121.0	AMD 180105C8000	1.0	2018-01-05	2017-12-26	C	8000.0	2.97	3.05	0.0	.
2018-01-02	101121.0	AMD 180105C8500	1.0	2018-01-05	2017-12-27	C	8500.0	2.49	2.51	0.0	.
2018-01-02	101121.0	AMD 180105C9000	1.0	2018-01-05	2018-01-02	C	9000.0	1.99	2.01	270.0	.
2018-01-02	101121.0	AMD 180105C9500	1.0	2018-01-05	2018-01-02	C	9500.0	1.49	1.51	210.0	.
2018-01-03	101121.0	AMD 180105C10000	1.0	2018-01-05	2018-01-03	C	10000.0	1.55	1.57	506.0	.
2018-01-03	101121.0	AMD 180105C10500	1.0	2018-01-05	2018-01-03	C	10500.0	1.06	1.08	1882.0	.
2018-01-03	101121.0	AMD 180105C11000	1.0	2018-01-05	2018-01-03	C	11000.0	0.62	0.64	13000.0	.
2018-01-03	101121.0	AMD 180105C11500	1.0	2018-01-05	2018-01-03	C	11500.0	0.29	0.30	15048.0	.
2018-01-03	101121.0	AMD 180105C12000	1.0	2018-01-05	2018-01-03	C	12000.0	0.11	0.12	44560.0	.
2018-01-03	101121.0	AMD 180105C12500	1.0	2018-01-05	2018-01-03	C	12500.0	0.04	0.05	11080.0	.
2018-01-03	101121.0	AMD 180105C13000	1.0	2018-01-05	2018-01-03	C	13000.0	0.01	0.03	2903.0	.
2018-01-03	101121.0	AMD 180105C13500	1.0	2018-01-05	2018-01-03	C	13500.0	0.00	0.02	144.0	.

2018-01-03	101121.0	AMD 180105C14000	1.0	2018-01-05	2018-01-03	C	14000.0	0.00	0.02	138.0	.
2018-01-03	101121.0	AMD 180105C14500	1.0	2018-01-05	2018-01-03	C	14500.0	0.00	0.01	111.0	.
2018-01-03	101121.0	AMD 180105C15000	1.0	2018-01-05	2018-01-03	C	15000.0	0.00	0.01	455.0	.
2018-01-03	101121.0	AMD 180105C15500	1.0	2018-01-05	2018-01-03	C	15500.0	0.00	0.01	220.0	.
2018-01-03	101121.0	AMD 180105C16000	1.0	2018-01-05	NaT	C	16000.0	0.00	0.01	0.0	.
2018-01-03	101121.0	AMD 180105C16500	1.0	2018-01-05	NaT	C	16500.0	0.00	0.01	0.0	.
2018-01-03	101121.0	AMD 180105C5000	1.0	2018-01-05	NaT	C	5000.0	6.50	6.65	0.0	.
2018-01-03	101121.0	AMD 180105C5500	1.0	2018-01-05	NaT	C	5500.0	6.00	6.15	0.0	.
2018-01-03	101121.0	AMD 180105C6500	1.0	2018-01-05	2018-01-03	C	6500.0	5.00	5.10	4.0	.
2018-01-03	101121.0	AMD 180105C7000	1.0	2018-01-05	2018-01-02	C	7000.0	4.50	4.60	0.0	.
2018-01-03	101121.0	AMD 180105C7500	1.0	2018-01-05	2017-12-13	C	7500.0	4.00	4.10	0.0	.
2018-01-03	101121.0	AMD 180105C8000	1.0	2018-01-05	2017-12-26	C	8000.0	3.50	3.60	0.0	.
2018-01-03	101121.0	AMD 180105C8500	1.0	2018-01-05	2018-01-03	C	8500.0	3.00	3.10	4.0	.
2018-01-03	101121.0	AMD 180105C9000	1.0	2018-01-05	2018-01-03	C	9000.0	2.55	2.57	434.0	.
2018-01-03	101121.0	AMD 180105C9500	1.0	2018-01-05	2018-01-03	C	9500.0	2.04	2.07	186.0	.
2018-01-04	101121.0	AMD 180105C10000	1.0	2018-01-05	2018-01-04	C	10000.0	2.09	2.12	271.0	.
2018-01-04	101121.0	AMD 180105C10500	1.0	2018-01-05	2018-01-04	C	10500.0	1.59	1.61	733.0	.

01-04	180105C10500	01-05										
2018-01-04	101121.0 AMD 180105C11000	1.0	2018-01-05	2018-01-04	C	11000.0	1.10	1.12	2364.0	.		
2018-01-04	101121.0 AMD 180105C11500	1.0	2018-01-05	2018-01-04	C	11500.0	0.61	0.63	2788.0	.		
2018-01-04	101121.0 AMD 180105C12000	1.0	2018-01-05	2018-01-04	C	12000.0	0.21	0.22	14766.0	.		
2018-01-04	101121.0 AMD 180105C12500	1.0	2018-01-05	2018-01-04	C	12500.0	0.05	0.06	16485.0	.		
2018-01-04	101121.0 AMD 180105C13000	1.0	2018-01-05	2018-01-04	C	13000.0	0.01	0.02	5073.0	.		
2018-01-04	101121.0 AMD 180105C13500	1.0	2018-01-05	2018-01-04	C	13500.0	0.00	0.02	10.0	.		
2018-01-04	101121.0 AMD 180105C14000	1.0	2018-01-05	2018-01-04	C	14000.0	0.00	0.01	1.0	.		
2018-01-04	101121.0 AMD 180105C14500	1.0	2018-01-05	2018-01-04	C	14500.0	0.00	0.02	1.0	.		
2018-01-04	101121.0 AMD 180105C15000	1.0	2018-01-05	2018-01-03	C	15000.0	0.00	0.02	0.0	.		
2018-01-04	101121.0 AMD 180105C15500	1.0	2018-01-05	2018-01-03	C	15500.0	0.00	0.02	0.0	.		
2018-01-04	101121.0 AMD 180105C16000	1.0	2018-01-05	NaT	C	16000.0	0.00	0.02	0.0	.		
2018-01-04	101121.0 AMD 180105C16500	1.0	2018-01-05	NaT	C	16500.0	0.00	0.02	0.0	.		

60 rows x 38 columns

Understanding the Raw Data

The raw dataset was very confusing. Therefore, various graphs were plotted. AAPL was utilized as an example option for plotting

Histogram Option plotting for AAPL

Plotted AAPL options at different option strike prices. The results illustrate that majority of options are issued closer to the strike price. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset

✓ Bid-Ask spread for AAPL Call Option

The bid ask spread for different strike price is plotted for AAPL call options. The price 1 day before expiration is used for plotting the graph. The graph illustrates the general price movement of AAPL options and also shows signs of liquidity as the bid ask spread is very tight. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset

```
## Filtering for Call Option at 100 and AAPL
# Filter the DataFrame for rows where the issuer is "APPLE INC"
AAPL = df[df["issuer"] == "APPLE INC"]

## Shows Main Strike Price for AAPL
plt.hist(AAPL["strike_price"])
plt.show()

## Creates subdataset for different strike prices
AAPL100 = AAPL[AAPL["strike_price"] == 100000.0]
AAPL120 = AAPL[AAPL["strike_price"] == 120000.0]
AAPL140 = AAPL[AAPL["strike_price"] == 140000.0]
AAPL160 = AAPL[AAPL["strike_price"] == 160000.0]
AAPL180 = AAPL[AAPL["strike_price"] == 180000.0]
AAPL200 = AAPL[AAPL["strike_price"] == 200000.0]

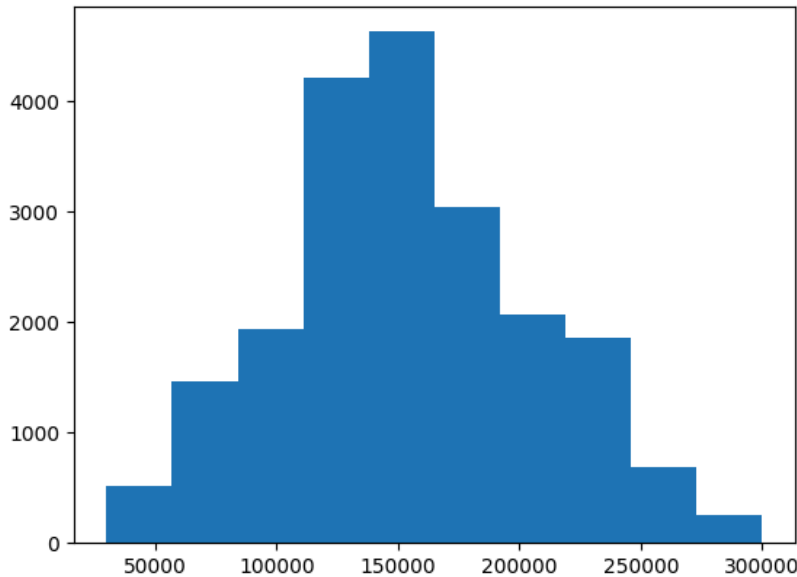
## Further filters by options 1 day before expiry behaviour
AAPL100DAY1= AAPL100[AAPL["daysLeft"] == 1]
AAPL120DAY1= AAPL120[AAPL["daysLeft"] == 1]
AAPL140DAY1= AAPL140[AAPL["daysLeft"] == 1]
AAPL160DAY1= AAPL160[AAPL["daysLeft"] == 1]
AAPL180DAY1= AAPL180[AAPL["daysLeft"] == 1]
AAPL200DAY1= AAPL200[AAPL["daysLeft"] == 1]

#@ Display the first few rows of the filtered DataFrame
AAPL100DAY1.head()

## Plots the Bid and Ask over time for different Strike Prices.
plt.plot(AAPL100DAY1['date'], AAPL100DAY1['best_bid'], alpha=0.5, label='Best Bid', color='red')
plt.plot(AAPL100DAY1['date'], AAPL100DAY1['best_offer'], alpha=0.5, label='Best Offer', color='orange')
plt.plot(AAPL120DAY1['date'], AAPL120DAY1['best_bid'], alpha=0.5, label='Best Bid', color='yellow')
plt.plot(AAPL120DAY1['date'], AAPL120DAY1['best_offer'], alpha=0.5, label='Best Offer', color='green')
plt.plot(AAPL140DAY1['date'], AAPL140DAY1['best_bid'], alpha=0.5, label='Best Bid', color='blue')
plt.plot(AAPL140DAY1['date'], AAPL140DAY1['best_offer'], alpha=0.5, label='Best Offer', color='purple')
plt.plot(AAPL160DAY1['date'], AAPL160DAY1['best_bid'], alpha=0.5, label='Best Bid', color='pink')
plt.plot(AAPL160DAY1['date'], AAPL160DAY1['best_offer'], alpha=0.5, label='Best Offer', color='black')
plt.plot(AAPL180DAY1['date'], AAPL180DAY1['best_bid'], alpha=0.5, label='Best Bid', color='brown')
plt.plot(AAPL180DAY1['date'], AAPL180DAY1['best_offer'], alpha=0.5, label='Best Offer', color='gray')
plt.plot(AAPL200DAY1['date'], AAPL200DAY1['best_bid'], alpha=0.5, label='Best Bid', color='cyan')
plt.plot(AAPL200DAY1['date'], AAPL200DAY1['best_offer'], alpha=0.5, label='Best Offer', color='magenta')

##Plots the Legen
plt.legend()

## Shows the plotted graph
plt.show()
```



```

<ipython-input-4-4123d8a1e438>:19: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
AAPL100DAY1= AAPL100[AAPL["daysLeft"] == 1]
<ipython-input-4-4123d8a1e438>:20: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
AAPL120DAY1= AAPL120[AAPL["daysLeft"] == 1]
<ipython-input-4-4123d8a1e438>:21: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
AAPL140DAY1= AAPL140[AAPL["daysLeft"] == 1]
<ipython-input-4-4123d8a1e438>:22: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
AAPL160DAY1= AAPL160[AAPL["daysLeft"] == 1]
<ipython-input-4-4123d8a1e438>:23: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
AAPL180DAY1= AAPL180[AAPL["daysLeft"] == 1]
<ipython-input-4-4123d8a1e438>:24: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
AAPL200DAY1= AAPL200[AAPL["daysLeft"] == 1]

```



Option Volume in the final week before expiry

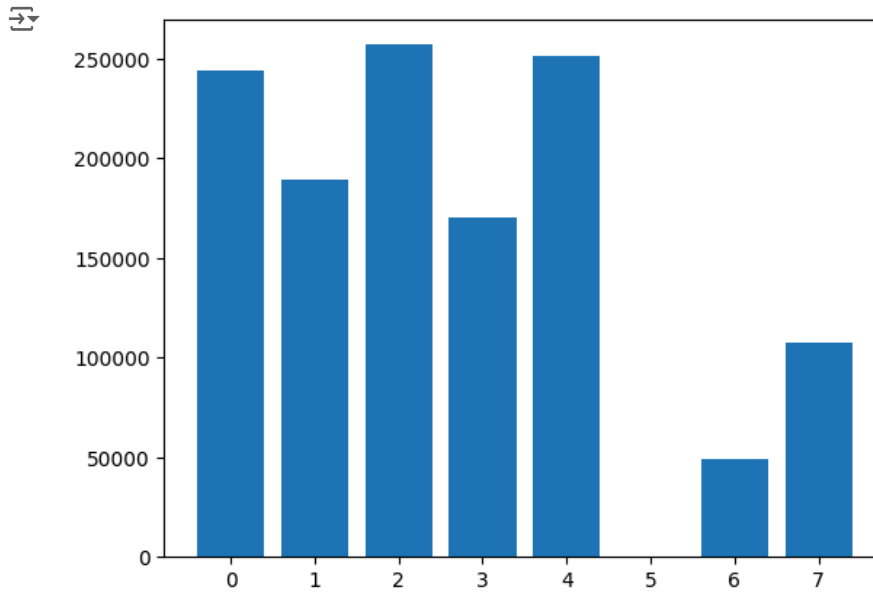
The graph below shows the option trading volume for AAPL 1 week before expiry of the option. This illustrates the trading pattern where the volume picks up in the final week of the option being traded. Day 5 is Sunday; hence, there is no trading activity on that day. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset

```
## Filtering for Call Option at 100 and AAPL  
# Filter the DataFrame for rows where the issuer is "APPLE INC"  
AAPL = df[df["issuer"] == "APPLE INC"]
```

```
plt.bar(AAPL['daysLeft'], AAPL['volume'])
```

```
plt.show()
```

```
## plt.show()
```



✓ Analyzing Implied Volatility

✓ AAPL average Implied Volatility graph

The implied volatility is averaged for different strike prices to generate a single implied volatility metric for a single day. The data is then graphed over time in order to visualize the implied volatility. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset

```

##Taking an average of all volatility for the day of a stock
AAPLgrouped = AAPL.groupby(['date','ticker'])['impl_volatility'].mean()

##Dropped all the Nan and infinite numbers
AAPLgrouped.replace([np.inf, -np.inf], np.nan, inplace=True)
AAPLgrouped.dropna(inplace=True)

##Reset the index
AAPLgrouped_df = AAPLgrouped.reset_index()

##ADF test for stationarity proved data is stationary
result = adfuller(AAPLgrouped)
print("ADF Statistic:", result[0])
print("p-value:", result[1])
for key, value in result[4].items():
    print('Critical Values:')
    print(f'    {key}, {value}')

##Plotted the implied volatility for AAPL
AAPLgrouped.plot()

##Reset the index
AAPLgrouped_df = AAPLgrouped.reset_index()

##Displayed the dataset to show one volatility number for each day
AAPLgrouped_df.head()

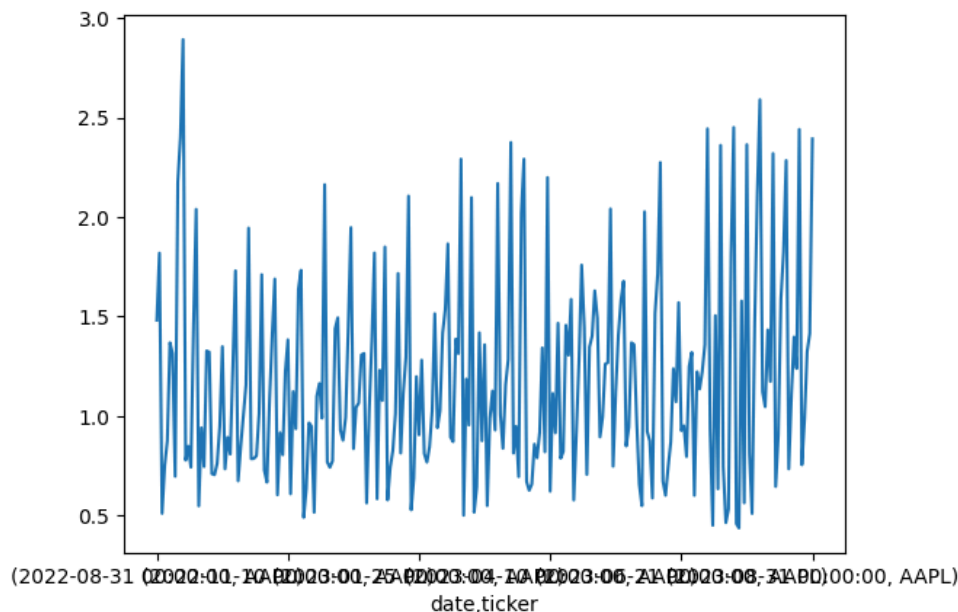
```

```

ADF Statistic: -3.8669317300500508
p-value: 0.0022921487399941787
Critical Values:
 1%, -3.4582467982399105
Critical Values:
 5%, -2.8738137461081323
Critical Values:
10%, -2.5733111490323846

```

	date	ticker	impl_volatility
0	2022-08-31	AAPL	1.481190
1	2022-09-01	AAPL	1.818609
2	2022-09-02	AAPL	0.510713
3	2022-09-06	AAPL	0.751199
4	2022-09-07	AAPL	0.875668



▼ Implied volatility for all stocks

The average implied volatility for all stock options was taken in order to get a dataset with implied volatilities indexed by date and ticker symbol. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset

```
##Taking an average of all volatility for the day of a stock
dfgrouped = df.groupby(['date','ticker'])['impl_volatility'].mean()

##Dropped all the Nan and infinite numbers
dfgrouped.replace([np.inf, -np.inf], np.nan, inplace=True)
dfgrouped.dropna(inplace=True)

##Reset the index
dfgrouped_df = dfgrouped.reset_index()

##The dataset is displayed
dfgrouped_df.head()

##Dropped all the Nan and infinite numbers
dfgrouped_df.replace([np.inf, -np.inf], np.nan, inplace=True)
dfgrouped_df.dropna(inplace=True)

##Dataset is displayed after dropping all Nan values
dfgrouped_df.head()

##the dataset is pivoted to have the columns be the ticker symbols
pivot_df = dfgrouped_df.pivot(index='date', columns='ticker', values='impl_volatility').reset_index(drop = True)

##Dropped column with Nan and did forward fill on the others as still facing Nan errors
pivot_df.fillna(method='ffill', inplace=True)
pivot_df.dropna(axis=1, how='any', inplace=True)

##Final Dataset is displayed below
pivot_df.head()
```

	ticker	AA	AAL	AAP	AAPL	ABBV	ABNB	ABT	ACB	ACN	ADBE	...	XPEV	XI
0		1.056230	1.003336	0.996470	0.875830	1.222613	1.194802	1.090021	4.192827	1.223037	1.190113	...	2.567583	0.8044
1		1.295366	1.577626	1.578111	1.358543	1.821039	1.012147	1.336068	5.830968	1.557699	1.184055	...	3.473922	0.8968
2		0.951653	0.781179	0.754409	0.549710	0.532687	0.500979	0.431204	2.322866	0.542045	0.435855	...	1.282976	0.3989
3		0.929964	1.219444	1.004532	0.989025	0.627974	0.579360	0.701597	3.004888	0.708661	0.483565	...	1.791460	0.6083
4		0.878599	1.202890	0.961088	1.125833	0.650075	0.871505	0.800695	4.513471	0.812730	0.732249	...	2.327566	0.6532

5 rows x 631 columns

Created a list of all tickers that existed in the large dataset.

```
## Getting a list of all tickers
tickers = pivot_df.columns.tolist()
print(tickers)
```

```
['AA', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABNB', 'ABT', 'ACB', 'ACN', 'ADBE', 'ADI', 'ADM', 'ADP', 'ADSK', 'AEO', 'AF
```

▼ Implied Volatility for top 100 hedge fund stocks

The average implied volatility for the top 100 stocks according to hedgeflow.com was taken to get a dataset with implied volatilities indexed by date and ticker symbol. The code is repeated in order to preserve the result on the 1 year dataset observed above. This results were

plotted from the 6 month dataset from 2023 March to 2023 August WRDS Dataset

```
##Taking an average of all volatility for the day of a stock
dfgrouped = df.groupby(['date','ticker'])['impl_volatility'].mean()

##Dropped all the Nan and infinite numbers
dfgrouped.replace([np.inf, -np.inf], np.nan, inplace=True)
dfgrouped.dropna(inplace=True)

##Reset the index
dfgrouped_df = dfgrouped.reset_index()

##The dataset is displayed
dfgrouped_df.head()

##Dropped all the Nan and infinite numbers
dfgrouped_df.replace([np.inf, -np.inf], np.nan, inplace=True)
dfgrouped_df.dropna(inplace=True)

##the dataset is pivoted to have the columns be the ticker symbols
pivot_df = dfgrouped_df.pivot(index='date', columns='ticker', values='impl_volatility')

##Dropped column with Nan and did forward fill on the others as still facing Nan errors
pivot_df.fillna(method='ffill', inplace=True)
pivot_df.dropna(axis=1, how='any', inplace=True)

##Final Dataset is displayed below
pivot_df.head()
```

```
<ipython-input-26-4268a7571bb2>:22: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in
pivot_df.fillna(method='ffill', inplace=True)
```

ticker	AA	AAL	AAP	AAPL	ABBV	ABNB	ABT	ACB	ACN	ADBE	...	XPEV	XI
2023-03-01	1.056230	1.003336	0.996470	0.875830	1.222613	1.194802	1.090021	4.192827	1.223037	1.190113	...	2.567583	0.8044
2023-03-02	1.295366	1.577626	1.578111	1.358543	1.821039	1.012147	1.336068	5.830968	1.557699	1.184055	...	3.473922	0.8968
2023-03-03	1.295366	1.577626	1.578111	1.358543	1.821039	1.012147	1.336068	5.830968	1.557699	1.184055	...	3.473922	0.8968
2023-03-06	1.295366	1.577626	1.578111	1.358543	1.821039	1.012147	1.336068	5.830968	1.557699	1.184055	...	3.473922	0.8968
2023-03-07	0.878599	1.202890	0.961088	1.125833	0.650075	0.871505	0.800695	4.513471	0.812730	0.732249	...	2.327566	0.6532

5 rows x 631 columns

Created a list of all tickers that existed in the top 100 stocks dataset.

```
## Getting a list of all tickers in the dataset
tickers = pivot_df.columns.tolist()
print(tickers)
```

```
['AA', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABNB', 'ABT', 'ACB', 'ACN', 'ADBE', 'ADI', 'ADM', 'ADP', 'ADSK', 'AEO', 'AF
```

✓ Implied Volatility for top 12 stocks analyzed in model building

The average implied volatility for the 12 stocks identified from cointegration on the 6 month data was taken to get a dataset with implied volatilities indexed by date and ticker symbol. The code is repeated in order to preserve the result on the 5 year dataset observed above. This results were plotted for 6 years from 2018 January to 2023 August WRDS Dataset

```

##Taking an average of all volatility for the day of a stock
dfgrouped = df.groupby(['date','ticker'])['impl_volatility'].mean()

##Dropped all the Nan and infinite numbers
dfgrouped.replace([np.inf, -np.inf], np.nan, inplace=True)
dfgrouped.dropna(inplace=True)

##Reset the index
dfgrouped_df = dfgrouped.reset_index()

##The dataset is displayed
dfgrouped_df.head()


##Dropped all the Nan and infinite numbers
dfgrouped_df.replace([np.inf, -np.inf], np.nan, inplace=True)
dfgrouped_df.dropna(inplace=True)

##the dataset is pivoted to have the columns be the ticker symbols
pivot_df = dfgrouped_df.pivot(index='date', columns='ticker', values='impl_volatility')

##Dropped column with Nan and did forward fill on the others as still facing Nan errors
pivot_df.fillna(method='ffill', inplace=True)
pivot_df.dropna(axis=1, how='any', inplace=True)

##Final Dataset is displayed below
pivot_df.head()

```

 <ipython-input-3-4268a7571bb2>:22: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in pivot_df.fillna(method='ffill', inplace=True)

date	ADP	AMD	BA	GLD	GOOG	LRCX	MSFT	NKE	NOW	NVDA	PG	TSLA
2018-01-02	0.479378	1.597138	0.401352	0.245916	0.351948	0.715155	0.640792	0.462814	0.658003	0.497846	0.435184	0.738733
2018-01-03	0.695151	1.685501	0.474812	0.149064	0.505058	0.903597	0.714734	0.797993	0.807744	0.954948	0.674633	0.895001
2018-01-04	0.868398	1.565132	0.758432	0.493345	0.897195	1.449049	1.124877	0.918952	1.096466	0.745380	0.769397	0.909521
2018-01-05	0.348830	0.710411	0.274704	0.157816	0.320865	0.392045	0.450390	0.477209	0.290096	0.371140	0.335793	0.445483

∨ Cointegration

∨ Cointegration Testing on AAPL and MSFT

This testing is done to illustrate the cointegration between this 2 tech stocks. The p value is less than 0.05; hence the implied volatility is cointegrated for these 2 stocks. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset

```

AAPLlog = np.log(pivot_df['AAPL'])
MSFTlog = np.log(pivot_df['MSFT'])

score, pvalue, _ = coint(AAPLlog, MSFTlog)

if pvalue < 0.05:
    print("Reject the null hypothesis: Time series are cointegrated.")
    print(pvalue)
else:
    print("Fail to reject the null hypothesis: Time series are not cointegrated.")
    print(pvalue)

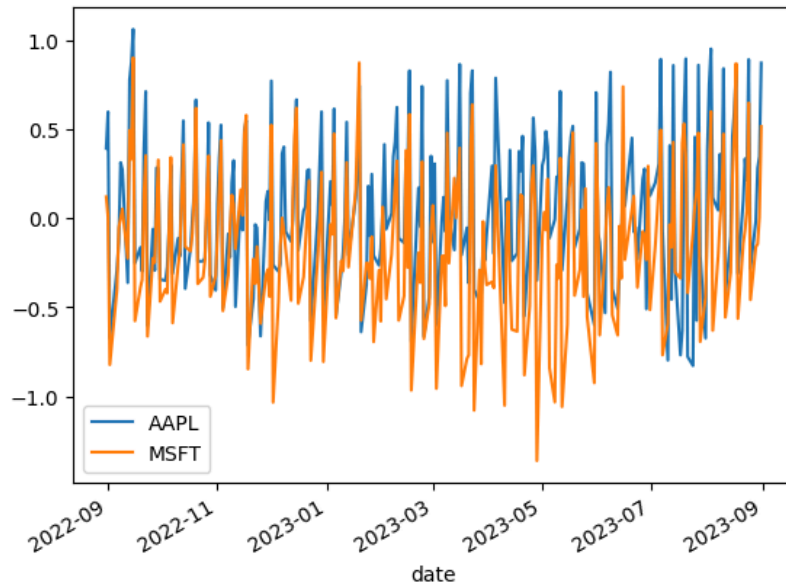
pd.concat([AAPLlog, MSFTlog], axis=1).plot()

```

```

↵ Reject the null hypothesis: Time series are cointegrated.
0.005875113903824956
<Axes: xlabel='date'>

```



∨ Cointegration of entire stocks in the dataset

The cointegration testing was conducted on the entire stock options implied volatility. There were over 630 stock which led to a 4 hour wait. The result is extremely dense and left here for visual purpose and future scope. This results were plotted from the 1 year dataset from 2022 August to 2023 August WRDS Dataset


```

# List of tickers to filter 10 stocks
tickers_to_filter = ['AAPL', 'ADBE', 'ADP', 'AMAT', 'AMD', 'AVGO', 'CRM', 'ETN', 'GE', 'GOOG', 'GOOGL', 'IBM', 'INTC',

# Filter columns where ticker is in the list of tickers
filtered_pivot_df = pivot_df.loc[:, pivot_df.columns.isin(tickers_to_filter)]

# Display the first few rows of the filtered DataFrame
filtered_pivot_df.head()

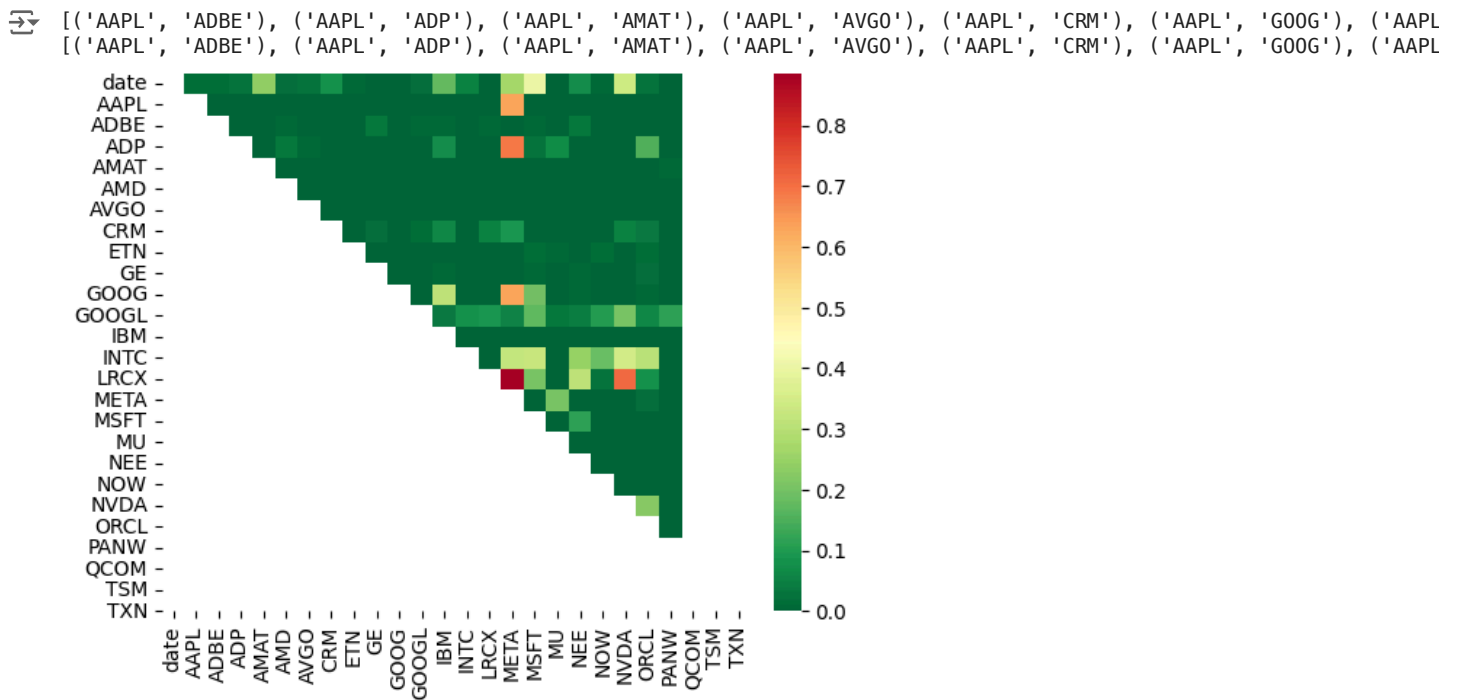
##Function to find cointegrated pairs for the tech stocks
def find_cointegrated_pairs(data):
    n=data.shape[1]
    score_matrix=np.zeros((n,n))
    pvalue_matrix=np.ones((n,n))
    keys=data.keys()
    pairs=[]
    for i in range(n):
        for j in range(i+1,n):
            s1=data[keys[i]]
            s2=data[keys[j]]
            result=coint(s1,s2)
            score, pvalue, _ = coint(s1, s2)
            score_matrix[i,j]=score
            pvalue_matrix[i,j]=pvalue
            if pvalue<0.05:
                pairs.append((keys[i],keys[j]))
    return score_matrix, pvalue_matrix, pairs

## assigning the tech tickers to the sybols list
symbol_list = tickers

##Printing the highly correlated pairs.
score_matrix, pvalue_matrix, pairs = find_cointegrated_pairs(filtered_pivot_df)
print(pairs)

##Printing the heatmap chart of the cointegrated tech stock.
scores, pvalues, pairs = find_cointegrated_pairs(filtered_pivot_df)
import seaborn
seaborn.heatmap(pvalues, xticklabels = tickers_to_filter, yticklabels = tickers_to_filter, cmap = 'RdYlGn_r'
                , mask = (pvalues >= 0.99)
                )
print(pairs)

```



✓ Cointegration on the top 100 hedge fund stock - 100 Stocks

The cointegration testing was applied on all the stock options implied volatility. There are 100 stock options implied plotted with a heatmap. This results were plotted from the 6 month dataset from 2023 March to 2023 August WRDS Dataset

```
# Display the first few rows of the filtered DataFrame
pivot_df.head()
```

```
##Function to find cointegrated pairs for the tech stocks
```

```
def find_cointegrated_pairs(data):
    n=data.shape[1]
    score_matrix=np.zeros((n,n))
    pvalue_matrix=np.ones((n,n))
    keys=data.keys()
    pairs=[]
    for i in range(n):
        for j in range(i+1,n):
            s1=data[keys[i]]
            s2=data[keys[j]]
            result=coint(s1,s2)
            score, pvalue, _ = coint(s1, s2)
            score_matrix[i,j]=score
            pvalue_matrix[i,j]=pvalue
            if pvalue<0.05:
                pairs.append((keys[i],keys[j]))
    return score_matrix, pvalue_matrix, pairs
```

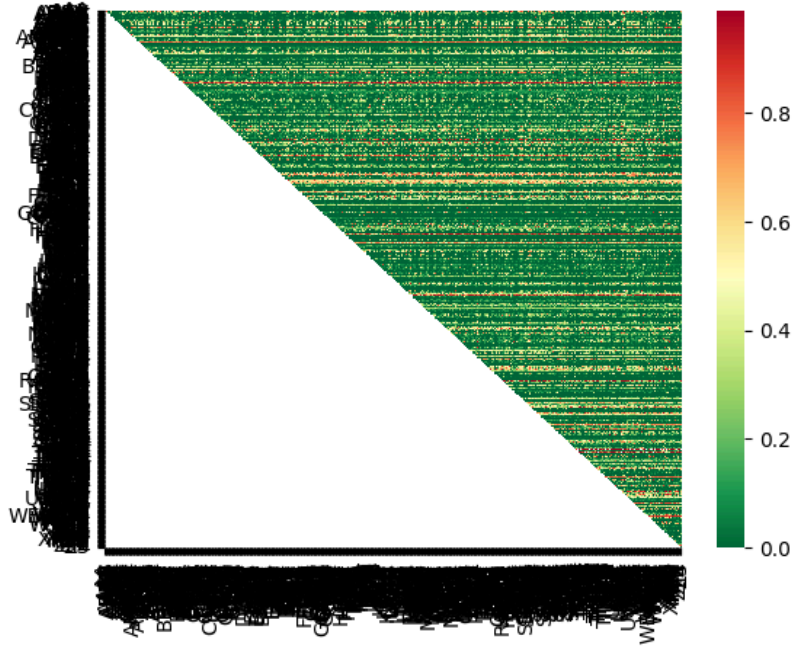
```
## assigning the tech tickers to the sybols list
symbol_list = tickers
```

```
##Printing the highly correlated pairs.
score_matrix, pvalue_matrix, pairs = find_cointegrated_pairs(pivot_df)
print(pairs)
```

The above code took 58 minutes to run and the results of the heat map are shown below. Analysing 100 stocks will be tough. Therefore, the below is there for future research purpose.

```
import seaborn
seaborn.heatmap(pvalue_matrix, xticklabels = tickers, yticklabels = tickers, cmap = 'RdYlGn_r'
                , mask = (pvalue_matrix >= 0.99)
                )
print(pairs)
```

```
[('AA', 'AAL'), ('AA', 'AAPL'), ('AA', 'ABBV'), ('AA', 'ABNB'), ('AA', 'ABT'), ('AA', 'ACB'), ('AA', 'ACN'), ('AA
```



✓ Cointegration on the top 100 hedge fund stock - Cross Sector Pick

The cointegration testing was applied on a cross sectors stock options implied volatility. There are 14 cross stock options implied plotted with a heatmap. This results were plotted from the 6 month dataset from 2023 March to 2023 August WRDS Dataset

```

# List of tickers to filter 10 stocks
tickers_to_filter = ['ADP', 'BA', 'BKNG', 'BX', 'ELV', 'GLD', 'ISRG', 'LRCX', 'NKE', 'PG', 'TGT', 'TSLA', 'UBER', 'WMT']
# Filter columns where ticker is in the list of tickers
filtered_pivot_df = pivot_df.loc[:, pivot_df.columns.isin(tickers_to_filter)]

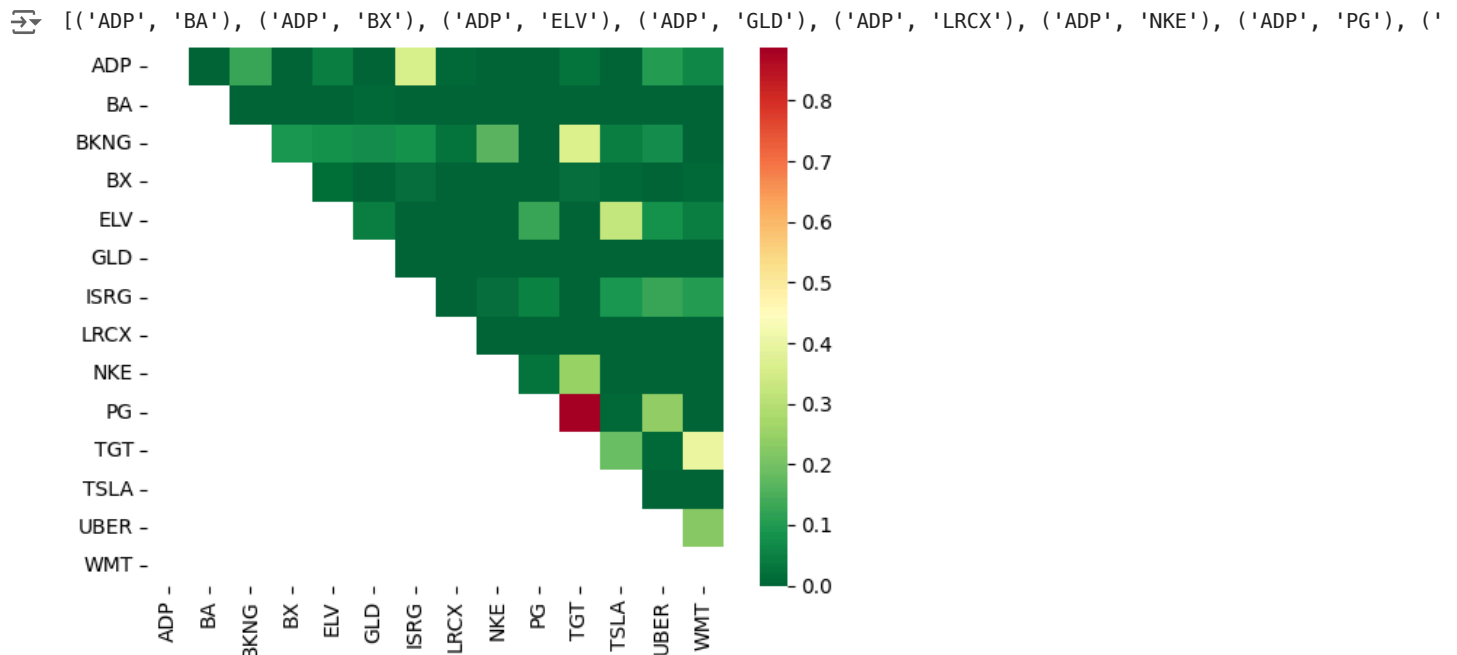
# Display the first few rows of the filtered DataFrame
filtered_pivot_df.head()

##Function to find cointegrated pairs for the tech stocks
def find_cointegrated_pairs(data):
    n=data.shape[1]
    score_matrix=np.zeros((n,n))
    pvalue_matrix=np.ones((n,n))
    keys=data.keys()
    pairs=[]
    for i in range(n):
        for j in range(i+1,n):
            s1=data[keys[i]]
            s2=data[keys[j]]
            result=coint(s1,s2)
            score, pvalue, _ = coint(s1, s2)
            score_matrix[i,j]=score
            pvalue_matrix[i,j]=pvalue
            if pvalue<0.05:
                pairs.append((keys[i],keys[j]))
    return score_matrix, pvalue_matrix, pairs

## assigning the tech tickers to the sybols list
symbol_list = tickers

##Printing the heatmap chart of the cointegrated tech stock.
scores, pvalues, pairs = find_cointegrated_pairs(filtered_pivot_df)
import seaborn
seaborn.heatmap(pvalues, xticklabels = tickers_to_filter, yticklabels = tickers_to_filter, cmap = 'RdYlGn_r'
                , mask = (pvalues >= 0.99)
                )
print(pairs)

```



Model Building - Tech Sector - NVDA

The model is built using Multi Linear Regression and Support Vector Machine. The models are compared to see performance. For this research part I have focused on the NVDA implied option prediction. The first MLR results were plotted from the 6 month dataset from 2023 March to 2023 August WRDS Dataset while the second MLR results were plotted for a 6 year dataset from 2018 January to 2023 August WRDS Dataset.

Multi Linear Regression - MLR

NVDA and NOW testing cointegration: Testing the graph and correlation on tech sector for NVDA and NOW. The implied volatility for both stock options is highly correlated with a significance of $8.72 * 10^{-9}$. The graph illustrate a mean reverting signal. The analysis below was done to build a basket that mimicks NVDA implied volatility. The heatmap above has led me to pick 5 stock options to build the MLR

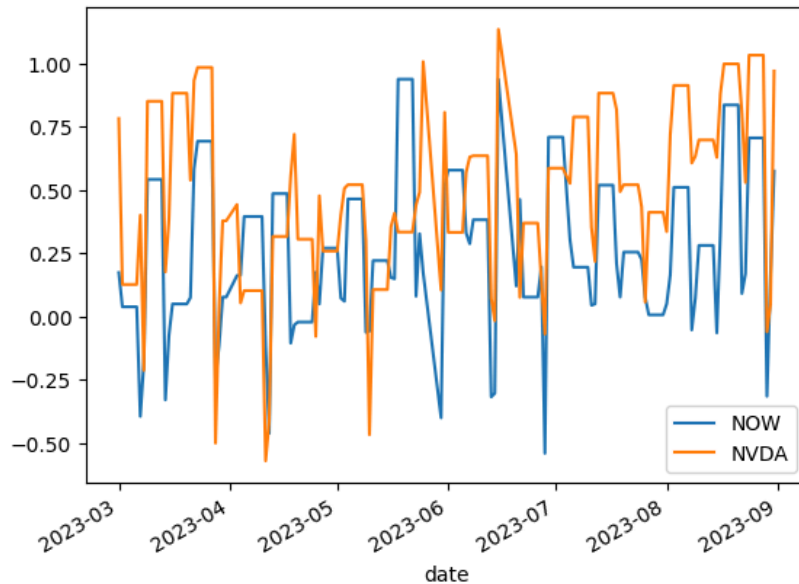
```
##Log of NVDA and NOW implied volatility is taken
NVDAlog = np.log(filtered_pivot_df['NVDA'])
NOWlog = np.log(filtered_pivot_df['NOW'])

##The cointegration of the logged value is taken
score, pvalue, _ = coint(NOWlog, NVDAlog)

##The significance test for cointegration is conducted
if pvalue < 0.05:
    print("Reject the null hypothesis: Time series are cointegrated.")
    print(pvalue)
else:
    print("Fail to reject the null hypothesis: Time series are not cointegrated.")
    print(pvalue)

##The graph of NVDA and NOW is plotted.
pd.concat([NOWlog, NVDAlog], axis=1).plot()

⇒ Reject the null hypothesis: Time series are cointegrated.
8.720241591243438e-09
<Axes: xlabel='date'>
```



Train and Test Split: The dataset of 6 months is split into a training and test data set. The training set contains 4 months of data and the test set contains 2 months of data. The split date is 30 June 2023.

```

## Data available if from March 2023 to August 2023. I have split the data into 4 Month Training and 2 Months Testing

# Define the split date
split_date = '06-30-2023'

# Convert split_date to datetime
split_date = pd.to_datetime(split_date)

# Split the DataFrame
train_df = filtered_pivot_df[filtered_pivot_df.index <= split_date]
test_df = filtered_pivot_df[filtered_pivot_df.index > split_date]

```

Building MLR Model for predicting NVDA: The MLR model is built where NOW, MSFT, GOOG, AMD and ADP are the predictors and NVDA is the target stock option whose implied volatility is predicted. The rsquared is obtained for the dataset. The rsquared is quite low at 0.35. However, the spread of the dataset is stationary with significance. Therefore, the spread has a mean reverting signal and the NVDA stock option implied volatility arbitrage can be traded. The Adickey Fuller test reveals the stationarity of the spread.

```

## Function to calculate the log value of the implied volatility on the training set.
def logVol(ticker):
    return np.log(train_df[ticker])

##testing to see the result of the function for the NVDA logged values
NVDAlog = logVol('NVDA')
NVDAlog.head()

## MLR regression fit on the training dataset to predict NVDA with a basket of NOW, MSFT, GOOG, AMD, and ADP
mlr = regression.linear_model.OLS(logVol('NVDA'), sm.add_constant(np.column_stack((logVol('NOW'), logVol('MSFT'), logVol('GOOG'), logVol('AMD'), logVol('ADP')))))

##Predicting the values on the
mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('NOW') + mlr.params[2] * logVol('MSFT') + mlr.params[3] * logVol('GOOG') + mlr.params[4] * logVol('AMD') + mlr.params[5] * logVol('ADP')

##Printing the rsquared value
print('MLR RSquared:', mlr.rsquared_adj)

##Finding the difference between the predicated and the actual log value
spread = NVDAlog - mlr_prediction

## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread)[1])

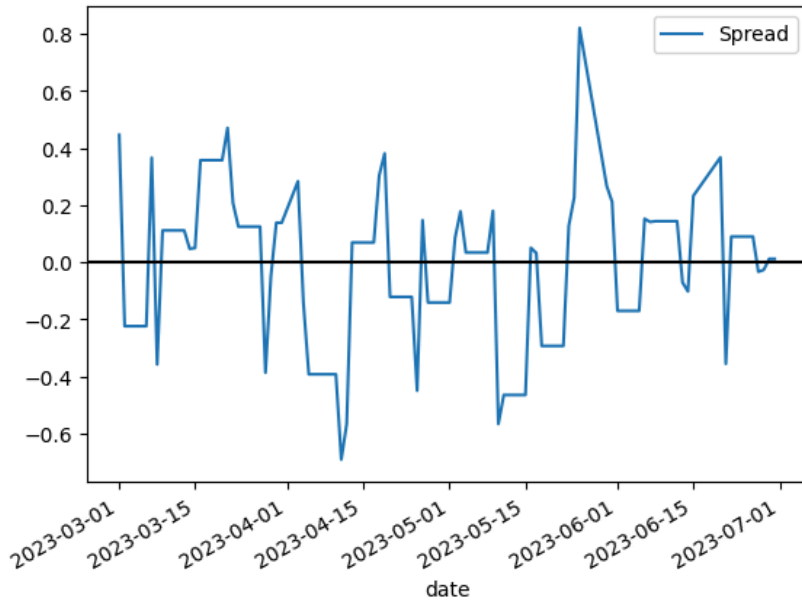
##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread)[0])

## Plotting the spread
spread.plot()
plt.axhline(spread.mean(), color='black')
plt.legend(['Spread'])
plt.show()

```

```

↳ <ipython-input-60-86176a9dcbc1>:13: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
    mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('NOW') + mlr.params[2] * logVol('MSFT') + mlr.params[3]
MLR RSquared: 0.3598336734532721
p value for insample stationarity: 4.365541441587516e-07
t statistics for in sample stationarity: -5.812623181796803
    
```



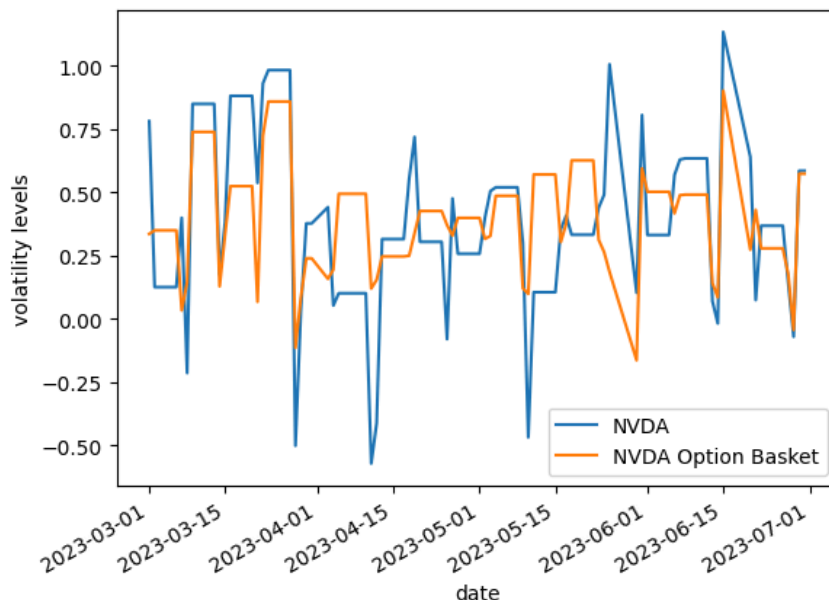
Plotting NVDA against Basket-Training Dataset The training set predictors is plotted against the NVDA options volatility. There is a clear mean reverting signal and arbitrage or a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of NVDA is higher than the basket, the investor can take a short position on NVDA call option

```

#Predictions and actual NVDA data plotted
mlr_prediction.name = 'NVDA Option Basket'
pd.concat([NVDAlog, mlr_prediction], axis=1).plot()
plt.ylabel('volatility levels')
    
```

```

↳ Text(0, 0.5, 'volatility levels')
    
```



Test Set Analysis for NVDA The following code runs the MLR model on the test data. The resulting spread is stationary as the p value is less than 0.05. Therefore, the spread has a constant mean and standard deviation.


```

## Function to calculate the log value of the implied volatility on the test set.
def logVolTest(ticker):
    return np.log(test_df[ticker])

##Code to see the result of the function for the NVDA logged values
NVDAlogTest = logVolTest('NVDA')
NVDAlogTest.head()

##Predicting the values on the
mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('NOW') + mlr.params[2] * logVolTest('MSFT') + mlr.pa

##Finding the difference between the predicated and the actual log value
spread_test = NVDAlogTest - mlr_prediction_test

## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread_test)[1])

##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread_test)[0])

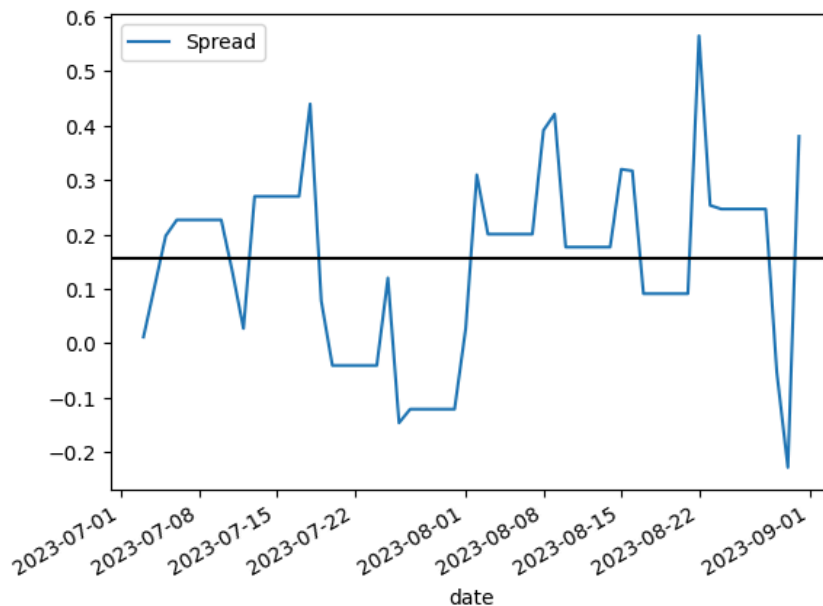
## Plotting the spread
spread_test.plot()
plt.axhline(spread_test.mean(), color='black')
plt.legend(['Spread'])
plt.show()

```

```

↗ <ipython-input-77-eb295b912cc9>:10: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. I
mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('NOW') + mlr.params[2] * logVolTest('MSFT') +
p value for insample stationarity: 0.0015252056724458247
t statistics for in sample stationarity: -3.978884362058182

```

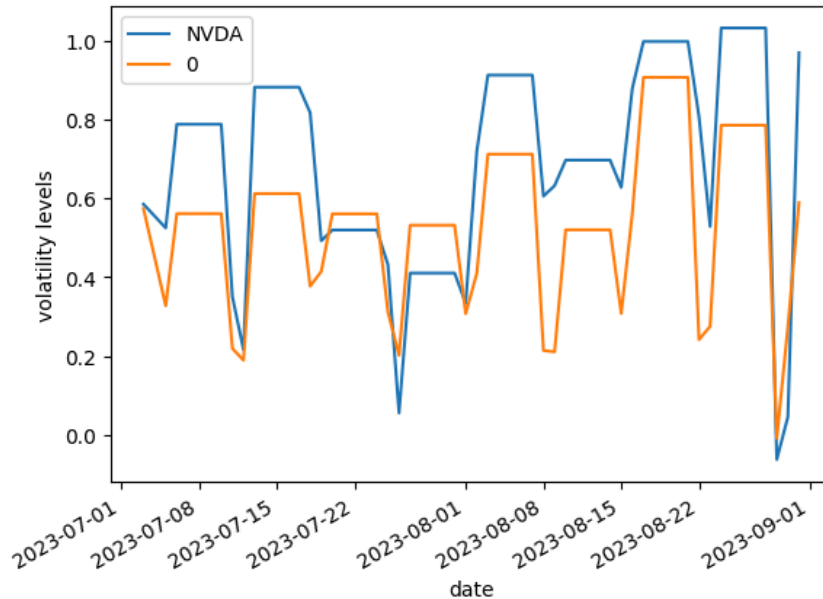


Plotting NVDA against Basket-Test Dataset The test set predictors is plotted against the NVDA options volatility. There is a clear mean reverting signal and arbitrage for a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of NVDA is higher than the basket, the investor can take a short position on NVDA call option

```
## The predicted against the basket log values are posted for the test dataset
```

```
mlr_prediction.name = 'NVDA Option Basket Test'
pd.concat([NVDAlogTest, mlr_prediction_test], axis=1).plot()
plt.ylabel('volatility levels')
```

```
⇒ Text(0, 0.5, 'volatility levels')
```



Multi Linear Regression - MLR 6 Year Dataset

NVDA and NOW testing cointegration: Testing the graph and correlation on tech sector for NVDA and NOW. The implied volatility for both stock options is highly correlated with a significance of $1.9e-13$. The graph illustrate a mean reverting signal. The analysis below was done to build a basket that mimicks NVDA implied volatility. The heatmap result from the 6 month data above is used to pick the 5 stocks. The MLR is applied on the 6 years trading dataset

```
# List of tickers to filter 10 stocks
tickers_to_filter = ['NVDA', 'NOW', 'MSFT', 'GOOG', 'AMD', 'ADP']

# Filter columns where ticker is in the list of tickers
filtered_pivot_df = pivot_df.loc[:, pivot_df.columns.isin(tickers_to_filter)]

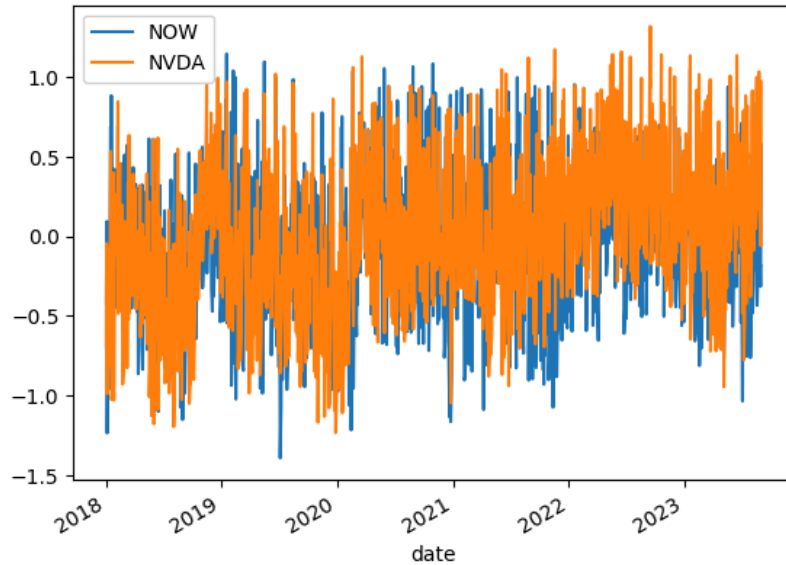
##Log of NVDA and NOW implied volatility is taken
NVDAlog = np.log(filtered_pivot_df['NVDA'])
NOWlog = np.log(filtered_pivot_df['NOW'])

##The cointegration of the logged value is taken
score, pvalue, _ = coint(NOWlog, NVDAlog)

##The significance test for cointegration is conducted
if pvalue < 0.05:
    print("Reject the null hypothesis: Time series are cointegrated.")
    print(pvalue)
else:
    print("Fail to reject the null hypothesis: Time series are not cointegrated.")
    print(pvalue)

##The graph of NVDA and NOW is plotted.
pd.concat([NOWlog, NVDAlog], axis=1).plot()
```

→ Reject the null hypothesis: Time series are cointegrated.
 1.9017047487690774e-13
 <Axes: xlabel='date'>



Train and Test Split: The dataset of 6 years is split into a training and test data set. The training set contains 5 years of data and the test set contains 8 months of data. The split date is 31 December 2022.

```
## Data available if from January 2018 to August 2023. I have split the data into 5 years Training and 8 Months Testi

# Define the split date
split_date = '12-31-2022'

# Convert split_date to datetime
split_date = pd.to_datetime(split_date)

# Split the DataFrame
train_df = filtered_pivot_df[filtered_pivot_df.index <= split_date]
test_df = filtered_pivot_df[filtered_pivot_df.index > split_date]
```

Building MLR Model for predicting NVDA: The MLR model is built where NOW, MSFT, GOOG, AMD and ADP are the predictors and NVDA is the target stock option whose implied volatility is predicted. The rsquared is obtained for the dataset. The rsquared is high at 0.62. This is higher compared to the 6 month data analysis. The spread of the dataset is stationary with significance. Therefore, the spread has a mean reverting signal and the NVDA stock option implied volatility arbitrage can be traded. The Adickey Fuller test reveals the stationarity of the spread.

```

## Function to calculate the log value of the implied volatility on the training set.
def logVol(ticker):
    return np.log(train_df[ticker])

##testing to see the result of the function for the NVDA logged values
NVDAlog = logVol('NVDA')
NVDAlog.head()

## MLR regression fit on the training dataset to predict NVDA with a basket of NOW, MSFT, GOOG, AMD, and ADP
mlr = regression.linear_model.OLS(logVol('NVDA'), sm.add_constant(np.column_stack((logVol('NOW'), logVol('MSFT'), log

##Predicting the values on the
mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('NOW') + mlr.params[2] * logVol('MSFT') + mlr.params[3] * log

##Printing the rsquared value
print('MLR RSquared:', mlr.rsquared_adj)


##Finding the difference between the predicated and the actual log value
spread = NVDAlog - mlr_prediction

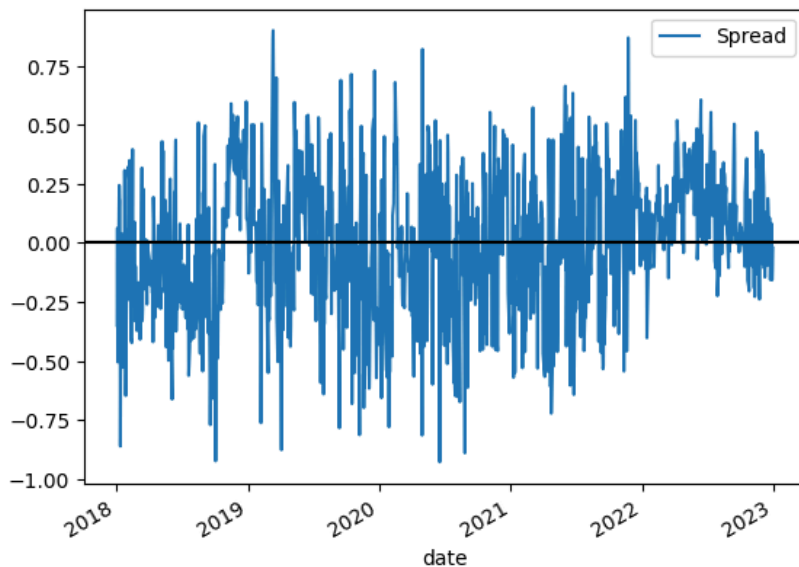
## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread)[1])

##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread)[0])

## Plotting the spread
spread.plot()
plt.axhline(spread.mean(), color='black')
plt.legend(['Spread'])
plt.show()


```

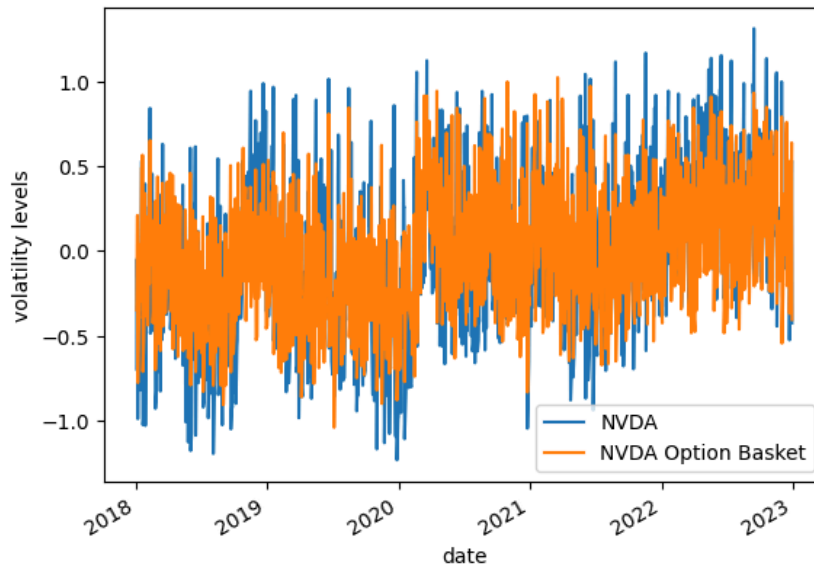
 <ipython-input-115-86176a9dcbc1>:13: FutureWarning: Series.__getitem__ treating keys as positions is deprecated.
 mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('NOW') + mlr.params[2] * logVol('MSFT') + mlr.params[3]
 MLR RSquared: 0.615256481962784
 p value for insample stationarity: 6.119061823123265e-05
 t statistics for in sample stationarity: -4.77317208063876



Plotting NVDA against Basket-Training Dataset The training set predictors is plotted against the NVDA options volatility. There is a clear mean reverting signal and arbitrage or a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of NVDA is higher than the basket, the investor can take a short position on NVDA call option

```
#Predictions and actual NVDA Options volatility data plotted
mlr_prediction.name = 'NVDA Option Basket'
pd.concat([NVDAlog, mlr_prediction], axis=1).plot()
plt.ylabel('volatility levels')
```

```
Text(0, 0.5, 'volatility levels')
```



Test Set Analysis for NVDA The following code runs the MLR model on the test data. The resulting spread is stationary as the p value is less than 0.05. Therefore, the spread has a constant mean and standard deviation.

```
## Function to calculate the log value of the implied volatility on the test set.
def logVolTest(ticker):
    return np.log(test_df[ticker])

##Code to see the result of the function for the NVDA logged values
NVDAlogTest = logVolTest('NVDA')
NVDAlogTest.head()

##Predicting the values on the
mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('NOW') + mlr.params[2] * logVolTest('MSFT') + mlr.pa

##Finding the difference between the predicated and the actual log value
spread_test = NVDAlogTest - mlr_prediction_test

## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread_test)[1])

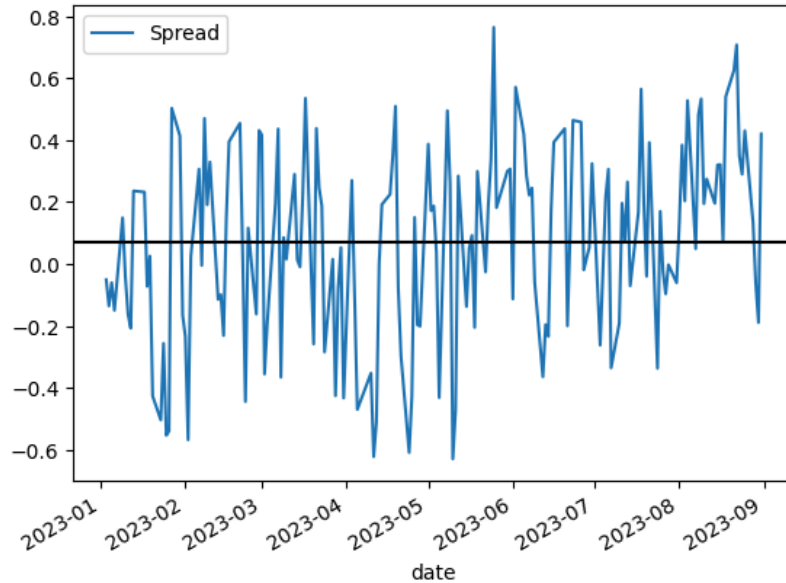
##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread_test)[0])

## Plotting the spread
spread_test.plot()
plt.axhline(spread_test.mean(), color='black')
plt.legend(['Spread'])
plt.show()
```

```

↳ <ipython-input-117-eb295b912cc9>:10: FutureWarning: Series.__getitem__ treating keys as positions is deprecated.
    mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('NOW') + mlr.params[2] * logVolTest('MSFT') +
p value for insample stationarity: 1.427881295338533e-15
t statistics for in sample stationarity: -9.258519261999913

```



Plotting NVDA against Basket-Test Dataset The test set predictors is plotted against the NVDA options volatility. There is a clear mean reverting signal and arbitrage for a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of NVDA is higher than the basket, the investor can take a short position on NVDA call option

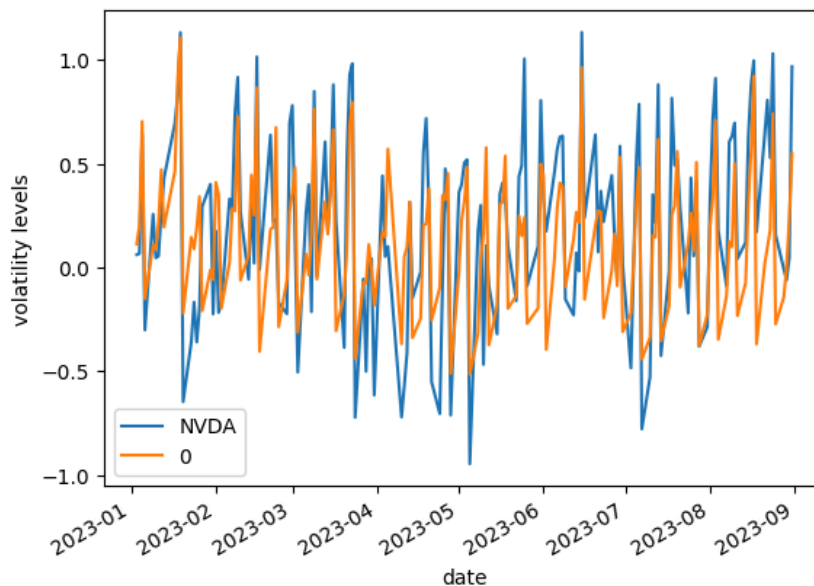
```
## The predicted against the basket log values are posted for the test dataset
```

```

mlr_prediction.name = 'NVDA Option Basket Test'
pd.concat([NVDAlogTest, mlr_prediction_test], axis=1).plot()
plt.ylabel('volatility levels')

```

```
↳ Text(0, 0.5, 'volatility levels')
```



Support Vector Machine - SVM 6 Year Dataset

NVDA SVM Model: SVM is used to build a model on the training dataset that was defined in the MLR section above. The same stocks are used as predictors. The spread is plotted and the p-value is $9.67e-06$ which is lower than 0.05; hence, the spread is stationary and has a mean reverting signal.

```
#SVM Model fitted to the training data
SVM_model = SVR()
SVM_model.fit(np.log(train_df[['NOW', 'MSFT', 'GOOG', 'AMD', 'ADP']]), np.log(train_df[['NVDA']]))

#SVM Model predicts the NVDA option volatility
SVM_preds = SVM_model.predict(np.log(train_df[['NOW', 'MSFT', 'GOOG', 'AMD', 'ADP']]))

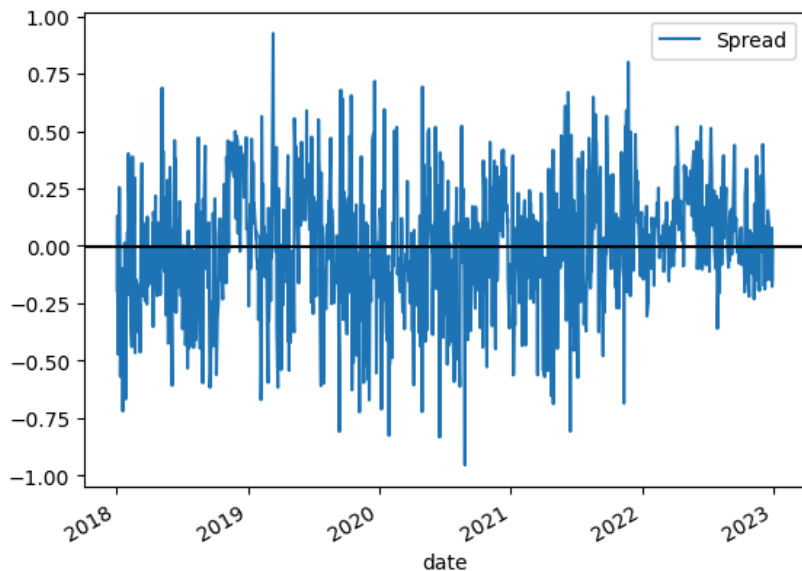
# Calculate the spread of the actual NVDA option volatility and the predicted one
spread = NVDAlog - SVM_preds

#The p-value is calculated for the spread. The value is less than 0.05; hence, spread is stationary
print ("p-value for in-sample stationarity: ", adfuller(spread)[1])
print ("t-statistics for in-sample stationarity: ", adfuller(spread)[0])

# Convert spread to a pandas Series for plotting
spread_series = pd.Series(spread, index=train_df.index)

# Plot the spread
spread_series.plot()
plt.axhline(spread_series.mean(), color='black')
plt.legend(['Spread'])
plt.show()

⚠ /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1183: DataConversionWarning: A column-vector
y = column_or_1d(y, warn=True)
p-value for in-sample stationarity: 9.673908337318868e-06
t-statistics for in-sample stationarity: -5.179636570726371
```



Test Prediction: The model is then used on the test dataset. The spread of the test dataset is calculated and the p value is $2.01e-16$; therefore, the spread is stationary

```
# predictions on the test dataset using the SVM model built
SVM_preds_test = SVM_model.predict(np.log(test_df[['NOW', 'MSFT', 'GOOG', 'AMD', 'ADP']]))

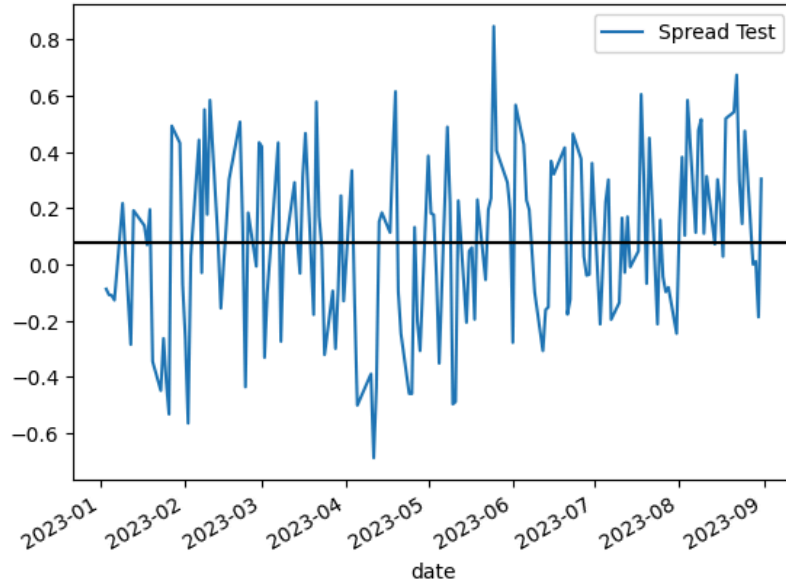
# Calculate the spread
spread_test = NVDAlogTest - SVM_preds_test

#printing p values for the spread. The spread is stationary as p value is less than 0.05
print ("p-value for in-sample stationarity: ", adfuller(spread_test)[1])
print ("t-statistics for in-sample stationarity: ", adfuller(spread_test)[0])
```

```
# Convert spread to a pandas Seriesn for plotting
spread_series = pd.Series(spread_test, index=test_df.index)

# Plot the spread on a graph
spread_series.plot()
plt.axhline(spread_series.mean(), color='black')
plt.legend(['Spread Test'])
plt.show()

⇒ p-value for in-sample stationarity: 2.0107562721764302e-16
t-statistics for in-sample stationarity: -9.593156268018701
```



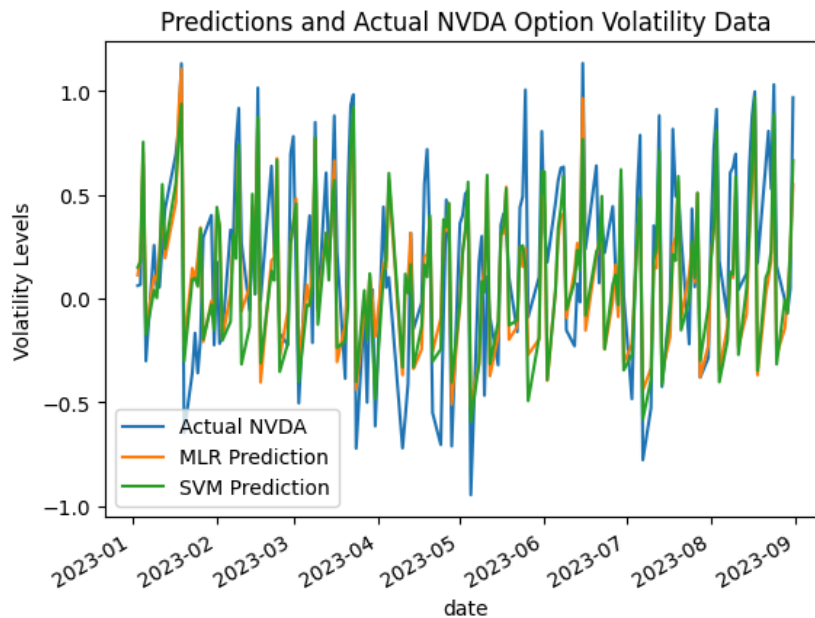
SVM vs MLR vs Actual NVDA: The 3 option volatility values are plotted for the test set. The results of the MLR and SVM model are very similar. An investor can trade this mean reverting stationary signals to take advantage of the volatility arbitrage.

```
mlr_prediction_test.name = "NVDA_basket_test"

# Convert SVM_preds_test to a pandas Series with the same index as NVDAlogTest for plotting
SVM_preds_test_series = pd.Series(SVM_preds_test, index=NVDALogTest.index, name="SVM_preds_test")

# Concatenate the data from the MLR, SVM and the actual NVDA options volatility
combined_data = pd.concat([NVDAlogTest, mlr_prediction_test, SVM_preds_test_series], axis=1)

# Plot the data
combined_data.plot()
plt.ylabel('Volatility Levels')
plt.title('Predictions and Actual NVDA Option Volatility Data')
plt.legend(['Actual NVDA', 'MLR Prediction', 'SVM Prediction'])
plt.show()
```

Model Building - Cross Sector - TSLA

The model is built using Multi Linear Regression and Support Vector Machine. The models are compared to see performance. For this research part I have focused on the TSLA implied option prediction. The predictors for TSLA is a cross basket of stocks. The first MLR results were plotted from the 6 month dataset from 2023 March to 2023 August WRDS Dataset while the second MLR results were plotted for a 6 year dataset from 2018 January to 2023 August WRDS Dataset.

Multi Linear Regression - MLR

TSLA and PG testing cointegration: Testing the graph and correlation on cross sector for TSLA and PG. The implied volatility for both stock options is highly correlated with a significance of $5.82e-09$. The graph illustrate a mean reverting signal. The analysis below was done to build a basket that mimicks TSLA implied volatility. The heatmap above has led me to pick 5 stock options to build the MLR

```
##Log of NVDA and NOW implied volatility is taken
TSLAlog = np.log(filtered_pivot_df['TSLA'])
BALog = np.log(filtered_pivot_df['BA'])

##The cointegration of the logged value is taken
score, pvalue, _ = coint(BAlog, TSLAlog)

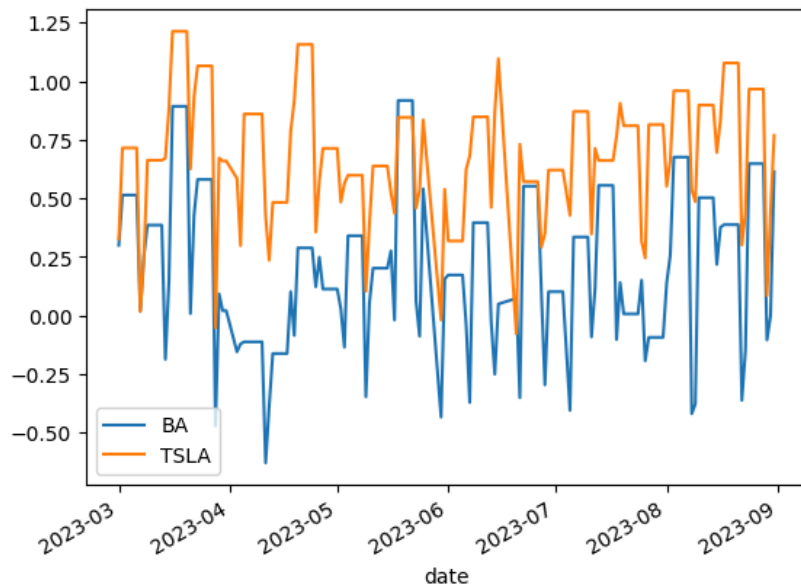
##The significance test for cointegration is conducted
if pvalue < 0.05:
    print("Reject the null hypothesis: Time series are cointegrated.")
    print(pvalue)
else:
    print("Fail to reject the null hypothesis: Time series are not cointegrated.")
    print(pvalue)

##The graph of NVDA and NOW is plotted.
pd.concat([BALog, TSLAlog], axis=1).plot()
```

→ Reject the null hypothesis: Time series are cointegrated.

5.817446934986433e-09

<Axes: xlabel='date'>



Train and Test Split: The dataset of 6 months is split into a training and test data set. The training set contains 4 months of data and the test set contains 2 months of data. The split date is 30 June 2023.

```
## Data available if from March 2023 to August 2023. I have split the data into 4 Month Training and 2 Months Testing
```

```
# Define the split date
split_date = '06-30-2023'
```

```
# Convert split_date to datetime
split_date = pd.to_datetime(split_date)
```

```
# Split the DataFrame
train_df = filtered_pivot_df[filtered_pivot_df.index <= split_date]
test_df = filtered_pivot_df[filtered_pivot_df.index > split_date]
```

Building MLR Model for predicting TSLA: The MLR model is built where PG, NKE, LRCX, GLD and BA are the predictors and TSLA is the target stock option whose implied volatility is predicted. The rsquared is obtained for the dataset. The rsquared is quite low at 0.40. However, the spread of the dataset is stationary with significance. Therefore, the spread has a mean reverting signal and the NVDA stock option implied volatility arbitrage can be traded. The Adickey Fuller test reveals the stationarity of the spread.

```
## Function to calculate the log value of the implied volatility on the training set.
```

```
def logVol(ticker):
    return np.log(train_df[ticker])
```

```
##testing to see the result of the function for the NVDA logged values
```

```
TSLAlog = logVol('TSLA')
TSLAlog.head()
```

```
## MLR regression fit on the training dataset to predict NVDA with a basket of NOW, MSFT, GOOG, AMD, and ADP
mlr = regression.linear_model.OLS(logVol('TSLA'), sm.add_constant(np.column_stack((logVol('PG'), logVol('NKE'), logVol
```

```
##Predicting the values on the
mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('PG') + mlr.params[2] * logVol('NKE') + mlr.params[3] * logVol
```

```
##Printing the rsquared value
print('MLR RSquared:', mlr.rsquared_adj)
```

```
##Finding the difference between the predicated and the actual log value
spread = TSLAlog - mlr_prediction
```

```

## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread)[1])

##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread)[0])

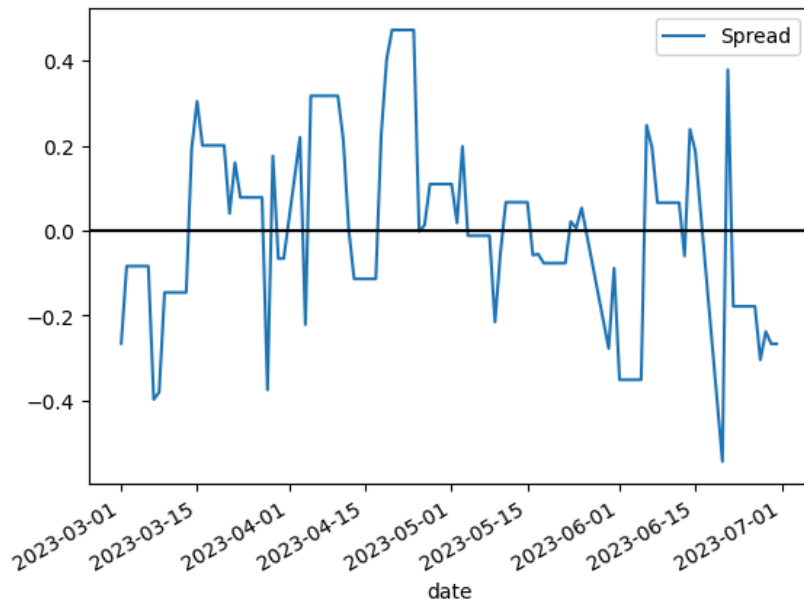
## Plotting the spread
spread.plot()
plt.axhline(spread.mean(), color='black')
plt.legend(['Spread'])
plt.show()

```

```

↵ <ipython-input-37-77e98c9473cd>:13: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, only string, tuple, or integer keys are allowed.
mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('PG') + mlr.params[2] * logVol('NKE') + mlr.params[3] *
MLR RSquared: 0.4021363553545346
p value for insample stationarity: 0.003250233657528494
t statistics for in sample stationarity: -3.768175161496668

```



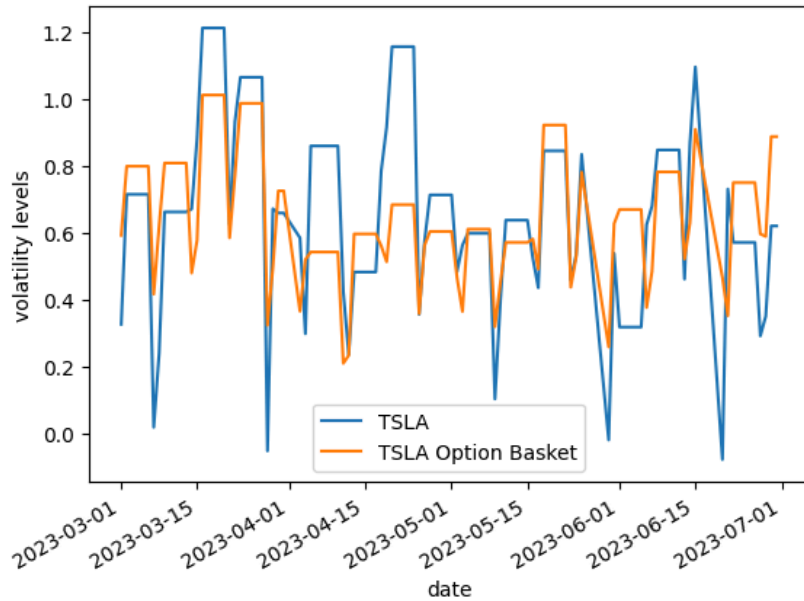
Plotting TSLA against Basket-Training Dataset The training set predictors is plotted against the TSLA options volatility. There is a clear mean reverting signal and arbitrage or a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of TSLA is higher than the basket, the investor can take a short position on TSLA call option

```

mlr_prediction.name = 'TSLA Option Basket'
pd.concat([TSLAlog, mlr_prediction], axis=1).plot()
plt.ylabel('volatility levels')

```

```
Text(0, 0.5, 'volatility levels')
```



Test Set Analysis for TSLA The following code runs the MLR model on the test data. The resulting spread is stationary as the p value is less than 0.05. Therefore, the spread has a constant mean and standard deviation.

```
## Function to calculate the log value of the implied volatility on the test set.
def logVolTest(ticker):
    return np.log(test_df[ticker])

##Code to see the result of the function for the NVDA logged values
TSLAlogTest = logVolTest('TSLA')
TSLAlogTest.head()

##Predicting the values on the
mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('PG') + mlr.params[2] * logVolTest('NKE') + mlr.param

##Finding the difference between the predicated and the actual log value
spread_test = TSLAlogTest - mlr_prediction_test

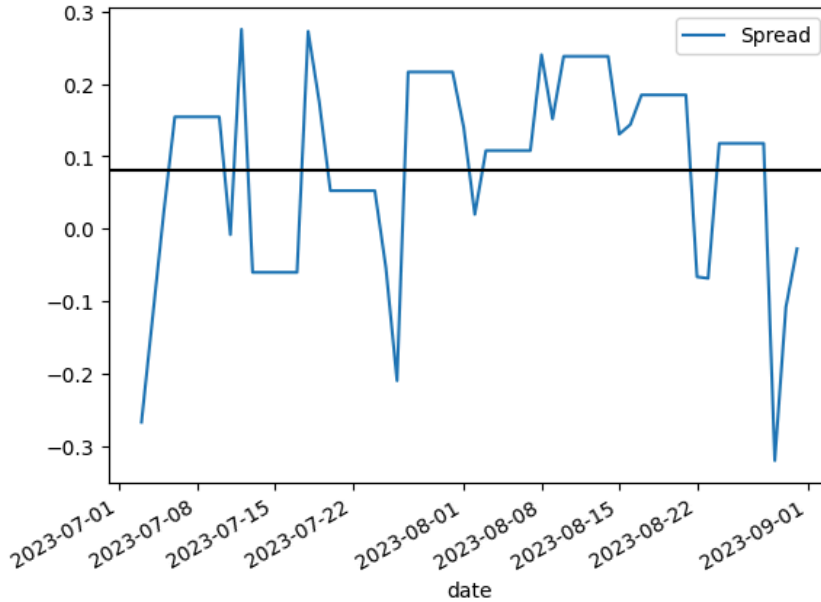
## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread_test)[1])

##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread_test)[0])

## Plotting the spread
spread_test.plot()
plt.axhline(spread_test.mean(), color='black')
plt.legend(['Spread'])
plt.show()
```

```

↳ <ipython-input-39-df37f6742c2d>:10: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
    mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('PG') + mlr.params[2] * logVolTest('NKE') + ml
p value for insample stationarity: 1.7701771584970987e-05
t statistics for in sample stationarity: -5.049422220927951
    
```



Plotting TSLA against Basket-Test Dataset The test set predictors is plotted against the TSLA options volatility. There is a clear mean reverting signal and arbitrage for a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of TSLA is higher than the basket, the investor can take a short position on TSLA call option

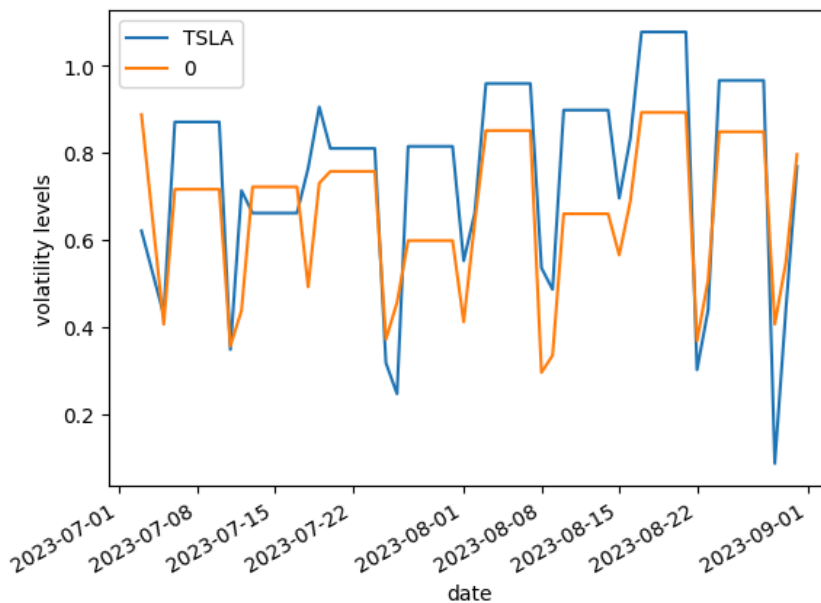
The predicted against the basket log values are posted

```

mlr_prediction.name = 'TSLA Option Basket Test'
pd.concat([TSLAlogTest, mlr_prediction_test], axis=1).plot()
plt.ylabel('volatility levels')
    
```

```

↳ Text(0, 0.5, 'volatility levels')
    
```



Multi Linear Regression - MLR 6 Year Dataset PENDING

TSLA and PG testing cointegration: Testing the graph and correlation on cross sector for TSLA and PG. The implied volatility for both stock options are not not correlated with a significance of 0.053. Since the pvalue is close to 0.05. For research purpose the data is considered to be correlated with a lower significance. The graph illustrate a mean reverting signal. The analysis below was done to build a basket that mimicks TSLA implied volatility. The heatmap above has led me to pick 5 stock options to build the MLR

```
# List of tickers to filter 10 stocks
tickers_to_filter = ['TSLA', 'PG', 'NKE', 'LRCX', 'GLD', 'BA']

# Filter columns where ticker is in the list of tickers
filtered_pivot_df = pivot_df.loc[:, pivot_df.columns.isin(tickers_to_filter)]

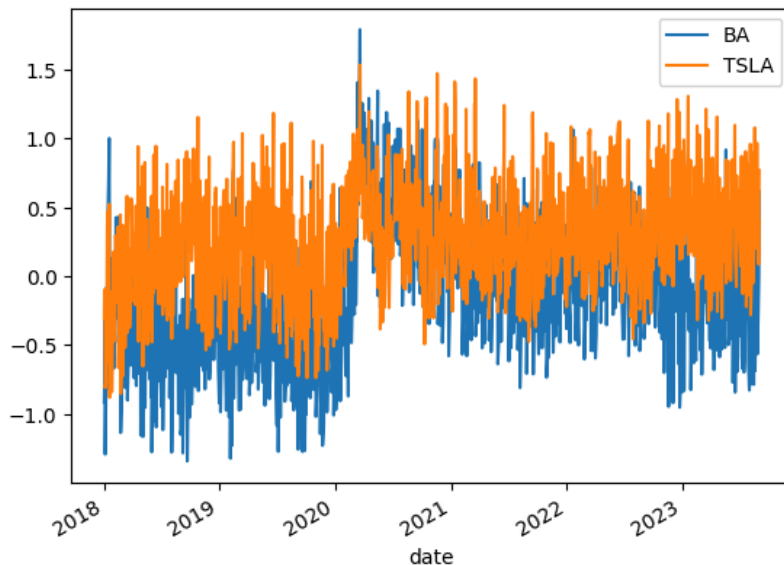
##Log of NVDA and NOW implied volatility is taken
TSLAlog = np.log(filtered_pivot_df['TSLA'])
BALog = np.log(filtered_pivot_df['BA'])

##The cointegration of the logged value is taken
score, pvalue, _ = coint(BAlog, TSLAlog)

##The significance test for cointegration is conducted
if pvalue < 0.05:
    print("Reject the null hypothesis: Time series are cointegrated.")
    print(pvalue)
else:
    print("Fail to reject the null hypothesis: Time series are not cointegrated.")
    print(pvalue)

##The graph of NVDA and NOW is plotted.
pd.concat([BALog, TSLAlog], axis=1).plot()
```

Fail to reject the null hypothesis: Time series are not cointegrated.
0.05291603945882446
<Axes: xlabel='date'>



Train and Test Split: The dataset of 6 months is split into a training and test data set. The training set contains 4 months of data and the test set contains 2 months of data. The split date is 30 June 2023.

```

## Data available if from January 2018 to August 2023. I have split the data into 5 years Training and 8 Months Testi

# Define the split date
split_date = '12-31-2022'

# Convert split_date to datetime
split_date = pd.to_datetime(split_date)

# Split the DataFrame
train_df = filtered_pivot_df[filtered_pivot_df.index <= split_date]
test_df = filtered_pivot_df[filtered_pivot_df.index > split_date]

```

Building MLR Model for predicting TSLA: The MLR model is built where PG, NKE, LRCX, GLD and BA are the predictors and TSLA is the target stock option whose implied volatility is predicted. The rsquared is obtained for the dataset. The rsquared is quite low at 0.40. However, the spread of the dataset is stationary with significance. Therefore, the spread has a mean reverting signal and the NVDA stock option implied volatility arbitrage can be traded. The Adickey Fuller test reveals the stationarity of the spread.

```

## Function to calculate the log value of the implied volatility on the training set.
def logVol(ticker):
    return np.log(train_df[ticker])

##testing to see the result of the function for the NVDA logged values
TSLAlog = logVol('TSLA')
TSLAlog.head()

## MLR regression fit on the training dataset to predict NVDA with a basket of NOW, MSFT, GOOG, AMD, and ADP
mlr = regression.linear_model.OLS(logVol('TSLA'), sm.add_constant(np.column_stack((logVol('PG'), logVol('NKE'), logVol

##Predicting the values on the
mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('PG') + mlr.params[2] * logVol('NKE') + mlr.params[3] * logVol

##Printing the rsquared value
print('MLR RSquared:', mlr.rsquared_adj)

##Finding the difference between the predicated and the actual log value
spread = TSLAlog - mlr_prediction

## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread)[1])

##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread)[0])

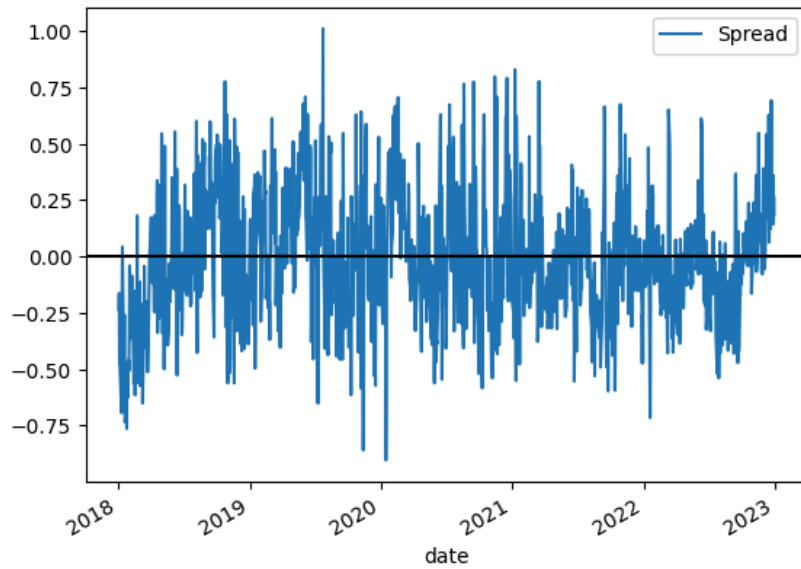
## Plotting the spread
spread.plot()
plt.axhline(spread.mean(), color='black')
plt.legend(['Spread'])
plt.show()

```

```

↳ <ipython-input-101-77e98c9473cd>:13: FutureWarning: Series.__getitem__ treating keys as positions is deprecated.
    mlr_prediction = mlr.params[0] + mlr.params[1] * logVol('PG') + mlr.params[2] * logVol('NKE') + mlr.params[3] *
MLR RSquared: 0.48095040906746633
p value for insample stationarity: 5.2268533347294655e-05
t statistics for in sample stationarity: -4.809008280440021

```



Plotting TSLA against Basket-Training Dataset The training set predictors is plotted against the TSLA options volatility. There is a clear mean reverting signal and arbitrage or a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of TSLA is higher than the basket, the investor can take a short position on TSLA call option

```

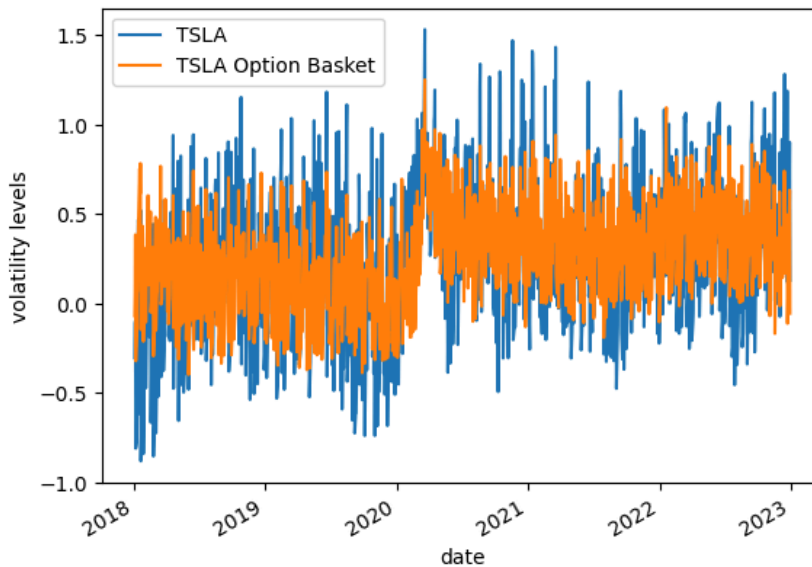
mlr_prediction.name = 'TSLA Option Basket'
pd.concat([TSLAlog, mlr_prediction], axis=1).plot()
plt.ylabel('volatility levels')

```

```

↳ Text(0, 0.5, 'volatility levels')

```



Test Set Analysis for TSLA The following code runs the MLR model on the test data. The resulting spread is stationary as the p value is less than 0.05. Therefore, the spread has a constant mean and standard deviation.

```

## Function to calculate the log value of the implied volatility on the test set.
def logVolTest(ticker):

```



```

return np.log(test_df[ticker])

##Code to see the result of the function for the NVDA logged values
TSLAlogTest = logVolTest('TSLA')
TSLAlogTest.head()

##Predicting the values on the
mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('PG') + mlr.params[2] * logVolTest('NKE') + mlr.param

##Finding the difference between the predicated and the actual log value
spread_test = TSLAlogTest - mlr_prediction_test

## Printing the AD Fuller test to see if spread is stationary
print('p value for insample stationarity:', adfuller(spread_test)[1])

##Printing the t Statistics
print('t statistics for in sample stationarity:', adfuller(spread_test)[0])

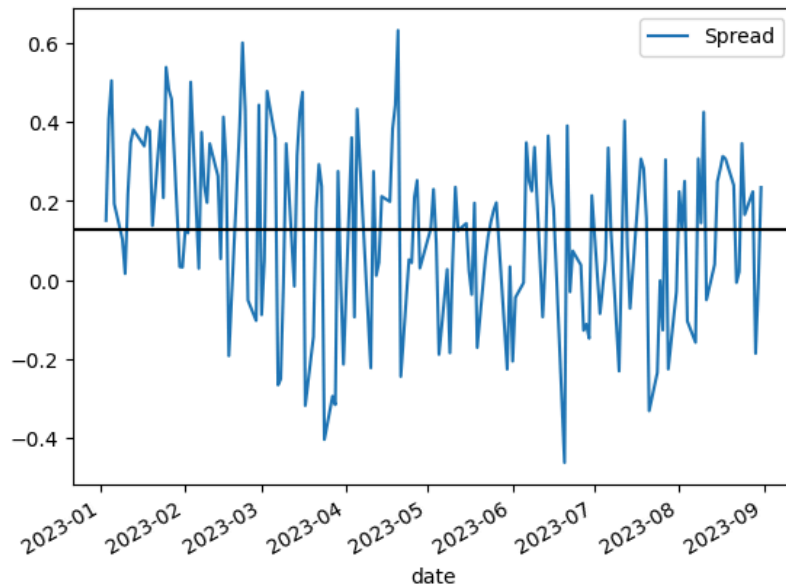
## Plotting the spread
spread_test.plot()
plt.axhline(spread_test.mean(), color='black')
plt.legend(['Spread'])
plt.show()

```

```

↳ <ipython-input-103-df37f6742c2d>:10: FutureWarning: Series.__getitem__ treating keys as positions is deprecated.
  mlr_prediction_test = mlr.params[0] + mlr.params[1] * logVolTest('PG') + mlr.params[2] * logVolTest('NKE') + ml
p value for insample stationarity: 0.02167477289427217
t statistics for in sample stationarity: -3.171822036085348

```



Plotting TSLA against Basket-Test Dataset The test set predictors is plotted against the TSLA options volatility. There is a clear mean reverting signal and arbitrage for a trader to act on. Whenever the implied volatility is less than the basket the investor can take a long position on the call option and whenever the implied volatility of TSLA is higher than the basket, the investor can take a short position on TSLA call option

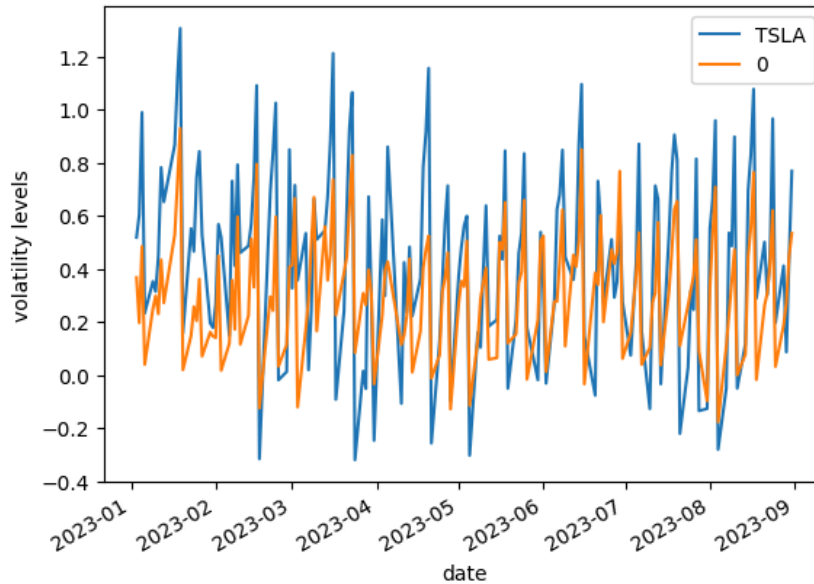
```

## The predicted against the basket log values are posted

mlr_prediction.name = 'TSLA Option Basket Test'
pd.concat([TSLAlogTest, mlr_prediction_test], axis=1).plot()
plt.ylabel('volatility levels')

```

```
Text(0, 0.5, 'volatility levels')
```



Support Vector Machine - SVM 6 Year Dataset

TSLA SVM Model: SVM is used to build a model on the training dataset that was defined in the MLR section above. The same stocks are used as predictors. The spread is plotted and the p-value is $5.43e-05$ which is lower than 0.05 ; hence, the spread is stationary and has a mean reverting signal.

```
#SVM Model fitted to the training data
SVM_model = SVR()
SVM_model.fit(np.log(train_df[['PG', 'NKE', 'LRCX', 'GLD', 'BA']]), np.log(train_df[['TSLA']]))

#SVM Model predicts the NVDA option volatility
SVM_preds = SVM_model.predict(np.log(train_df[['PG', 'NKE', 'LRCX', 'GLD', 'BA']]))

# Calculate the spread of the actual NVDA option volatility and the predicted one
spread = TSLAlog - SVM_preds

#The pvalue is calculated for the spread. The value is less than 0.05; hence, spread is stationary
print("p-value for in-sample stationarity: ", adfuller(spread)[1])
print("t-statistics for in-sample stationarity: ", adfuller(spread)[0])

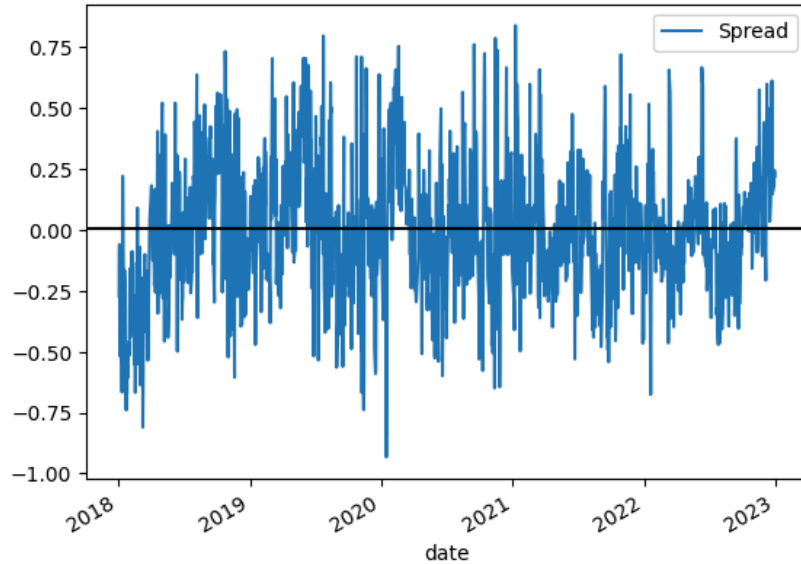
# Convert spread to a pandas Series for plotting
spread_series = pd.Series(spread, index=train_df.index)

# Plot the spread
spread_series.plot()
plt.axhline(spread_series.mean(), color='black')
plt.legend(['Spread'])
plt.show()
```

```

→ /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1183: DataConversionWarning: A column-vector
y = column_or_1d(y, warn=True)
p-value for in-sample stationarity: 5.433179311021402e-05
t-statistics for in-sample stationarity: -4.800226119244746

```



Test Prediction: The model is then used on the test dataset. The spread of the test dataset is calculated and the p value is $8.47e-20$; therefore, the spread is stationary

```

# predictions on the test dataset using the SVM model built
SVM_preds_test = SVM_model.predict(np.log(test_df[['PG', 'NKE', 'LRCX', 'GLD', 'BA' ]]))

# Calculate the spread
spread_test = TSLAlogTest - SVM_preds_test

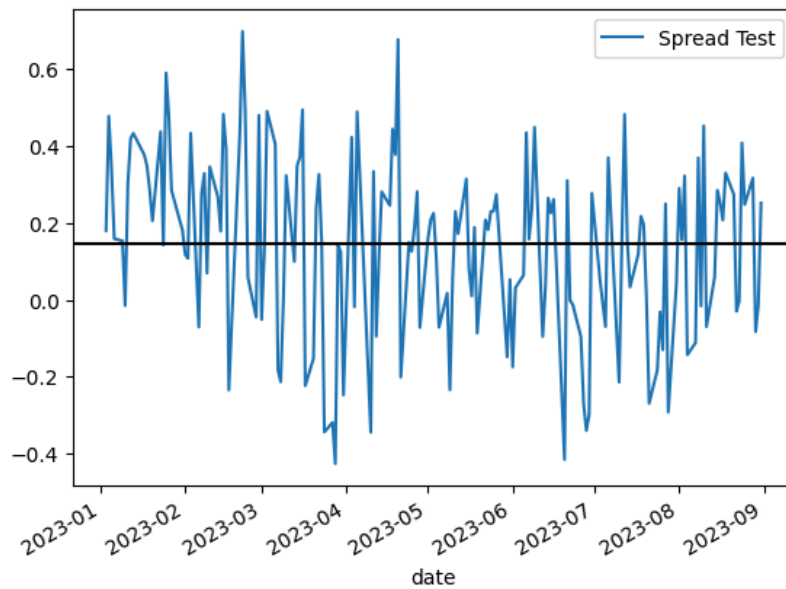
#printing p values for the spread. The spread is stationary as p value is less than 0.05
print ("p-value for in-sample stationarity: ", adfuller(spread_test)[1])
print ("t-statistics for in-sample stationarity: ", adfuller(spread_test)[0])

# Convert spread to a pandas Seriesn for plotting
spread_series = pd.Series(spread_test, index=test_df.index)

# Plot the spread on a graph
spread_series.plot()
plt.axhline(spread_series.mean(), color='black')
plt.legend(['Spread Test'])
plt.show()

```

→ p-value for in-sample stationarity: 8.469519838772011e-20
 t-statistics for in-sample stationarity: -10.957917422363442



SVM vs MLR vs Actual TSLA: The 3 option volatility values are plotted for the test set. The results of the MLR and SVM model are very similar. An investor can trade this mean reverting stationary signals to take advantage of the volatility arbitrage.

```
mlr_prediction_test.name = "TSLA_basket_test"
```

```
# Convert SVM_preds_test to a pandas Series with the same index as NVDAlogTest for plotting
SVM_preds_test_series = pd.Series(SVM_preds_test, index=TSLAlogTest.index, name="SVM_preds_test")
```

```
# Concatenate the data from the MLR, SVM and the actual NVDA options volatility
combined_data = pd.concat([TSLAlogTest, mlr_prediction_test, SVM_preds_test_series], axis=1)
```

```
# Plot the data
combined_data.plot()
plt.ylabel('Volatility Levels')
plt.title('Predictions and Actual NVDA Option Volatility Data')
plt.legend(['Actual TSLA', 'MLR Prediction', 'SVM Prediction'])
plt.show()
```

→

