Project 1:
Stock Portfolio

Riley Hopkins

Investment Strategy

- $100,000 investment on January 1st 2023, split evenly between 4 stocks ($25k each)
- Use daily market data to measure our cumulative return
  - Cumulative return: Net profit/loss if we sold everything on each day
  - Uses close price for each day
- Use cumulative return to calculate volatility, sharpe ratio, other metrics
- Get total return at the end

Introduction to Stocks

NVDA

- Nvidia
- Graphics card company
- Performed well recently

C:EURUSD (Forex)

- Comparing Euros and USD
- Volatile
- Benchmark comparing 2 currencies

WGMI (ETF)

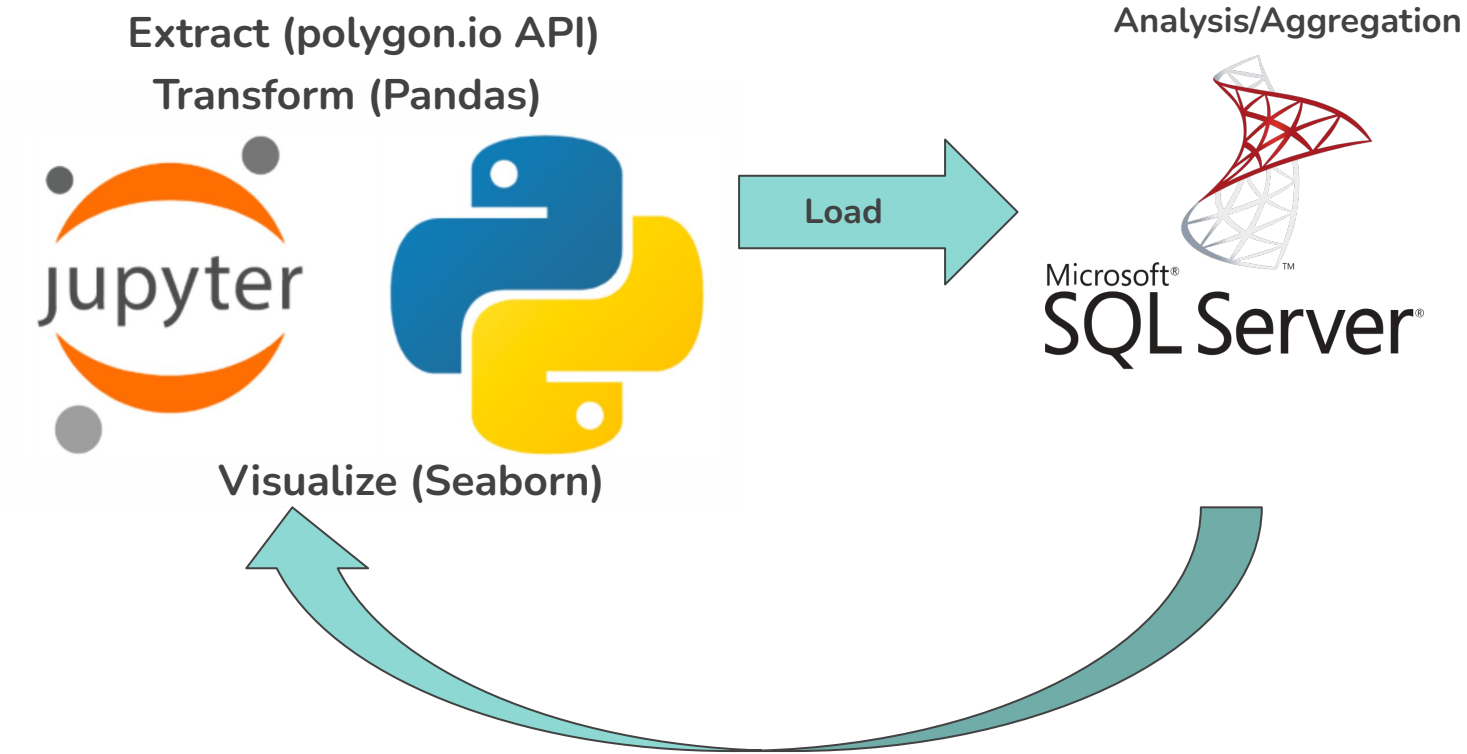
- Valkyrie Bitcoin Miners ETF
- Exchange trade fund
- Performed well recently

NDX (Market Index)

- NASDAQ
- 100 Largest Companies
- Represents the whole market

Idea:
Compare NVDA and WGMI to see if there is a correlation between performance of graphics cards and bitcoin mining operations.

ETL Pipeline



ETL Pipeline (Extract)

- Load data from API
- Convert into dataframe from json
- Use Pandas in Python notebooks for dataframes

```
# Construct the API URLs
wgmi_url = f"https://api.polygon.io/v2/aggs/ticker/WGMI/range/
nvda_url = f"https://api.polygon.io/v2/aggs/ticker/NVDA/range/
forex_url = f"https://api.polygon.io/v2/aggs/ticker/C:EURUSD/r
index_url = f"https://api.polygon.io/v2/aggs/ticker/I:NDX/rang

# Make requests to API
etf_data = requests.get(wgmi_url).json()
nvda_data = requests.get(nvda_url).json()
forex_data = requests.get(forex_url).json()
index_data = requests.get(index_url).json()

# Convert responses into dataframes
etf_df = pd.json_normalize(etf_data['results'])
nvda_df = pd.json_normalize(nvda_data['results'])
forex_df = pd.json_normalize(forex_data['results'])
index_df = pd.json_normalize(index_data['results'])
```

ETL Pipeline (Transform)

- Check for nulls
- Drop unnecessary columns
- Rename column names
- Data type conversions (UNIX time to datetime)
- Add in ticker symbols
- Combine all dataframes into one

```
# Drop unecessary columns
etf_df.drop(columns=['v', 'vw', 'n'], inplace=True)
nvda_df.drop(columns=['v', 'vw', 'n'], inplace=True)
forex_df.drop(columns=['v', 'vw', 'n'], inplace=True)
```

```
etf_df.rename(columns={
    'o': 'Open Price ($)',
    'c': 'Close Price ($)',
    'h': 'Highest Price ($)',
    'l': 'Lowest Price ($)',
    't': 'Date'
    }, inplace=True)
```

```
# Convert Unix to datetime
etf_df['Date'] = pd.to_datetime(etf_df['Date'], unit="ms")
nvda_df['Date'] = pd.to_datetime(nvda_df['Date'], unit="ms")
forex_df['Date'] = pd.to_datetime(forex_df['Date'], unit="ms")
index_df['Date'] = pd.to_datetime(index_df['Date'], unit="ms")

# Format datetime
etf_df['Date'] = etf_df['Date'].dt.strftime('%Y-%m-%d')
nvda_df['Date'] = nvda_df['Date'].dt.strftime('%Y-%m-%d')
forex_df['Date'] = forex_df['Date'].dt.strftime('%Y-%m-%d')
index_df['Date'] = index_df['Date'].dt.strftime('%Y-%m-%d')
```

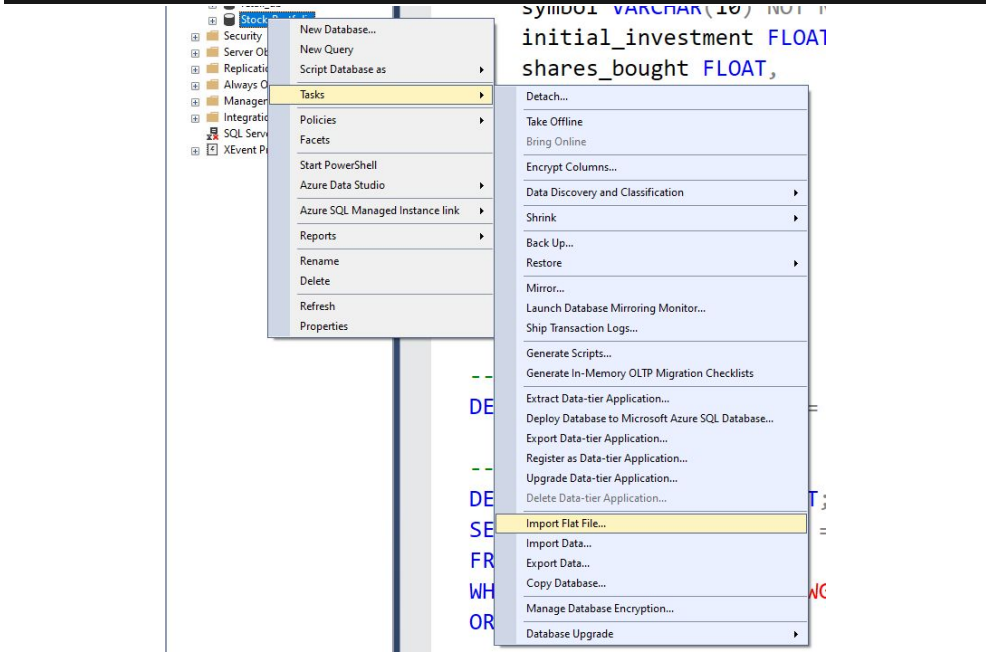
```
# Add in ticker symbols
etf_df.insert(loc=0, column="Symbol", value="WGMI")
nvda_df.insert(loc=0, column="Symbol", value="NVDA")
forex_df.insert(loc=0, column="Symbol", value="C:EURUSD")
index_df.insert(loc=0, column="Symbol", value="NDX")
```

```
# Combine all 4 dataframes
df = pd.concat([etf_df, nvda_df, forex_df, index_df], ignore_index=True)
```

ETL Pipeline (Load)

- Exported to CSV file
- Imported as flat file into SQL Server

```
# Export to a CSV file
df.to_csv('stockdata.csv', index=False)
```



Analysis/Aggregation in SQL Server

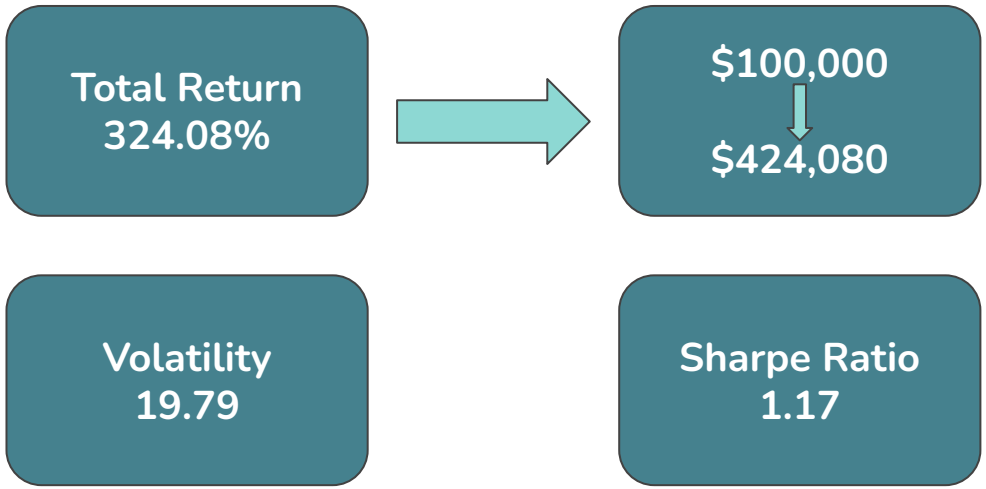
- 2 Tables (flat values & aggregations)
- Data based on % of initial investment

	symbol	initial_investment	shares_bought	total_return	volatility	avg_return	sharpe_ratio
1	WGMI	25000	5483.89900650562	95.5360976526639	22.6497528306754	45.9282186811011	1.67490974377274
2	NVDA	25000	1746.41989052232	210.609513825946	51.8433440107945	70.4742062298819	1.2052131368284
3	C:EURUSD	25000	23163.1603217425	0.0926514484209692	0.335181711335708	0.0617460878085324	-23.6593483920945
4	NDX	25000	2.0718910586732	17.8371326905399	4.33673084656527	7.99192697095851	0
5	PORTFOLIO	100000	30395.5511098291	324.075395617571	19.7912523498427	31.1140244924375	1.16829885813986

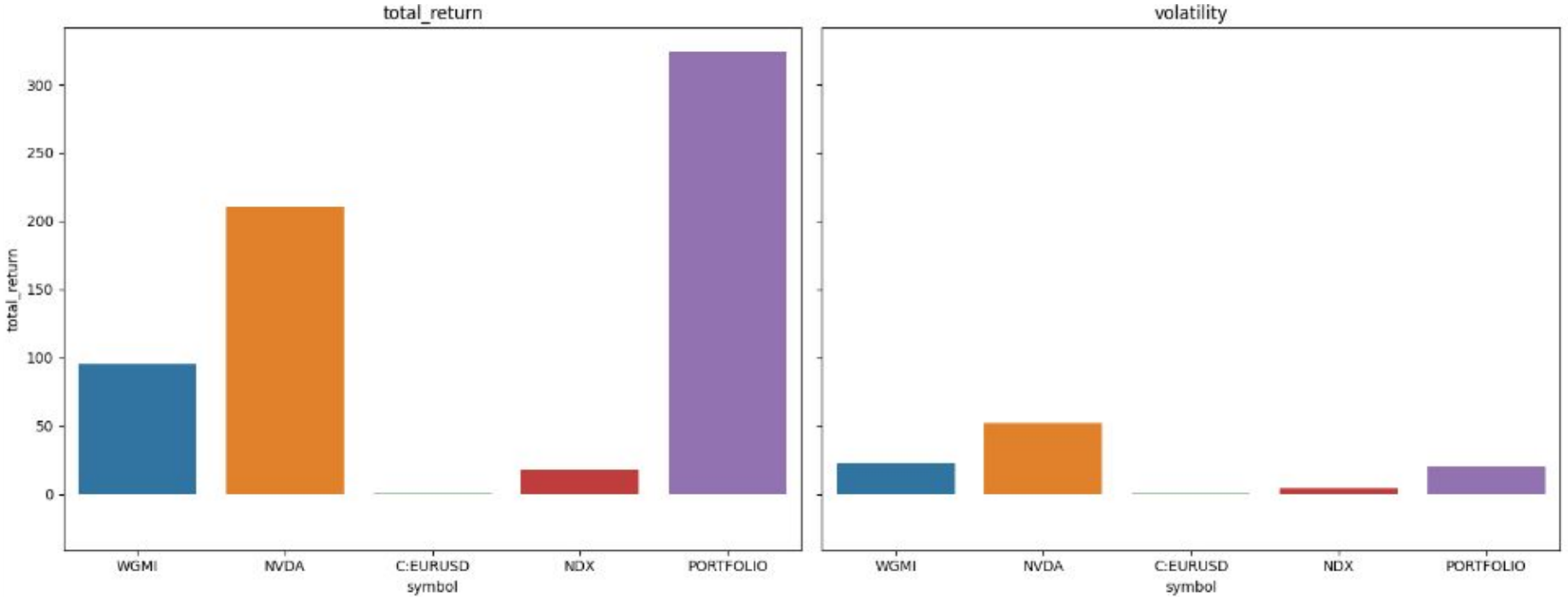
	Symbol	Date	Cumulative Return	moving_avg_10_day	moving_avg_100_day
1	C:EURUSD	2023-01-01	0	-0.381961767540239	-0.0406662929756215
		023-01-02	-0.270084743188654	-0.369260597960899	-0.0462955668774601
		023-01-03	-0.574447816254458	-0.359857376775641	-0.053775189364292
		023-01-04	-0.435467882359113	-0.352631346419884	-0.0613178241303107
		023-01-05	-0.62980839589897	-0.329020852758058	-0.0691902164125238
		023-01-06	-0.3057547500642	-0.308789102948202	-0.0772817013803566
		023-01-07	-0.303438049664091	-0.315159062606137	-0.0858246179298339
		023-01-08	-0.302049133929584	-0.27267756997733	-0.0917839755718258
		023-01-09	-0.140136903463455	-0.203188155282436	-0.0988437731166593
		023-01-10	-0.126703354659494	-0.14829146797072	-0.102044519342818

```sql
-- Cumulative Returns and moving averages
DROP TABLE cumulative_returns;
SELECT
    d.Symbol,
    d.[Date],
    (((s.shares_bought * d.Close_Price)-s.initial_investment)/@Investment)*100 AS [Cumulative Return],
    AVG((((s.shares_bought * d.Close_Price)-s.initial_investment)/@Investment)*100) OVER (
        PARTITION BY d.Symbol
        ORDER BY Date
        ROWS BETWEEN 5 PRECEDING AND 4 FOLLOWING
    ) AS moving_avg_10_day,
    AVG((((s.shares_bought * d.Close_Price)-s.initial_investment)/@Investment)*100) OVER (
        PARTITION BY d.Symbol
        ORDER BY Date
        ROWS BETWEEN 50 PRECEDING AND 49 FOLLOWING
    ) AS moving_avg_100_day
INTO cumulative_returns
FROM stockdata d
LEFT JOIN stocks s ON s.symbol = d.symbol
```

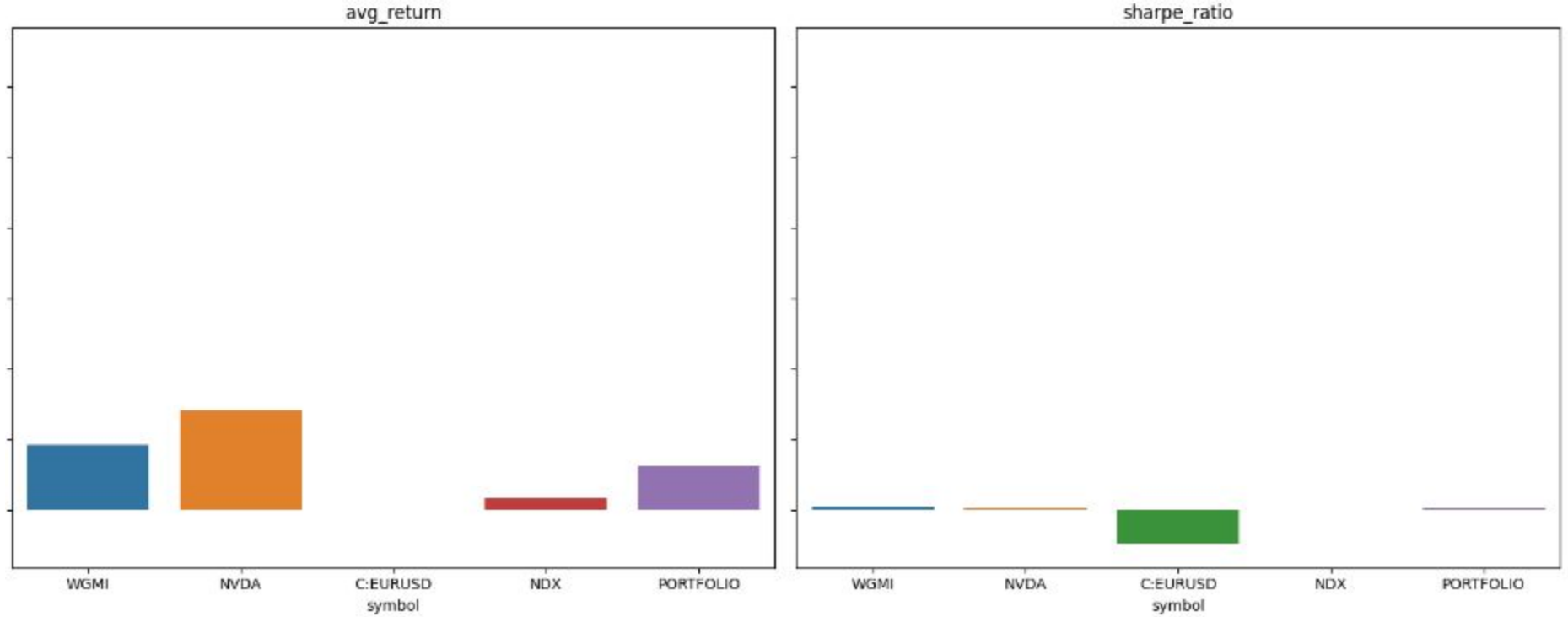
Results
Overall Portfolio



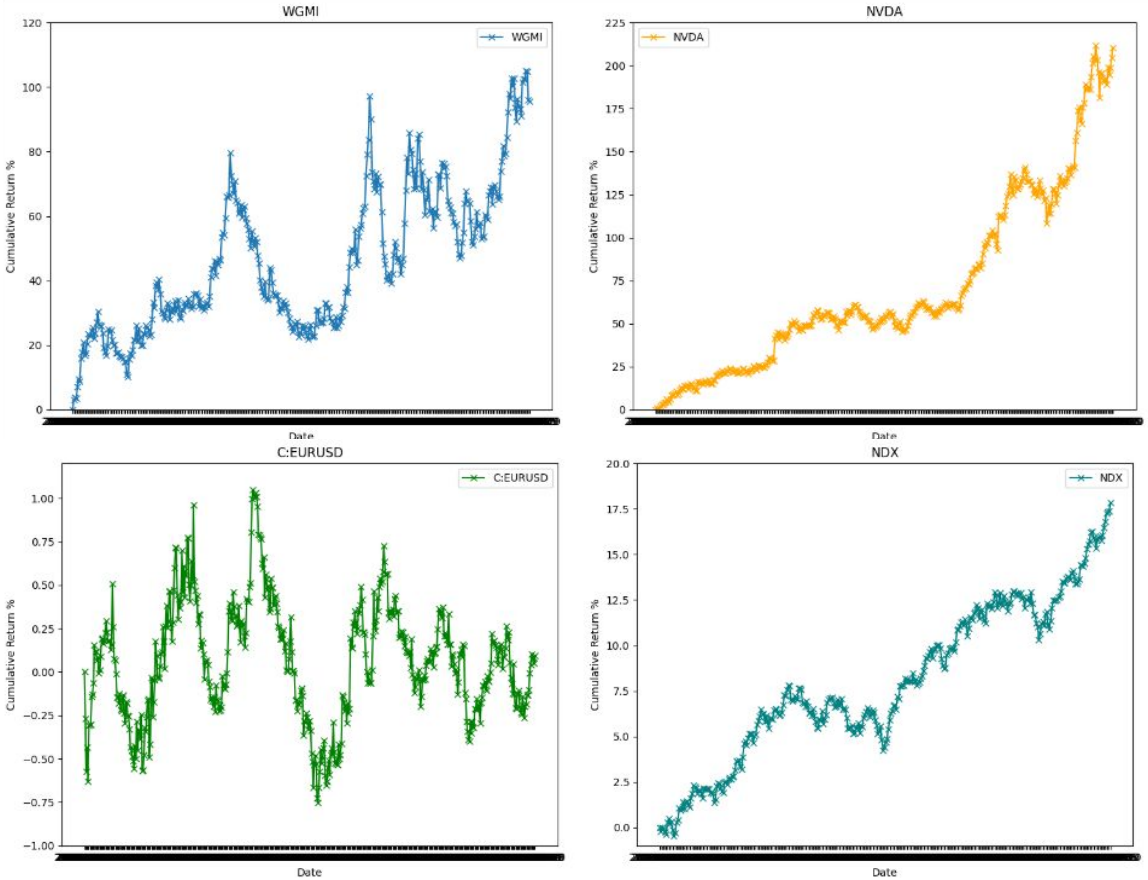
Results (Visualized w/ Seaborn)
Flat Values



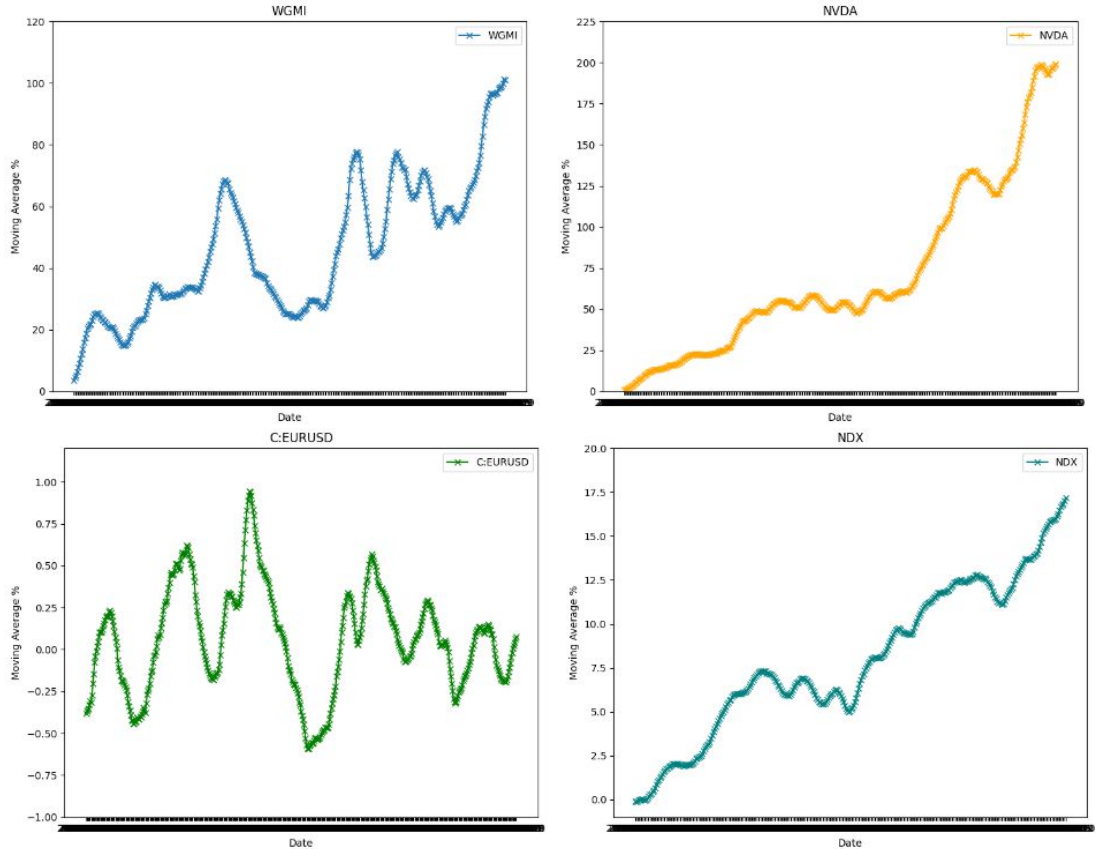
Results
Flat Values



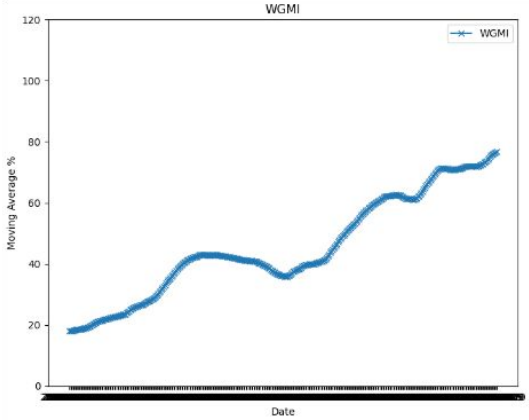
Results
Cumulative Values

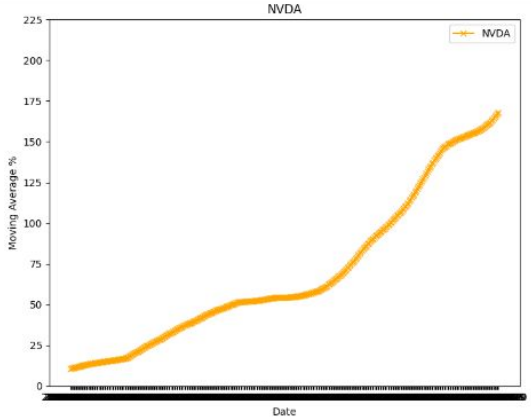


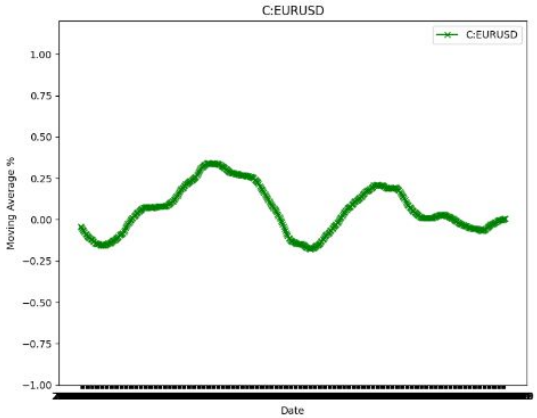
Results
Moving 10 Day Average

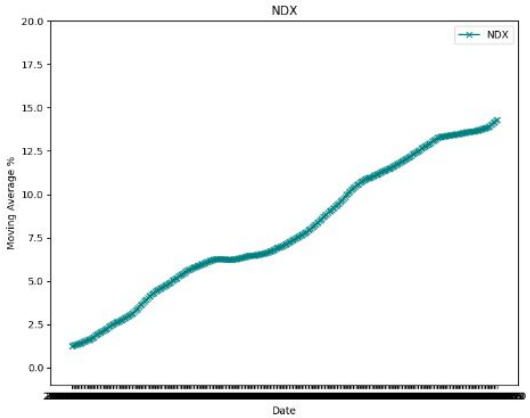


Results
Moving 100 Day Average

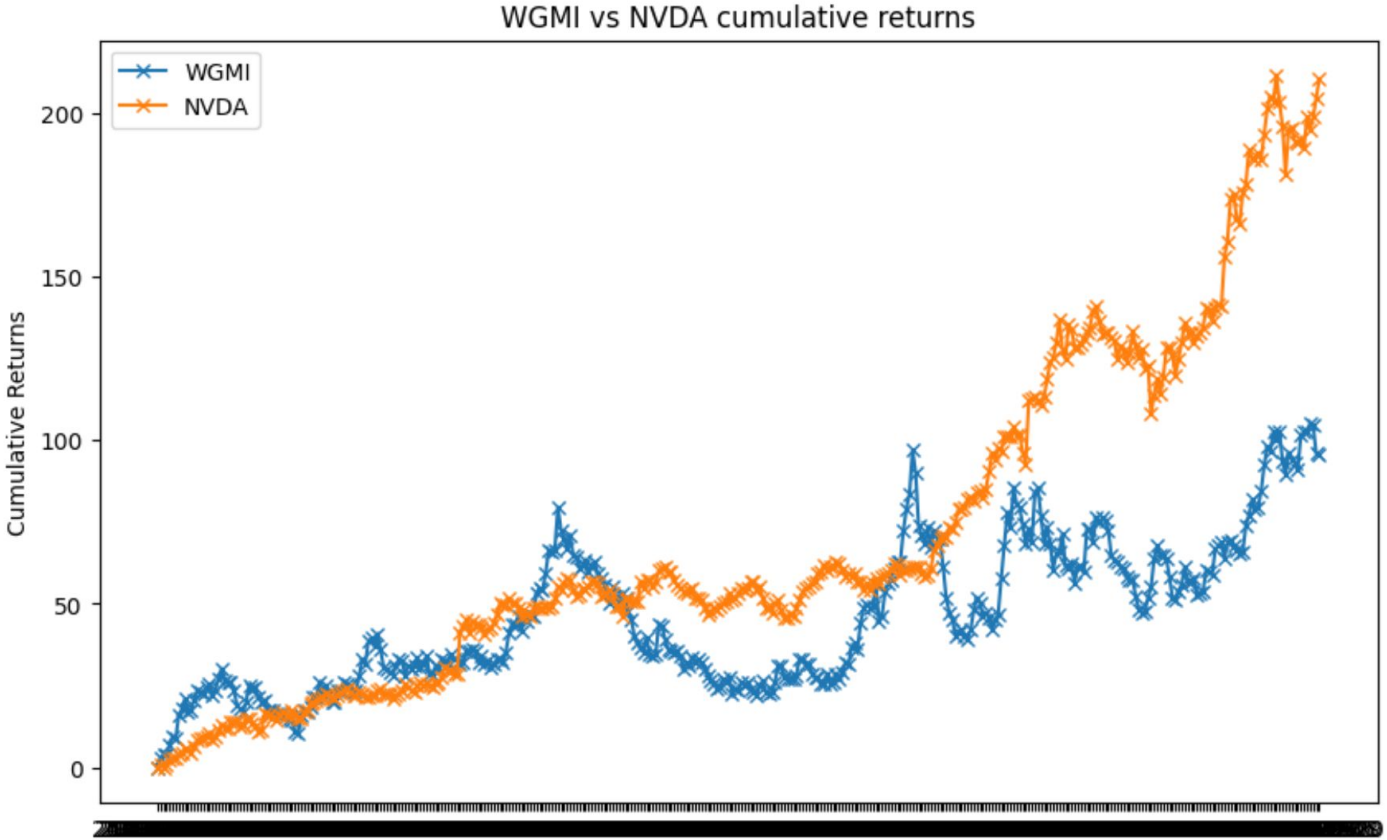




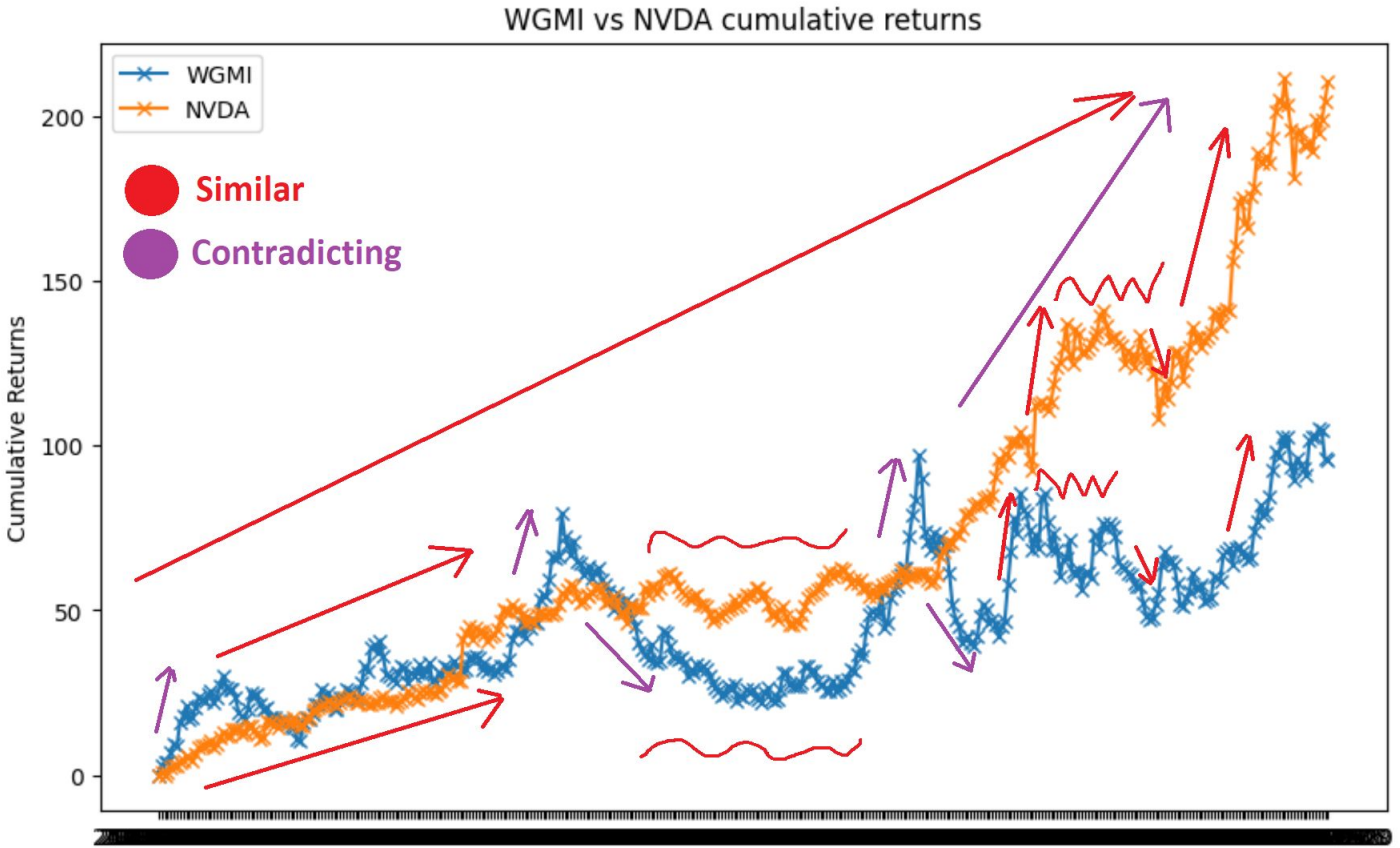




Results
NVDA vs WGMI



Results
NVDA vs WGMI – Annotated



Conclusion - Hypothesis

Contradictions:

- WGMI seems more unstable than NVDA (more drastic rises/drops)
- NVDA had a steadier increase
- NVDA ended much higher

Similarities:

- Increasing in same direction
- Same “flat spots,” “sawtooth spots”
- Very similar sharp increases

Conclusion:

There is a correlation between the two, but there is less trust in Bitcoin than in NVIDIA. NVIDIA has other factors that causes it to do well.

Conclusion - Overall

- NVDA Performed exceptionally well
- WGMI also performed well
- C:EURUSD was unreliable and barely profited
- The overall market did well over the time period

Improvements

- When transforming dataframes, combine them all first

```
# Convert Unix to datetime
etf_df['Date'] = pd.to_datetime(etf_df['Date'], unit="ms")
nvda_df['Date'] = pd.to_datetime(nvda_df['Date'], unit="ms")
forex_df['Date'] = pd.to_datetime(forex_df['Date'], unit="ms")
index_df['Date'] = pd.to_datetime(index_df['Date'], unit="ms")
```

```
# Combine all 4 dataframes
df = pd.concat([etf_df, nvda_df, forex_df, index_df], i
```

- Better modularity for swapping stocks or adding more (store API urls in a list and query all of them)

```
-- Buying Price of ETF
DECLARE @etf_buyprice FLOAT;
SELECT TOP 1 @etf_buyprice = stockdata.Close_Price
FROM stockdata
WHERE stockdata.Symbol = 'WGMI'
ORDER BY Date ASC;
```

```
# Construct the API URLs
wgmi_url = f"https://api.polygon.io/v2/aggs/ticker/WGMI/ran
nvda_url = f"https://api.polygon.io/v2/aggs/ticker/NVDA/ran
forex_url = f"https://api.polygon.io/v2/aggs/ticker/C:EURUS
index_url = f"https://api.polygon.io/v2/aggs/ticker/I:NDX/r

# Make requests to API
etf_data = requests.get(wgmi_url).json()
nvda_data = requests.get(nvda_url).json()
forex_data = requests.get(forex_url).json()
index_data = requests.get(index_url).json()
```

shares_bought
5483.899007
1746.419891

- Couldn't get Beta values working
- Shares shouldn't be decimals

Questions?