Python Polars Portfolio ——Nvidia Stock Research

Table of Contents

1. Introduction

This is Jia (Julia)LIAO NVIDA Nvidia Stock Research portfolio. The main purpose of this portfolio is to demonstrate my coding skill related to the Python Polars library. The entire case focuses on Nvidia's stock data and determine for factors that may affect stock prices to support possible future stock predictions.

In this case, I used time series stock data from yfinance, Google Trends data from pytrends, and propose four questions as following:
1. Long-term Trend: Over the past ten years, what kind of trend has NVIDIA's stock price shown? (Interval detailed to every day)
2. Short-term Trend: What significant trend changes have occurred in the stock price over the past 60 days? (Interval detailed to every 15 minutes)
3. The impact of brand popularity and topic discussion: Is there any correlation between NVIDIA's stock performance and Google Trends?
4. Comparison with Industry Trends: Compared to other companies in the same industry, such as AMD and Intel, how does NVIDIA's stock price trend?

This description document will present the key parts of the code and display the key results of code execution in the form of screenshots, with the aim of providing a convenience quickly view for everyone on any device without code running environment.

2. Long-term Trend: Over the past ten years, what kind of trend has NVIDIA's stock price shown? (Interval detailed to every day)

2.1 Fetch NVIDIA stock price data in the past decade from yfinance

```
import yfinance as yf
import polars as pl
import datetime

# Define time range from 10 years ago to today
end =   datetime.datetime.now()
start = end - datetime.timedelta(days=10*365)

# Define the ticker symbol, set the company name as a variable here for future code reuse
ticker = "NVDA"

# Obtain Nvidia stock data
df_pandas = yf.download(ticker, start=start, end=end)

# Convert Pandas Dataframe to Polars Dataframe
df_polars = pl.DataFrame(df_pandas.reset_index())

# print Polars DataFrame
print(df_polars)
```

```
[**********************100%%**********************]  1 of 1 completedshape: (2_515, 7)
┌─────────────────────┬────────────┬────────────┬────────────┬────────────┬────────────┬───────────┐
│ Date                ┆ Open       ┆ High       ┆ Low        ┆ Close      ┆ Adj Close  ┆ Volume    │
│ ---                 ┆ ---        ┆ ---        ┆ ---        ┆ ---        ┆ ---        ┆ ---       │
│ datetime[ns]        ┆ f64        ┆ f64        ┆ f64        ┆ f64        ┆ f64        ┆ i64       │
╞═════════════════════╪════════════╪════════════╪════════════╪════════════╪════════════╪═══════════╡
│ 2014-08-11 00:00:00 ┆ 0.4755     ┆ 0.4775     ┆ 0.47125    ┆ 0.4725     ┆ 0.449791   ┆ 344624000 │
│ 2014-08-12 00:00:00 ┆ 0.47225    ┆ 0.475      ┆ 0.46725    ┆ 0.4725     ┆ 0.449791   ┆ 296152000 │
│ 2014-08-13 00:00:00 ┆ 0.47325    ┆ 0.47925    ┆ 0.47025    ┆ 0.47525    ┆ 0.452408   ┆ 256596000 │
│ 2014-08-14 00:00:00 ┆ 0.477      ┆ 0.477      ┆ 0.468      ┆ 0.47       ┆ 0.447411   ┆ 255992000 │
│ …                   ┆ …          ┆ …          ┆ …          ┆ …          ┆ …          ┆ …         │
│ 2024-08-02 00:00:00 ┆ 103.760002 ┆ 108.720001 ┆ 101.370003 ┆ 107.269997 ┆ 107.269997 ┆ 482027500 │
│ 2024-08-05 00:00:00 ┆ 92.059998  ┆ 103.410004 ┆ 90.690002  ┆ 100.449997 ┆ 100.449997 ┆ 552842400 │
│ 2024-08-06 00:00:00 ┆ 103.839996 ┆ 107.709999 ┆ 100.550003 ┆ 104.25     ┆ 104.25     ┆ 409012100 │
│ 2024-08-07 00:00:00 ┆ 107.809998 ┆ 108.800003 ┆ 98.690002  ┆ 98.910004  ┆ 98.910004  ┆ 408658700 │
└─────────────────────┴────────────┴────────────┴────────────┴────────────┴────────────┴───────────┘
```

2.2 Calculate the moving averages with polars function

```
# Calculate the moving averages with polars library function
df_polars = df_polars.with_columns(
    [
        df_polars['Close'].rolling_mean(window_size=30, min_periods=1).alias('30_day_MA'),
        df_polars['Close'].rolling_mean(window_size=180, min_periods=1).alias('180_day_MA')
    ]
)
```

2.3 Comparison of two data visualization codes

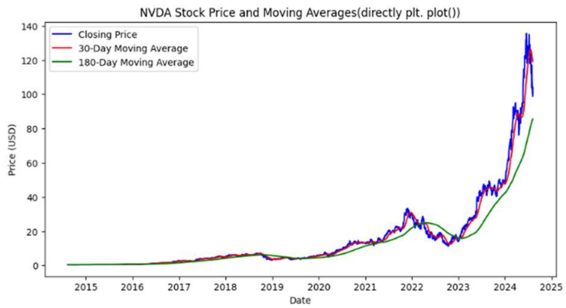
2.3.1 Plotting with normal functions, directly plt. plot()

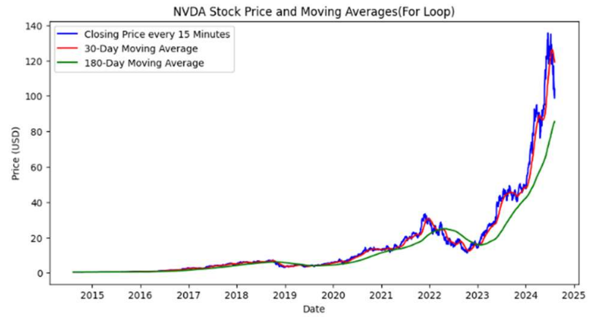
```
    # Plotting the moving averages along with the closing prices with common data visualization functions
# Traditional way, suitable for small group data
plt.figure(figsize=(10, 5))
plt.plot(df_polars['Date'], df_polars['Close'], label='Closing Price', color='blue')
plt.plot(df_polars['Date'], df_polars['30_day_MA'], label='30-Day Moving Average', color='red')
plt.plot(df_polars['Date'], df_polars['180_day_MA'], label='180-Day Moving Average', color='green')
plt.title(str(ticker)   +' Stock Price and Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.show()
```

2.3.2 Plotting by using For Loop

```
# Plotting the close prices with for Loop
# Columns to plot and their labels
columns_to_plot = [
    ('Close', 'Closing Price every 15 Minutes', 'blue'),
    ('30_day_MA', '30-Day Moving Average', 'red'),
    ('180_day_MA', '180-Day Moving Average', 'green')
]

plt.figure(figsize=(10, 5))

# Plot each column using a loop
for column, label, color in columns_to_plot:
    plt.plot(df_polars['Date'], df_polars[column], label=label, color=color)

plt.title(f'{ticker} Stock Price and Moving Averages(For Loop)')
plt.xlabel('Date')
```

```
plt.ylabel('Price (USD)')
plt.legend()
plt.show()
```





The outcome line chart is look like same but For loop coding way has higher scalability than plt.plot()

2.4 Discoveries on Long-term Trend: NVIDIA's stock price trend over the past ten years

Over the past ten years, the analysis of NVIDIA's stock prices and trading volumes has revealed the following key highlights:

- The long-term upward trend highlighted by the 180 day moving average (green line) suggests that NVIDIA has been a strong performer over the past decade.
- But the 30-day moving average (red line) shows a downward trend, indicating that there has been pullback in the stock price recently.
- Overall, there is a significant upward trend, especially noticeable from around 2020 onwards.

3. Short-term Trend: What significant trend changes have occurred in the stock price over the past 60 days? (Interval detailed to every 15 minutes)

3.1 Fetch NVIDIA stock price data in past 60 day from yfinance (Interval every 15 minutes)

```
# Define time range 60 days
end = datetime.datetime.now()
start = end - datetime.timedelta(days=60)

# Obtain Nvidia stock data interval every 15minute
df_pandas10min = yf.download(ticker, start=start, end=end, interval='15m')


# Convert Pandas Dataframe to Polars Dataframe
df_polars15min = pl.DataFrame(df_pandas10min.reset_index())

# print Polars DataFrame
print(df_polars15min)
```

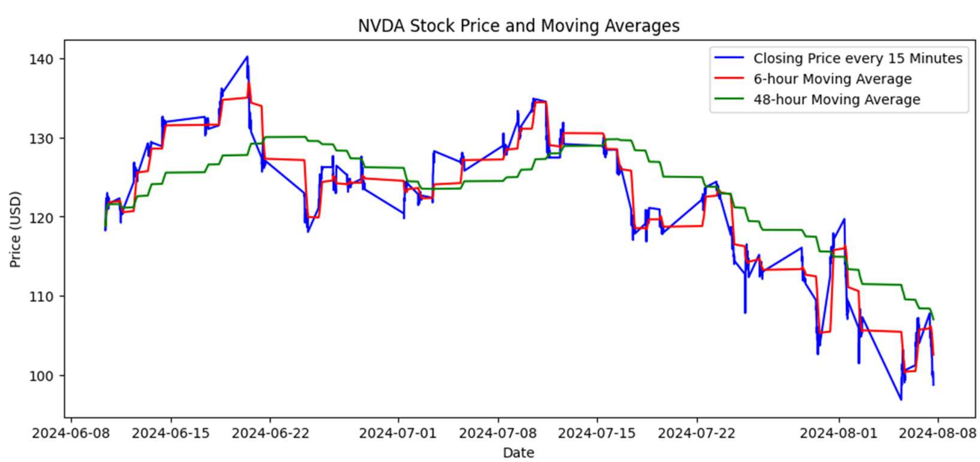
3.2 Calculate the moving averages with polars function

```
# Calculate the moving averages with polars library function
```

```
df_polars15min = df_polars15min.with_columns(
    [
        df_polars15min['Close'].rolling_mean(window_size=24, min_periods=1).alias('6_hour_MA'),
        df_polars15min['Close'].rolling_mean(window_size=192, min_periods=1).alias('48_hour_MA'),
    ]
)
print(df_polars15min)
```

3.3 Plotting with For Loop

```
# Plotting the closing prices with for Loop
# Columns to plot and their labels
columns_to_plot = [
    ('Close', 'Closing Price every 15 Minutes', 'blue'),
    ('6_hour_MA', '6-hour Moving Average', 'red'),
    ('48_hour_MA', '48-hour Moving Average', 'green')
]

plt.figure(figsize=(12, 5))

# Plot each column using a loop
for column, label, color in columns_to_plot:
    plt.plot(df_polars15min['Datetime'], df_polars15min[column], label=label, color=color)

plt.title(f'{ticker} Stock Price and Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.show()
```



3.4 Discoveries on short-term Trend and significant trend changes have occurred in the stock price over the past 60 day

- The stock exhibits significant short-term volatility, with frequent sharp rises and falls. This is evident from the jagged nature of the blue line representing the closing prices.
- Several trend reversals are evident in the chart, where the stock price changes direction. These are captured by both moving averages, with the 6-hour moving average showing quicker responses to these changes.

4. The impact of brand popularity and topic discussion:Is there any correlation between NVIDIA's stock performance and Google Trends?

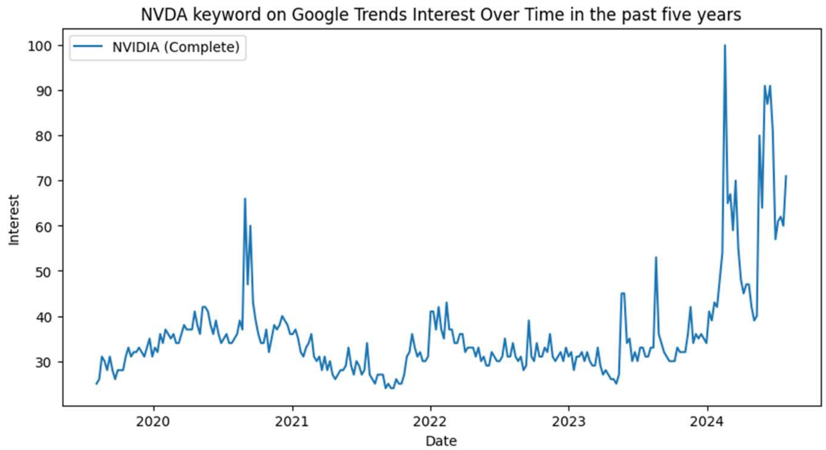
4.1 Fetch NVIDIA Google Trend data in the past 5 years from pytrends

```
rom pytrends.request import TrendReq
import time

# Initialize pytrends request with US English locale and UTC+6(aka360) timezone
pytrends = TrendReq(hl='en-US', tz=360)

# Define the keywords and timeframe( )
keywords = ["NVIDIA"]
timeframe = 'today 5-y'   # Last 5 years

# Build payload
pytrends.build_payload(keywords, timeframe=timeframe)

# Wait for a few seconds before making the request
time.sleep(10)   # Adjust the sleep time because without this sleep() API will reject my call.

# Get interest over time
interest_over_time_df = pytrends.interest_over_time()
# If this fails during the rerun, it is likely due to too frequent calls and due to too frequent calls from same
IP(with code 429)
# The code itself is 100% correct and I have obtained the data successful.
# Changing IP or free up memory then rerun again may solve the 429 errors,

# Convert Pandas DataFrame to Polars DataFrame
pl_df_GTrend = pl.DataFrame(interest_over_time_df.reset_index())

# Display the DataFrame
print(pl_df_GTrend)
# The values are normalized and scaled from 0 to 100.
print(pl_df_GTrend.describe())
```

```
shape: (262, 3)
┌─────────────────────┬────────┬───────────┐
│ date                ┆ NVIDIA ┆ isPartial │
│ ---                 ┆ ---    ┆ ---       │
│ datetime[ns]        ┆ i64    ┆ bool      │
╞═════════════════════╪════════╪═══════════╡
│ 2019-08-04 00:00:00 ┆ 25     ┆ false     │
│ 2019-08-11 00:00:00 ┆ 26     ┆ false     │
│ 2019-08-18 00:00:00 ┆ 31     ┆ false     │
│ 2019-08-25 00:00:00 ┆ 30     ┆ false     │
│ …                   ┆ …      ┆ …         │
│ 2024-07-14 00:00:00 ┆ 62     ┆ false     │
│ 2024-07-21 00:00:00 ┆ 60     ┆ false     │
│ 2024-07-28 00:00:00 ┆ 71     ┆ false     │
│ 2024-08-04 00:00:00 ┆ 84     ┆ true      │
└─────────────────────┴────────┴───────────┘
shape: (9, 4)
┌────────────┬─────────────────────┬───────────┬───────────┐
│ describe   ┆ date                ┆ NVIDIA    ┆ isPartial │
│ ---        ┆ ---                 ┆ ---       ┆ ---       │
│ str        ┆ str                 ┆ f64       ┆ str       │
╞════════════╪═════════════════════╪═══════════╪═══════════╡
│ count      ┆ 262                 ┆ 262.0     ┆ 262       │
│ null_count ┆ 0                   ┆ 0.0       ┆ 0         │
│ mean       ┆ null                ┆ 36.206107 ┆ null      │
│ std        ┆ null                ┆ 11.90236  ┆ null      │
│ min        ┆ 2019-08-04 00:00:00 ┆ 24.0      ┆ False     │
│ 25%        ┆ null                ┆ 30.0      ┆ null      │
│ 50%        ┆ null                ┆ 33.0      ┆ null      │
│ 75%        ┆ null                ┆ 37.0      ┆ null      │
│ max        ┆ 2024-08-04 00:00:00 ┆ 100.0     ┆ True      │
└────────────┴─────────────────────┴───────────┴───────────┘
```

4.2 Pick only complete data in Google Trend

```
 # Separate pick only complete data (Partial data refers to incomplete cycles, which can affect the accuracy of the data)
pl_df_GTrend_5y_c = pl_df_GTrend.filter(pl_df_GTrend['isPartial'] == False)

# Rename the first column of pl_df_GTrend_5y_c 'date' to 'Date'
pl_df_GTrend_5y_c = pl_df_GTrend_5y_c.rename({"date": "Date"})
# Plot ponly the complete data
plt.figure(figsize=(10, 5))
for keyword in keywords:
  plt.plot(pl_df_GTrend_5y_c['Date'], pl_df_GTrend_5y_c[keyword], label=f'{keyword} (Complete)', linestyle='-')

plt.title(str(ticker)+' keyword on Google Trends Interest Over Time in the past five years')
plt.xlabel('Date')
plt.ylabel('Interest')
plt.legend()
plt.show()
```



4.3 Get the complete Google trend period then fetch NVIDIA' stock data as the same time period

```
# find the start and end date of pl_df_GTrend_5y_c to Obtain Nvidia stock data with same range time with
Google Trends data
start_date = pl_df_GTrend_5y_c['Date'].min()
end_date = pl_df_GTrend_5y_c['Date'].max()
ticker = "NVDA"
print("Start Date:", start_date )
print("End Date:", end_date)

# Obtain Nvidia stock data with same range time with Google Trends data
df_pandas_Nvi_5y = yf.download(ticker, start=start_date, end=end_date)

# Convert Pandas Dataframe to Polars Dataframe
df_polars_Nvi_5y = pl.DataFrame(df_pandas_Nvi_5y.reset_index())

# print Polars DataFrame
print(df_polars_Nvi_5y)
```

4.4 Comparison of Two kind of Normalize Coding

4.4.1 Normalize NVIDA stock columns by using For Loop

```
# for loop normalize 'Close' and 'Volume' columns.
# more efficient and easily scalable way

for column in ['Close', 'Volume']:
    min_value = df_polars_Nvi_5y[column].min()
    max_value = df_polars_Nvi_5y[column].max()
    df_polars_Nvi_5y = df_polars_Nvi_5y.with_columns(
        ( (df_polars_Nvi_5y[column] - min_value) / (max_value - min_value) *
100).alias(f'Normalized_{column}')
    )

print(df_polars_Nvi_5y.dtypes)
print(df_polars_Nvi_5y.describe())
```

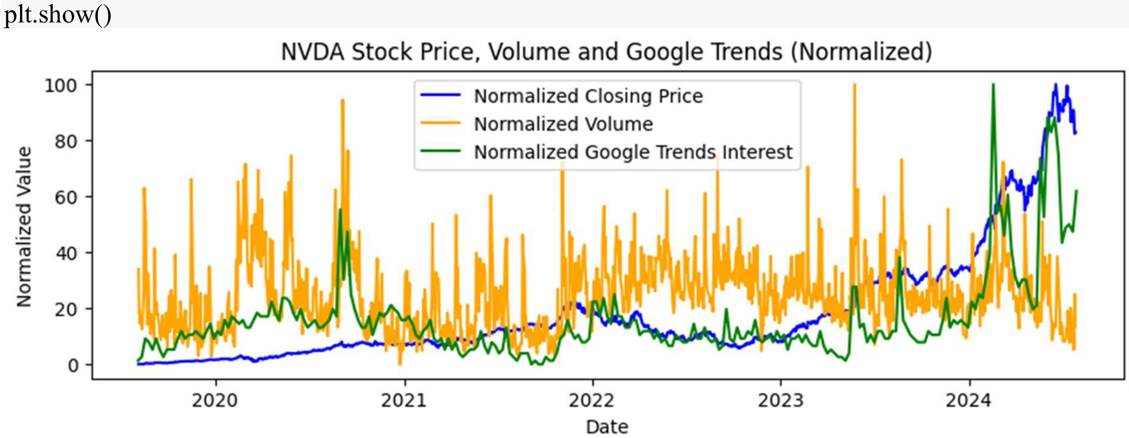
4.4.2 Normalize Google Trend with normal way

```
# Normalize 'NVIDIA' prices to a 0-100 scale to make it can compare to Trends Interest data
```

```
# Traditional way, suitable for small group data or single column
# Although this Trend metadata is Normalized, in order to compare trends, I need to Normalize again to match the range of Normalized_Close and Normalized_Volume   values
min_close = pl_df_GTrend_5y_c['NVIDIA'].min()
max_close = pl_df_GTrend_5y_c['NVIDIA'].max()

Normalized_pl_df_GTrend_5y_c = pl_df_GTrend_5y_c.with_columns(
    ( (pl_df_GTrend_5y_c['NVIDIA'] - min_close) / (max_close - min_close) * 100).alias('Normalized_NVIDIA')
)
print(Normalized_pl_df_GTrend_5y_c.dtypes)
print(Normalized_pl_df_GTrend_5y_c)
print(Normalized_pl_df_GTrend_5y_c.describe())
```

4.5 Plotting Data from different data frames by using For Loop

```
# Plotting the Normalized data with for Loop
# Columns to plot and their labels
columns_to_plot = [
    ('Normalized_Close', 'Normalized Closing Price', 'blue'),
    ('Normalized_Volume', 'Normalized Volume', 'orange'),
    ('Normalized_NVIDIA', 'Normalized Google Trends Interest', 'green')
]

plt.figure(figsize=(10, 3))

# Plot each column using a loop
for column, label, color in columns_to_plot:
    # Check if the column exists in the DataFrame
    if column in Normalized_pl_df_GTrend_5y_c.columns:
        plt.plot(Normalized_pl_df_GTrend_5y_c['Date'], Normalized_pl_df_GTrend_5y_c[column], label=label, color=color)
    elif column in df_polars_Nvi_5y.columns:
        plt.plot(df_polars_Nvi_5y['Date'], df_polars_Nvi_5y[column], label=label, color=color)

plt.title(f'{ticker} Stock Price, Volume and Google Trends (Normalized)')
plt.xlabel('Date')
plt.ylabel('Normalized Value')
plt.legend()
plt.show()
```



4.6 Interpolation Null value data to conduct correlation analysis

4.6.1 Expand the Google Trend 5 years DataFrame date range from by week to by day

```
# Create a complete date range DataFrame on Normalized_pl_df_GTrend_5y_c
date_range = pl.DataFrame({"Date": pd.date_range(start=Normalized_pl_df_GTrend_5y_c['Date'].min(),
                                                  end=Normalized_pl_df_GTrend_5y_c['Date'].max(),
                                                  freq='D')})

# Join the original DataFrame with the complete date range, ensuring no dates are out of the original range
Normalized_pl_df_GTrend_5y_c_full = date_range.join(Normalized_pl_df_GTrend_5y_c, on='Date',
how='left')


# Display the DataFrame with the complete date range to verify
print(Normalized_pl_df_GTrend_5y_c_full)
```

```
┌─────────────────────┬────────┬───────────┬───────────────────┐
│ Date                ┆ NVIDIA ┆ isPartial ┆ Normalized_NVIDIA │
│ ---                 ┆ ---    ┆ ---       ┆ ---               │
│ datetime[ns]        ┆ i64    ┆ bool      ┆ f64               │
╞═════════════════════╪════════╪═══════════╪═══════════════════╡
│ 2019-08-04 00:00:00 ┆ 25     ┆ false     ┆ 1.315789          │
│ 2019-08-05 00:00:00 ┆ null   ┆ null      ┆ null              │
│ 2019-08-06 00:00:00 ┆ null   ┆ null      ┆ null              │
│ 2019-08-07 00:00:00 ┆ null   ┆ null      ┆ null              │
│ …                   ┆ …      ┆ …         ┆ …                 │
│ 2024-07-25 00:00:00 ┆ null   ┆ null      ┆ null              │
│ 2024-07-26 00:00:00 ┆ null   ┆ null      ┆ null              │
│ 2024-07-27 00:00:00 ┆ null   ┆ null      ┆ null              │
│ 2024-07-28 00:00:00 ┆ 71     ┆ false     ┆ 61.842105         │
└─────────────────────┴────────┴───────────┴───────────────────┘
```

4.6.2 Interpolate data with interpolate()

```
#   linear interpolation (in Pandas)
Normalized_pl_df_GTrend_5y_c_full = Normalized_pl_df_GTrend_5y_c_full.with_columns(
    pl.col("Normalized_NVIDIA").interpolate()
)

print(Normalized_pl_df_GTrend_5y_c_full)
print(Normalized_pl_df_GTrend_5y_c_full.describe())
```

Date --- datetime[ns]	NVIDIA --- i64	isPartial --- bool	Normalized_NVIDIA --- f64
2019-08-04 00:00:00	25	false	1.315789
2019-08-05 00:00:00	null	null	1.503759
2019-08-06 00:00:00	null	null	1.691729
2019-08-07 00:00:00	null	null	1.879699
…	…	…	…
2024-07-25 00:00:00	null	null	55.639098
2024-07-26 00:00:00	null	null	57.706767
2024-07-27 00:00:00	null	null	59.774436
2024-07-28 00:00:00	71	false	61.842105

shape: (9, 5)

describe --- str	Date --- str	NVIDIA --- f64	isPartial --- str	Normalized_NVIDIA --- f64
count	1821	261.0	261	1821.0
null_count	0	1560.0	1560	0.0
mean	null	36.022989	null	15.767797
std	null	11.549536	null	14.636925
min	2019-08-04 00:00:00	24.0	False	0.0
25%	null	30.0	null	8.458647
50%	null	33.0	null	11.654135
75%	null	37.0	null	17.105263
max	2024-07-28 00:00:00	100.0	False	100.0

4.7 Merge dataframes to plot by for Loop

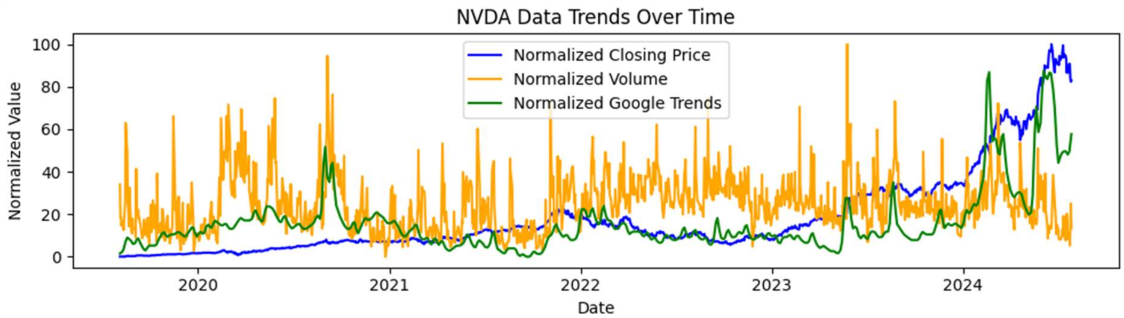
4.7.1 Merge the two dataframes

```
# Merge the two dataframes
joined_df_polars_Nvi_G_5y = df_polars_Nvi_5y.join(Normalized_pl_df_GTrend_5y_c_full.select(['Date',
'Normalized_NVIDIA']), left_on='Date', right_on='Date', how='inner')

print(joined_df_polars_Nvi_G_5y.columns)
```

4.7.2 Plot by For Loop

```
# Plot by for Loop
# Columns to plot and their labels
columns_to_plot = [
    ('Normalized_Close', 'Normalized Closing Price', 'blue'),
    ('Normalized_Volume', 'Normalized Volume', 'orange'),
    ('Normalized_NVIDIA', 'Normalized Google Trends', 'green')
]

plt.figure(figsize=(10, 3))

# Plot each column using a loop
for column, label, color in columns_to_plot:
    plt.plot(joined_df_polars_Nvi_G_5y['Date'].to_list(), joined_df_polars_Nvi_G_5y[column].to_list(),
label=label, color=color)
```

```
plt.title(f'{ticker} Data Trends Over Time')
plt.xlabel('Date')
plt.ylabel('Normalized Value')
plt.legend()
plt.tight_layout()
plt.show()
```



4.8 Correlation matrix by directly .corr()

```
# Calculate the correlation matrix in Normalized_Close, Normalized_Volume
and   Interpolated_Normalized_NVIDIA

correlation_matrix = joined_df_polars_Nvi_G_5y[['Normalized_Close', 'Normalized_Volume',
'Normalized_NVIDIA']].corr()

# Display the correlation matrix
print(correlation_matrix)
```

Normalized_Close --- f64	Normalized_Volume --- f64	Normalized_NVIDIA --- f64
1.0	-0.002092	0.73827
-0.002092	1.0	0.161244
0.73827	0.161244	1.0

4.9 Discoveries on data correlation between NVIDIA's stock performance and Google Trends

- Close price & Volume: Interpretation: The correlation between the Close price and volume is nearly zero (-0.002). This indicates there is no significant linear relationship between the stock's closing price and its trading volume. Changes in the closing price do not correspond to consistent changes in the trading volume and vice versa.

- Volume & NVIDIA Google Trend: Interpretation: The correlation between the volume and the Google search interest for "NVIDIA" is weakly positive (0.1591). While there is a positive relationship, it is not strong. This suggests that increased public interest in NVIDIA, as indicated by search trends, might lead to a slight increase in trading volume, but the relationship is not strong enough to be considered significant.

- Close price & NVIDIA Google Trend: There is a strong positive correlation between the Close price and the Google search interest for "NVIDIA" (0.74). This suggests that as public interest in NVIDIA increases, as indicated by Google Trends, the stock's closing price also tends to rise. This strong correlation indicates that search interest could potentially be a indicator which could be taken into consideration of when doing stock price prediction.

5. Comparison with Industry Trends: Compared to other companies in the same industry (such as AMD and Intel), how does NVIDIA's stock price trend?

5.1 Fetch multiple stocks data by using for loop and store in dictionary

```
import yfinance as yf
import polars as pl
import pandas as pd
import datetime

# Define time range from 60d ago to today
end =   datetime.datetime.now()
start = end - datetime.timedelta(days=60)


# Define the ticker symbols and fetch data using yfinance
tickers = ["NVDA", "AMD", "INTC"]
interval = '2m'

# Initialize an empty dictionary
data_pd = {}

# Fetch the data by using for loop
for ticker in tickers:
    data_pd[ticker] = yf.download(ticker, start=start, interval=interval)


# Print the dictionary keys (ticker symbols)
print(data_pd.keys())

# For loop print each ticker
for ticker in tickers:
    print(f"{ticker}")
    print(data_pd[ticker])
```

5.2 Calculate and plot moving averages by using For loop

5.2.1 Plotting moving averages data visualization from different data frames in library by using For Loop with finance stock graphic

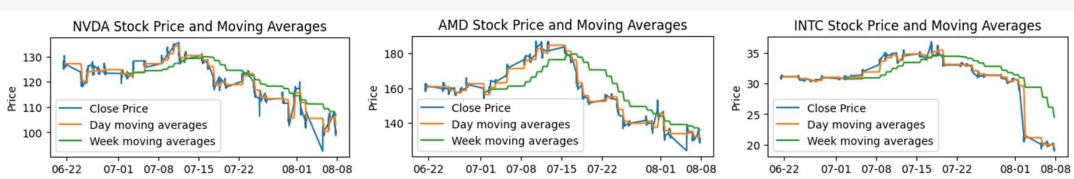
```
# Retrieve the first row of all three ticker DataFrames and extract the YYYY-MM-DD portion of the 'Datetime' column
for ticker, df in data_pl.items():
  first_row = df.head(1)
  first_date   = first_row['Datetime'][0].strftime('%Y-%m-%d')
  print(f" {ticker} {first_date}")
```

```
NVDA 2024-06-21
AMD 2024-06-21
INTC 2024-06-21
```

```
# Count rows containing first_date in each ticker DataFrame
for ticker, df in data_pl.items():
  first_row = df.head(1)
  first_date  = first_row['Datetime'][0].strftime('%Y-%m-%d')
  count = df.filter(pl.col('Datetime').dt.strftime('%Y-%m-%d') == first_date).shape[0]
  print(f"Number of rows in {ticker} containing {first_date}: {count}")
```

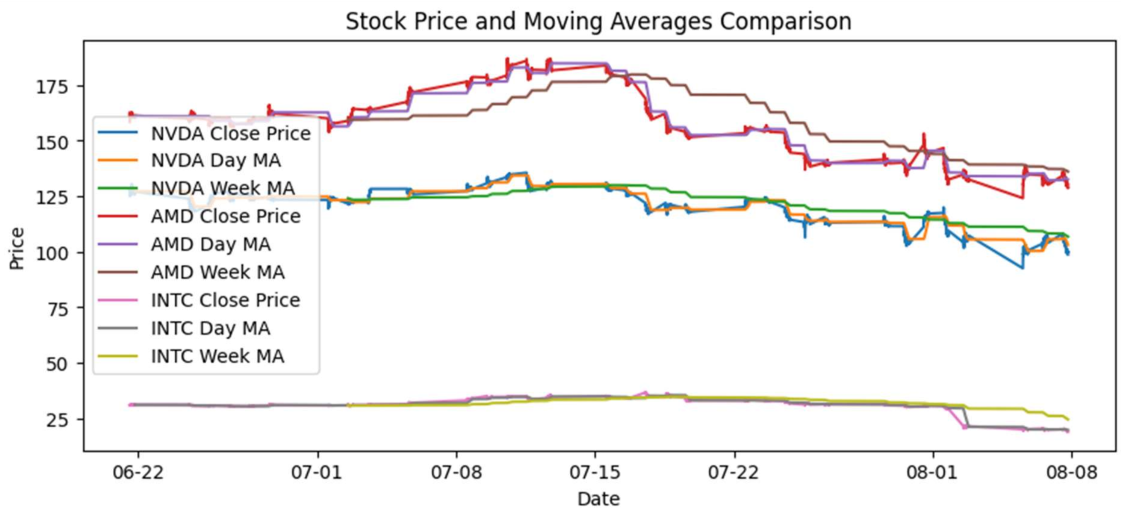
```
Number of rows in NVDA containing 2024-06-21: 195
Number of rows in AMD containing 2024-06-21: 195
Number of rows in INTC containing 2024-06-21: 195
```

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Calculate moving averages
for ticker, df in data_pl.items():
    df = df.with_columns([
        pl.col('Close').rolling_mean(window_size=count).alias(f'day_MA'),
        pl.col('Volume').rolling_mean(window_size=count).alias(f'day_V'),
        pl.col('Close').rolling_mean(window_size=7*count).alias(f'week_MA'),
        pl.col('Volume').rolling_mean(window_size=7*count).alias(f'week_V')

    ])
    data_pl[ticker] = df

# Plot moving averages


for ticker, df in data_pl.items():
    plt.figure(figsize=(5, 2))
    plt.plot(df['Datetime'], df['Close'], label='Close Price')
    plt.plot(df['Datetime'], df['day_MA'], label='Day moving averages')
    plt.plot(df['Datetime'], df['week_MA'], label='Week moving averages')
    plt.title(f"{ticker} Stock Price and Moving Averages")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()

    # Format x-axis to display only MM-DD
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m-%d'))

    plt.show()
```



5.2.2 Plot all tickers on the same plot for comparison

```
# Plot all tickers on the same plot for comparison
plt.figure(figsize=(10, 4))

for ticker, df in data_pl.items():
    plt.plot(df['Datetime'], df['Close'], label=f'{ticker} Close Price')
    plt.plot(df['Datetime'], df['day_MA'], label=f'{ticker} Day MA')
    plt.plot(df['Datetime'], df['week_MA'], label=f'{ticker} Week MA')

plt.title("Stock Price and Moving Averages Comparison")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()

# Format x-axis to display only MM-DD
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m-%d'))

plt.show()
```

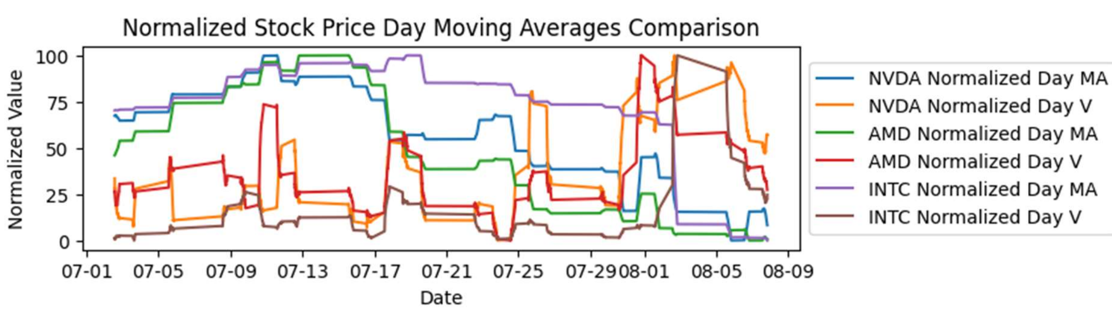


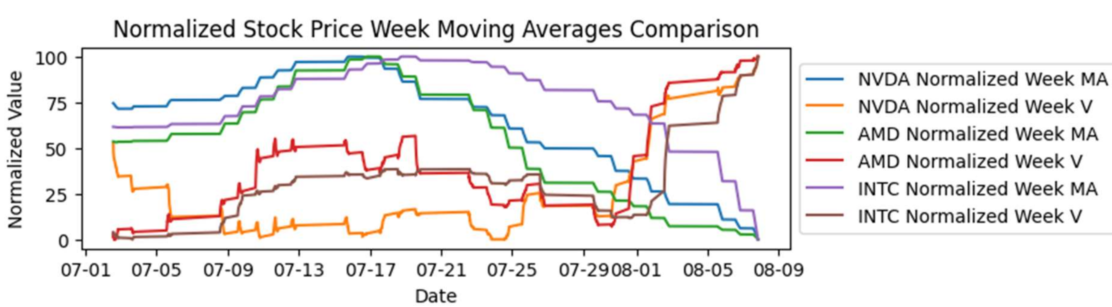
5.3 Normalize columns for each ticker then plot all tickers on the same plot for comparison

```
# Remove rows with null values in any column
for ticker, df in data_pl.items():
  data_pl[ticker] = df.drop_nulls()

# Print the updated DataFrames
for ticker, df in data_pl.items():
  print(f"{ticker}\n{df.head()}")
```

```
# Normalize the columns for each ticker
for ticker, df in data_pl.items():
    for column in ['Close','Volume','day_V', 'week_V','day_MA', 'week_MA']:
        min_value = df[column].min()
        max_value = df[column].max()
        df = df.with_columns(
            ((df[column] - min_value) / (max_value - min_value) * 100).alias(f'Normalized_{column}')
        )
    data_pl[ticker] = df
```

```
# Plot the normalized moving averages for all tickers on the same plot
plt_d = plt
plt_d.figure(figsize=(7, 2))

for ticker, df in data_pl.items():
    plt_d.plot(df['Datetime'], df['Normalized_day_MA'], label=f'{ticker} Normalized Day MA')
    plt_d.plot(df['Datetime'], df['Normalized_day_V'], label=f'{ticker} Normalized Day V')

plt_d.title("Normalized Stock Price Day Moving Averages Comparison")
plt_d.xlabel("Date")
plt_d.ylabel("Normalized Value")


# Position the legend to the right of the plot
plt_d.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt_d.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m-%d'))
plt_d.show()

plt_w = plt

plt_w.figure(figsize=(7, 2))

for ticker, df in data_pl.items():
    plt_w.plot(df['Datetime'], df['Normalized_week_MA'], label=f'{ticker} Normalized Week MA')
    plt_w.plot(df['Datetime'], df['Normalized_week_V'], label=f'{ticker} Normalized Week V')
plt_w.title("Normalized Stock Price Week Moving Averages Comparison")
plt_w.xlabel("Date")
plt_w.ylabel("Normalized Value")

# Position the legend to the right of the plot
plt_w.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt_w.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m-%d'))

plt_w.show()
```





5.4 Correlation analysis by using For Loop .corr()

```
# Define the metrics to analyze
metrics = ['Normalized_day_MA', 'Normalized_Volume']

# Loop over metrics to extract columns and calculate correlation matrices


for metric in metrics:
    data_dict = {f"{ticker.lower()}_{metric.split('_')[1].lower()}": data_pl[ticker][metric] for ticker in tickers}
    correlation_matrix = pd.DataFrame(data_dict).corr()
    print(f"\n Correlation matrix for {metric} in the past 60 days :")
    print(correlation_matrix)
```

```
 Correlation matrix for Normalized_day_MA in the past 60 days :
          nvda_day   amd_day  intc_day
nvda_day  1.000000  0.949410  0.809662
amd_day   0.949410  1.000000  0.720698
intc_day  0.809662  0.720698  1.000000

 Correlation matrix for Normalized_Volume in the past 60 days :
             nvda_volume  amd_volume  intc_volume
nvda_volume     1.000000    0.776994     0.638200
amd_volume      0.776994    1.000000     0.560696
intc_volume     0.638200    0.560696     1.000000
```

5.5 Discoveries on comparison with industry:

The correlation matrix for stock prices moving averages day trends (in the past 60 days) indicates a high correlation between NVIDIA and AMD (0.949) or Intel (0.810).

The correlation matrix for trading volumes (in the past 60 days) indicates a medium correlation between NVIDIA and AMD (0.452) and low correlations between NVIDIA and Intel (0.352).

NVIDIA's stock price shows a very strong positive correlation with AMD's stock price and trading volumes trends medium correlations. These suggests that the two companies' stock prices tend to move together closely, reflecting similar market dynamics and investor sentiment within the business areas sector which NVIDIA and AMD are highly overlapping (for example GPU) .and high correlation implies that factors affecting AMD's stock, such as technological advancements, market trends, or economic conditions, are likely to have a similar impact on NVIDIA's stock in recent times.

NVIDIA's stock price also shows a strong positive correlation with Intel's stock price, although slightly lower than with AMD and low correlations in trading volumes trends. These indicates the business area that are Intel involving but AMD not involving should not be given attention when considering as factor and impact on NVIDIA's stock price recently, although, in these business factors, NVIDIA and Intel are both involving still should not be considered as important factors that affect the stock price in the past 60 days (for example 5G, IoT, autonomous driving)

Further factors detail needs to be determined in conjunction with business analysis report.