AML_Project_Test_Code (1)

July 24, 2024

1 BUAN 6341 Project 1 - Group 5

1.1 Group Members

- **Siva Srinivas Narra** (SXN230069)
- **Prashanth Chowdary** (PXY230011)
- **Tarun Raghu** (TXR230002)

---

2 Project Overview

2.1 Goal

Predict NVIDIA stock price will go up or down using historical prices, technical indicators, economic indicators, and company financials etc.

2.2 Why NVIDIA?

NVIDIA is a leader in: - **Gaming:** Cutting-edge GPUs. - **AI & Machine Learning:** Pioneering advancements. - **Data Centers:** Powering cloud computing and big data.

2.3 Financial Performance

NVIDIA shows strong revenue growth and solid profitability, making it an ideal subject for comprehensive analysis.

2.3.1 Objectives:

- Analyze historical price trends.
- Utilize technical indicators.
- Examine economic indicators.
- Evaluate company financials.

Join us in exploring NVIDIA, a technological and market leader in the semiconductor industry.

2.4 Step 1: Data Collection

2.4.1 Importing Hourly Stock Data for NVIDIA (NVDA)

In this section, we will import the hourly stock data for NVIDIA (ticker: NVDA) from December 10,2020 to July 22, 2024, using raw data files. The data will be stored in a DataFrame named `nvda_stock_data`.

[ ]:
```python
from google.colab import files
import pandas as pd

# Load the data
nvda_stock_data = pd.read_csv('NVDA_intraday_data_adjusted.csv')

# Add 'nvda_' prefix to all column names
nvda_stock_data.columns = ['nvda_' + col for col in nvda_stock_data.columns]

# Display the first few rows of the updated DataFrame
print("\nNVIDIA Stock Data:")
nvda_stock_data
```

```
NVIDIA Stock Data:
```

[ ]:
```
          nvda_datetime   nvda_open   nvda_high    nvda_low  nvda_close  \
0      10/12/2020 13:30   13.989500   14.175000   13.912500   14.080792
1      10/12/2020 14:30   14.083000   14.190500   14.030251   14.129033
2      10/12/2020 15:30   14.130930   14.228999   14.100999   14.227374
3      10/12/2020 16:30   14.229500   14.260750   14.191499   14.254750
4      10/12/2020 17:30   14.265375   14.345750   14.229500   14.343750
…                   …         …         …         …         …
7233    7/22/2024 16:30  121.775001  122.949996  121.540000  122.800003
7234    7/22/2024 17:30  122.809997  124.069999  122.599998  123.514999
7235    7/22/2024 18:30  123.517501  123.750000  122.610000  122.839996
7236    7/22/2024 19:30  122.864997  123.750000  122.709999  123.739997
7237    7/22/2024 20:00  123.540000  123.540000  123.540000  123.540000

       nvda_volume
0      132333280.0
1       50193760.0
2       37900160.0
3       40635000.0
4       44107080.0
…              …
7233    20145140.0
7234    27192892.0
7235    25875275.0
7236    20012085.0
7237           NaN
```

```
[7238 rows x 6 columns]
```

2.4.2 Analyzing the Competitive Impact on NVIDIA's Stock Price

NVIDIA's stock price is significantly influenced by its competitive environment. Key competitors such as Intel, AMD, Google, and Qualcomm can impact NVIDIA's market share and investor sentiment through their performance and innovation. Therefore, it's important for investors to closely monitor these companies.

In this section, we retrieve and display the hourly stock data for each competitor. To facilitate identification, each column in the data is prefixed with the company name. This allows for easy differentiation between the data of various competitors.

[ ]:
```python
import pandas as pd

# Load the data
intel_data = pd.read_csv('INTEL_intraday_data_adjusted.csv')

# Add 'intel_' prefix to all column names
intel_data.columns = ['intel_' + col for col in intel_data.columns]

# Display the first few rows of the updated DataFrame
print("\nINTEL Stock Data:")
intel_data
```

```
INTEL Stock Data:
```

[ ]:
```
          intel_datetime  intel_open  intel_high  intel_low  intel_close  \
0     10/12/2020 13:30   53.549999   53.619998  53.209999    53.349998
1     10/12/2020 14:30   53.345001   53.665000  53.279998    53.580001
2     10/12/2020 15:30   53.574199   53.799999  53.540000    53.775001
3     10/12/2020 16:30   53.770000   54.119998  53.764999    54.090000
4     10/12/2020 17:30   54.139999   54.169998  53.979999    54.150001
…                  …          …          …          …            …
7231   7/22/2024 16:30   32.895000   33.119998  32.880001    32.994998
7232   7/22/2024 17:30   33.000000   33.145000  32.950000    33.130001
7233   7/22/2024 18:30   33.125598   33.359001  33.103599    33.314998
7234   7/22/2024 19:30   33.310001   33.409999  33.270000    33.369998
7235   7/22/2024 20:00   33.369998   33.369998  33.369998    33.369998

      intel_volume
0        4553973.0
1        2931119.0
2        2562218.0
3        3400834.0
```

```
4        2794134.0
…             …
7231     2768152.0
7232     3075312.0
7233     3858738.0
7234     6012255.0
7235           NaN

[7236 rows x 6 columns]
```

[ ]:
```python
import pandas as pd

# Load the data
amd_data = pd.read_csv('AMD_intraday_data_adjusted.csv')

# Add 'amd_' prefix to all column names
amd_data.columns = ['amd_' + col for col in amd_data.columns]

# Display the first few rows of the updated DataFrame
print("\nAMD Stock Data:")
amd_data
```

```
AMD Stock Data:
```

[ ]:
```
           amd_datetime    amd_open    amd_high     amd_low   amd_close  \
0      10/12/2020 13:30   83.650001   84.940002   83.120002   84.769996
1      10/12/2020 14:30   84.778800   85.129997   84.569999   84.949996
2      10/12/2020 15:30   84.955001   84.970100   84.150001   84.589996
3      10/12/2020 16:30   84.589996   84.699996   84.230003   84.500000
4      10/12/2020 17:30   84.510002   84.720001   84.379997   84.535003
…                   …         …           …           …           …
7231    7/22/2024 16:30  154.250000  155.569900  154.182998  155.050003
7232    7/22/2024 17:30  155.065002  155.679992  154.380004  155.488403
7233    7/22/2024 18:30  155.460006  155.750000  154.350006  155.320007
7234    7/22/2024 19:30  155.349899  156.059997  155.059997  155.869995
7235    7/22/2024 20:00  155.869995  155.869995  155.869995  155.869995

       amd_volume
0      17685351.0
1       6286127.0
2       5267087.0
3       4053806.0
4       3385459.0
…             …
7231    3598134.0
```

```
7232    3121564.0
7233    3867858.0
7234    4326745.0
7235          NaN

[7236 rows x 6 columns]
```

[ ]:
```python
import pandas as pd

# Load the data
qcom_data = pd.read_csv('QCOM_intraday_data_adjusted.csv')

# Add 'qcom_' prefix to all column names
qcom_data.columns = ['qcom_' + col for col in qcom_data.columns]

# Display the first few rows of the updated DataFrame
print("\nQCOM Stock Data:")
qcom_data
```

```
QCOM Stock Data:
```

[ ]:
```
         qcom_datetime   qcom_open   qcom_high    qcom_low  qcom_close  \
0     10/12/2020 13:30  127.699600  127.699600  124.952301  125.190002
1     10/12/2020 14:30  125.175003  126.540000  124.989997  126.099998
2     10/12/2020 15:30  126.110000  126.970001  126.000999  126.900001
3     10/12/2020 16:30  126.879997  127.529998  126.459999  127.199996
4     10/12/2020 17:30  127.150001  127.519996  126.779998  127.507102
…                  …         …           …           …         …
7232   7/22/2024 16:30  192.065002  193.669998  191.779998  193.220001
7233   7/22/2024 17:30  193.225006  194.360000  193.199996  193.919998
7234   7/22/2024 18:30  193.949996  194.537597  193.199996  194.205001
7235   7/22/2024 19:30  194.210006  195.500000  193.859802  194.970001
7236   7/22/2024 20:00  194.970001  194.970001  194.970001  194.970001

      qcom_volume
0       1463335.0
1       1278993.0
2        749534.0
3        737437.0
4       1112964.0
…             …
7232     445975.0
7233     571279.0
7234     676729.0
7235    1162651.0
```

```
7236          NaN

[7237 rows x 6 columns]
```

```
[ ]: import pandas as pd

     # Load the data
     google_data = pd.read_csv('GOOGL_intraday_data_adjusted.csv')

     # Add 'google_' prefix to all column names
     google_data.columns = ['google_' + col for col in google_data.columns]

     # Display the first few rows of the updated DataFrame
     print("\nGOOGLE Stock Data:")
     google_data
```

```
GOOGLE Stock Data:
```

```
[ ]:       google_datetime  google_open  google_high  google_low  google_close  \
     0     10/12/2020 13:30    76.900000    77.378003   76.465002     77.226752
     1     10/12/2020 14:30    77.200000    77.820001   76.953497     77.757001
     2     10/12/2020 15:30    77.692499    78.108563   77.621503     78.108563
     3     10/12/2020 16:30    78.122498    78.716663   77.994000     78.585498
     4     10/12/2020 17:30    78.648999    79.414502   78.555499     79.346997
     …                   …          …          …          …          …
     7232   7/22/2024 16:30   181.410003   182.610000  181.410003    182.350006
     7233   7/22/2024 17:30   182.360000   182.619995  181.964996    182.399993
     7234   7/22/2024 18:30   182.399993   182.449996  181.964996    182.274993
     7235   7/22/2024 19:30   182.279998   182.699996  181.600006    181.639999
     7236   7/22/2024 20:00   181.669998   181.669998  181.669998    181.669998

           google_volume
     0         9434280.0
     1         5777900.0
     2         5775520.0
     3         4340740.0
     4         6191680.0
     …               …
     7232      1690215.0
     7233      1660425.0
     7234      2211049.0
     7235      3430312.0
     7236            NaN

     [7237 rows x 6 columns]
```

2.4.3 Combining Competitor Data with NVIDIA Stock Data

This section merges the stock data of Intel, AMD, Qualcomm, and Google with NVIDIA's stock data. The merge is performed using a left join on the `Datetime` column, appending each company's data horizontally to NVIDIA's data.

[ ]:
```
# Convert columns to datetime format
nvda_stock_data['nvda_datetime'] = pd.
    ↪to_datetime(nvda_stock_data['nvda_datetime'])
intel_data['intel_datetime'] = pd.to_datetime(intel_data['intel_datetime'])
amd_data['amd_datetime'] = pd.to_datetime(amd_data['amd_datetime'])
qcom_data['qcom_datetime'] = pd.to_datetime(qcom_data['qcom_datetime'])
google_data['google_datetime'] = pd.to_datetime(google_data['google_datetime'])

# Remove timezone information
nvda_stock_data['nvda_datetime'] = nvda_stock_data['nvda_datetime'].dt.
    ↪tz_localize(None)
intel_data['intel_datetime'] = intel_data['intel_datetime'].dt.tz_localize(None)
amd_data['amd_datetime'] = amd_data['amd_datetime'].dt.tz_localize(None)
qcom_data['qcom_datetime'] = qcom_data['qcom_datetime'].dt.tz_localize(None)
google_data['google_datetime'] = google_data['google_datetime'].dt.
    ↪tz_localize(None)

# Rename datetime columns to 'Datetime' for merging
nvda_stock_data.rename(columns={'nvda_datetime': 'Datetime'}, inplace=True)
intel_data.rename(columns={'intel_datetime': 'Datetime'}, inplace=True)
amd_data.rename(columns={'amd_datetime': 'Datetime'}, inplace=True)
qcom_data.rename(columns={'qcom_datetime': 'Datetime'}, inplace=True)
google_data.rename(columns={'google_datetime': 'Datetime'}, inplace=True)

# Merge the DataFrames
merged_data = nvda_stock_data.merge(intel_data, on='Datetime', how='left',␣
    ↪suffixes=('', '_Intel'))
merged_data = merged_data.merge(amd_data, on='Datetime', how='left',␣
    ↪suffixes=('', '_AMD'))
merged_data = merged_data.merge(qcom_data, on='Datetime', how='left',␣
    ↪suffixes=('', '_Qualcomm'))
merged_data = merged_data.merge(google_data, on='Datetime', how='left',␣
    ↪suffixes=('', '_Google'))

# Display the first few rows of the merged data
print("\nMerged Data:")
merged_data
```

```
Merged Data:
```

```
[ ]:                Datetime   nvda_open   nvda_high    nvda_low  nvda_close  \
     0    2020-10-12 13:30:00   13.989500   14.175000   13.912500   14.080792
     1    2020-10-12 14:30:00   14.083000   14.190500   14.030251   14.129033
     2    2020-10-12 15:30:00   14.130930   14.228999   14.100999   14.227374
     3    2020-10-12 16:30:00   14.229500   14.260750   14.191499   14.254750
     4    2020-10-12 17:30:00   14.265375   14.345750   14.229500   14.343750
     …                    …           …           …           …           …
     7233 2024-07-22 16:30:00  121.775001  122.949996  121.540000  122.800003
     7234 2024-07-22 17:30:00  122.809997  124.069999  122.599998  123.514999
     7235 2024-07-22 18:30:00  123.517501  123.750000  122.610000  122.839996
     7236 2024-07-22 19:30:00  122.864997  123.750000  122.709999  123.739997
     7237 2024-07-22 20:00:00  123.540000  123.540000  123.540000  123.540000

           nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
     0     132333280.0   53.549999   53.619998  53.209999    53.349998  …
     1      50193760.0   53.345001   53.665000  53.279998    53.580001  …
     2      37900160.0   53.574199   53.799999  53.540000    53.775001  …
     3      40635000.0   53.770000   54.119998  53.764999    54.090000  …
     4      44107080.0   54.139999   54.169998  53.979999    54.150001  …
     …               …           …           …          …            …  …
     7233   20145140.0   32.895000   33.119998  32.880001    32.994998  …
     7234   27192892.0   33.000000   33.145000  32.950000    33.130001  …
     7235   25875275.0   33.125598   33.359001  33.103599    33.314998  …
     7236   20012085.0   33.310001   33.409999  33.270000    33.369998  …
     7237          NaN   33.369998   33.369998  33.369998    33.369998  …

            qcom_open   qcom_high    qcom_low  qcom_close  qcom_volume  \
     0     127.699600  127.699600  124.952301  125.190002    1463335.0
     1     125.175003  126.540000  124.989997  126.099998    1278993.0
     2     126.110000  126.970001  126.000999  126.900001     749534.0
     3     126.879997  127.529998  126.459999  127.199996     737437.0
     4     127.150001  127.519996  126.779998  127.507102    1112964.0
     …              …           …           …           …            …
     7233  192.065002  193.669998  191.779998  193.220001     445975.0
     7234  193.225006  194.360000  193.199996  193.919998     571279.0
     7235  193.949996  194.537597  193.199996  194.205001     676729.0
     7236  194.210006  195.500000  193.859802  194.970001    1162651.0
     7237  194.970001  194.970001  194.970001  194.970001          NaN

           google_open  google_high  google_low  google_close  google_volume
     0       76.900000    77.378003   76.465002     77.226752      9434280.0
     1       77.200000    77.820001   76.953497     77.757001      5777900.0
     2       77.692499    78.108563   77.621503     78.108563      5775520.0
     3       78.122498    78.716663   77.994000     78.585498      4340740.0
     4       78.648999    79.414502   78.555499     79.346997      6191680.0
     …               …            …           …             …              …
     7233   181.410003   182.610000  181.410003    182.350006      1690215.0
```

```
7234   182.360000   182.619995  181.964996    182.399993       1660425.0
7235   182.399993   182.449996  181.964996    182.274993       2211049.0
7236   182.279998   182.699996  181.600006    181.639999       3430312.0
7237   181.669998   181.669998  181.669998    181.669998             NaN

[7238 rows x 26 columns]
```

2.4.4 Applying Key Technical Indicators to Stock Data

In this section, we calculate and add key technical indicators to the `stock_data` DataFrame to enhance stock price analysis. Below is an overview of some of the indicators used:

1. **Moving Averages (SMA and EMA)**
   - **Simple Moving Average (SMA):** Computes the average closing price over a specified period (e.g., 20 days). SMA helps smooth out price data to identify overall trends.
   - **Exponential Moving Average (EMA):** Calculates a weighted average of the closing price, giving more importance to recent prices. EMA responds more quickly to price changes compared to SMA, highlighting recent trends.
2. **Moving Average Convergence Divergence (MACD)**
   - **Description:** Computes the MACD line and the MACD signal line. The MACD helps identify changes in trend strength, direction, momentum, and duration. It provides signals for potential buy or sell opportunities.
3. **Relative Strength Index (RSI)**
   - **Description:** Calculates the RSI over a specified period (e.g., 14 days). RSI measures the speed and change of price movements to identify overbought or oversold conditions, indicating potential reversal points.
4. **Bollinger Bands**
   - **Description:** Uses the Simple Moving Average (SMA) and calculates two outer bands at a specified number of standard deviations from the SMA. Bollinger Bands help assess market volatility and identify potential price reversals by showing the range in which prices typically move.

These indicators along with other indicators are integrated into the `nvda_stock_data` DataFrame to provide insights into price movements, trends, and volatility. Utilizing these technical indicators helps in making more informed trading decisions and understanding the stock's performance better.

[ ]:
```
!pip install pandas_ta
import pandas_ta as ta
import pandas as pd

# Simple Moving Average (SMA) over a 20-day period
merged_data['NVDA_SMA_20'] = ta.sma(merged_data['nvda_close'], length=20)

# Exponential Moving Average (EMA) over a 20-day period
merged_data['NVDA_EMA_20'] = ta.ema(merged_data['nvda_close'], length=20)

# Moving Average Convergence Divergence (MACD)
```

```
merged_data['NVDA_MACD'], merged_data['NVDA_MACD_signal'], _ = ta.
 ↪macd(merged_data['nvda_close'])

# Relative Strength Index (RSI) over a 14-day period
merged_data['NVDA_RSI'] = ta.rsi(merged_data['nvda_close'], length=14)

# Bollinger Bands
bbands = ta.bbands(merged_data['nvda_close'])
bbands.columns = [f'NVDA_{col}' for col in bbands.columns]
merged_data = pd.concat([merged_data, bbands], axis=1)

# Average True Range (ATR)
merged_data['NVDA_ATR'] = ta.atr(merged_data['nvda_high'],␣
 ↪merged_data['nvda_low'], merged_data['nvda_close'])

# On-Balance Volume (OBV)
merged_data['NVDA_OBV'] = ta.obv(merged_data['nvda_close'],␣
 ↪merged_data['nvda_volume'])

# Stochastic Oscillator (Stoch)
stoch_data = ta.stoch(merged_data['nvda_high'], merged_data['nvda_low'],␣
 ↪merged_data['nvda_close'])
stoch_data.columns = [f'NVDA_{col}' for col in stoch_data.columns]
merged_data = pd.concat([merged_data, stoch_data], axis=1)

envelope_percentage = 2 / 100

# Calculate the upper and lower envelopes
merged_data['NVDA_EMA_Upper'] = merged_data['NVDA_EMA_20'] * (1 +␣
 ↪envelope_percentage)
merged_data['NVDA_EMA_Lower'] = merged_data['NVDA_EMA_20'] * (1 -␣
 ↪envelope_percentage)

# Calculate Money Flow Multiplier
merged_data['MFM'] = ((merged_data['nvda_close'] - merged_data['nvda_low']) -␣
 ↪(merged_data['nvda_high'] - merged_data['nvda_close'])) /␣
 ↪(merged_data['nvda_high'] - merged_data['nvda_high'])

# Calculate Money Flow Volume
merged_data['MFV'] = merged_data['MFM'] * merged_data['nvda_volume']

# Calculate CMF for a specific period (e.g., 20 days)
period = 20
merged_data['NVDA_CMF'] = merged_data['MFV'].rolling(window=period).sum() /␣
 ↪merged_data['nvda_volume'].rolling(window=period).sum()
```

```
# Drop intermediate columns
merged_data.drop(columns=['MFM', 'MFV'], inplace=True)

# Calculate the Typical Price
merged_data['Typical_Price'] = (merged_data['nvda_high'] +␣
 ↪merged_data['nvda_low'] + merged_data['nvda_close']) / 3

# Calculate the VWAP
merged_data['Cumulative_TP_Volume'] = (merged_data['Typical_Price'] *␣
 ↪merged_data['nvda_volume']).cumsum()
merged_data['Cumulative_Volume'] = merged_data['nvda_volume'].cumsum()
merged_data['NVDA_VWAP'] = merged_data['Cumulative_TP_Volume'] /␣
 ↪merged_data['Cumulative_Volume']

# Drop intermediate columns
merged_data.drop(columns=['Typical_Price', 'Cumulative_TP_Volume',␣
 ↪'Cumulative_Volume'], inplace=True)

# Create 'Date' column containing only date information
merged_data['Date'] = merged_data['Datetime'].dt.date

merged_data
```

```
Requirement already satisfied: pandas_ta in /usr/local/lib/python3.10/dist-
packages (0.3.14b0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from pandas_ta) (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->pandas_ta) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->pandas_ta) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->pandas_ta) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-
packages (from pandas->pandas_ta) (1.25.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas->pandas_ta) (1.16.0)
```

[ ]:

```
                Datetime  nvda_open   nvda_high    nvda_low  nvda_close  \
0    2020-10-12 13:30:00   13.989500   14.175000   13.912500   14.080792
1    2020-10-12 14:30:00   14.083000   14.190500   14.030251   14.129033
2    2020-10-12 15:30:00   14.130930   14.228999   14.100999   14.227374
3    2020-10-12 16:30:00   14.229500   14.260750   14.191499   14.254750
4    2020-10-12 17:30:00   14.265375   14.345750   14.229500   14.343750
…                    …         …          …          …          …
7233 2024-07-22 16:30:00  121.775001  122.949996  121.540000  122.800003
7234 2024-07-22 17:30:00  122.809997  124.069999  122.599998  123.514999
```

```
7235 2024-07-22 18:30:00  123.517501  123.750000  122.610000  122.839996
7236 2024-07-22 19:30:00  122.864997  123.750000  122.709999  123.739997
7237 2024-07-22 20:00:00  123.540000  123.540000  123.540000  123.540000

      nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
0     132333280.0   53.549999   53.619998  53.209999    53.349998  …
1      50193760.0   53.345001   53.665000  53.279998    53.580001  …
2      37900160.0   53.574199   53.799999  53.540000    53.775001  …
3      40635000.0   53.770000   54.119998  53.764999    54.090000  …
4      44107080.0   54.139999   54.169998  53.979999    54.150001  …
…             …         …          …          …           …  …
7233   20145140.0   32.895000   33.119998  32.880001    32.994998  …
7234   27192892.0   33.000000   33.145000  32.950000    33.130001  …
7235   25875275.0   33.125598   33.359001  33.103599    33.314998  …
7236   20012085.0   33.310001   33.409999  33.270000    33.369998  …
7237          NaN   33.369998   33.369998  33.369998    33.369998  …

      NVDA_BBP_5_2.0  NVDA_ATR      NVDA_OBV  NVDA_STOCHk_14_3_3  \
0                NaN       NaN  1.323333e+08                 NaN
1                NaN       NaN  1.825270e+08                 NaN
2                NaN       NaN  2.204272e+08                 NaN
3                NaN       NaN  2.610622e+08                 NaN
4           0.866563       NaN  3.051693e+08                 NaN
…                …        …            …                   …
7233        0.716152  1.819937  1.931697e+10           74.815584
7234        0.806222  1.794941  1.934416e+10           86.014900
7235        0.629847  1.748160  1.931829e+10           88.412135
7236        0.793107  1.697577  1.933830e+10           89.477582
7237        0.662428  1.590607           NaN           89.601965

      NVDA_STOCHd_14_3_3  NVDA_EMA_Upper  NVDA_EMA_Lower  NVDA_CMF  NVDA_VWAP  \
0                    NaN             NaN             NaN       NaN  14.056097
1                    NaN             NaN             NaN       NaN  14.072734
2                    NaN             NaN             NaN       NaN  14.092173
3                    NaN             NaN             NaN       NaN  14.114508
4                    NaN             NaN             NaN       NaN  14.142233
…                    …             …             …       …        …
7233           69.677912      122.869397      118.050990       NaN  34.001157
7234           78.646441      123.166150      118.336105       NaN  34.007301
7235           83.080873      123.369069      118.531066       NaN  34.013124
7236           87.968206      123.640090      118.791459       NaN  34.017645
7237           89.163894      123.865872      119.008387       NaN        NaN

            Date
0     2020-10-12
1     2020-10-12
2     2020-10-12
```

```
3      2020-10-12
4      2020-10-12
…             …
7233   2024-07-22
7234   2024-07-22
7235   2024-07-22
7236   2024-07-22
7237   2024-07-22

[7238 rows x 45 columns]
```

2.5 Economic Indicator Data Fetching and Analysis

This section aims to fetch key economic indicators from the Federal Reserve Economic Data (FRED) and combine them into a single DataFrame for analysis. The indicators include the Federal Funds Rate, Consumer Price Index for All Urban Consumers, Real Gross Domestic Product, and Unemployment Rate.

2.5.1 Economic Indicators Explained

1. **Federal Funds Rate (FEDFUNDS)**:
   - The interest rate at which depository institutions trade federal funds (balances held at Federal Reserve Banks) with each other overnight. This rate influences other interest rates, such as for mortgages, loans, and savings, and is a key tool used by the Federal Reserve to control monetary policy.
2. **Consumer Price Index for All Urban Consumers (CPIAUCNS)**:
   - A measure of the average change over time in the prices paid by urban consumers for a market basket of consumer goods and services. It is a widely used indicator of inflation, reflecting the cost of living and purchasing power of consumers.
3. **Real Gross Domestic Product (GDP)**:
   - The total value of all goods and services produced in a country, adjusted for inflation. It provides a comprehensive overview of economic activity and health, indicating how well the economy is performing. Real GDP is used to compare the economic performance of different periods.
4. **Unemployment Rate (UNRATE)**:
   - The percentage of the total labor force that is unemployed but actively seeking employment and willing to work. It is a key indicator of labor market health and economic stability, influencing consumer spending and economic growth.

[ ]:
```python
import pandas_datareader as pdr
import pandas as pd
from datetime import datetime

# Define the time period with correct date format
start_date = datetime(2023, 1, 1)
end_date = datetime(2024, 7, 1)
```

```
# Define the data series you want to download
data_series = {
    'FEDFUNDS': 'FEDFUNDS',           # Federal Funds Rate
    'CPIAUCNS': 'CPIAUCNS',          # Consumer Price Index for All Urban␣
 ↪Consumers
    'GDP': 'GDP',                    # Real Gross Domestic Product
    'UNRATE': 'UNRATE'          # Unemployment Rate
}

# Fetch the data
data = {}
for name, code in data_series.items():
    try:
        data[name] = pdr.get_data_fred(code, start_date, end_date)
    except Exception as e:
        print(f"Error fetching data for {name}: {e}")

# Combine all data into a single DataFrame
economic_df = pd.concat(data.values(), axis=1, keys=data.keys())

# Flatten the MultiIndex columns and rename to the desired names
economic_df.columns = [col[0] for col in economic_df.columns]

# Rename columns to the specified names
economic_df.columns = [name for name in data_series.keys()]

economic_df.reset_index(inplace=True)
economic_df['DATE'] = economic_df['DATE'].dt.strftime('%Y_%m_%d')
# Display the first few rows of the combined dataset
print("\nEconomic Indicator Data:")
economic_df.head()
```

```
Economic Indicator Data:
```

[ ]:

	DATE	FEDFUNDS	CPIAUCNS	GDP	UNRATE
0	2023_01_01	4.33	299.170	26813.601	3.4
1	2023_02_01	4.57	300.840	NaN	3.6
2	2023_03_01	4.65	301.836	NaN	3.5
3	2023_04_01	4.83	303.363	27063.012	3.4
4	2023_05_01	5.06	304.127	NaN	3.7

In the code below, the fetched economic indicator data is joined with historical stock price data at the month and year level granularity. This allows for a comprehensive analysis of how these economic indicators impact the stock price over time.

```
[ ]: # Convert 'Date' in merged_data to datetime
merged_data['Date'] = pd.to_datetime(merged_data['Date'], format='%Y_%m_%d')

# Convert 'DATE' in economic_df to datetime
economic_df['DATE'] = pd.to_datetime(economic_df['DATE'], format='%Y_%m_%d')

# Extract year and month from 'Date' column in merged_data
merged_data['Year'] = merged_data['Date'].dt.year
merged_data['Month'] = merged_data['Date'].dt.month

# Extract year and month from 'DATE' column in economic_df
economic_df['Year'] = economic_df['DATE'].dt.year
economic_df['Month'] = economic_df['DATE'].dt.month

# Merge on 'Year' and 'Month'
merged_data = merged_data.merge(economic_df, on=['Year', 'Month'], how='left')

# Display the first few rows of the final dataset
print("\nFinal Merged Data:")
merged_data
```

```
Final Merged Data:
```

```
[ ]:              Datetime   nvda_open   nvda_high    nvda_low  nvda_close  \
0    2020-10-12 13:30:00   13.989500   14.175000   13.912500   14.080792
1    2020-10-12 14:30:00   14.083000   14.190500   14.030251   14.129033
2    2020-10-12 15:30:00   14.130930   14.228999   14.100999   14.227374
3    2020-10-12 16:30:00   14.229500   14.260750   14.191499   14.254750
4    2020-10-12 17:30:00   14.265375   14.345750   14.229500   14.343750
…                    …         …         …         …         …
7233 2024-07-22 16:30:00  121.775001  122.949996  121.540000  122.800003
7234 2024-07-22 17:30:00  122.809997  124.069999  122.599998  123.514999
7235 2024-07-22 18:30:00  123.517501  123.750000  122.610000  122.839996
7236 2024-07-22 19:30:00  122.864997  123.750000  122.709999  123.739997
7237 2024-07-22 20:00:00  123.540000  123.540000  123.540000  123.540000

       nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
0      132333280.0   53.549999   53.619998  53.209999    53.349998  …
1       50193760.0   53.345001   53.665000  53.279998    53.580001  …
2       37900160.0   53.574199   53.799999  53.540000    53.775001  …
3       40635000.0   53.770000   54.119998  53.764999    54.090000  …
4       44107080.0   54.139999   54.169998  53.979999    54.150001  …
…              …          …          …          …            …  …
7233    20145140.0   32.895000   33.119998  32.880001    32.994998  …
7234    27192892.0   33.000000   33.145000  32.950000    33.130001  …
7235    25875275.0   33.125598   33.359001  33.103599    33.314998  …
7236    20012085.0   33.310001   33.409999  33.270000    33.369998  …
```

```
7237          NaN   33.369998   33.369998  33.369998    33.369998  …

      NVDA_CMF  NVDA_VWAP       Date  Year  Month DATE  FEDFUNDS  CPIAUCNS  \
0          NaN  14.056097 2020-10-12  2020     10  NaT       NaN       NaN
1          NaN  14.072734 2020-10-12  2020     10  NaT       NaN       NaN
2          NaN  14.092173 2020-10-12  2020     10  NaT       NaN       NaN
3          NaN  14.114508 2020-10-12  2020     10  NaT       NaN       NaN
4          NaN  14.142233 2020-10-12  2020     10  NaT       NaN       NaN
…          …        …         …    …     …    …        …         …
7233       NaN  34.001157 2024-07-22  2024      7  NaT       NaN       NaN
7234       NaN  34.007301 2024-07-22  2024      7  NaT       NaN       NaN
7235       NaN  34.013124 2024-07-22  2024      7  NaT       NaN       NaN
7236       NaN  34.017645 2024-07-22  2024      7  NaT       NaN       NaN
7237       NaN        NaN 2024-07-22  2024      7  NaT       NaN       NaN

      GDP  UNRATE
0     NaN     NaN
1     NaN     NaN
2     NaN     NaN
3     NaN     NaN
4     NaN     NaN
…    …      …
7233  NaN     NaN
7234  NaN     NaN
7235  NaN     NaN
7236  NaN     NaN
7237  NaN     NaN

[7238 rows x 52 columns]
```

2.5.2 Impact of the NVIDIA Income Statement on Stock Price

The income statment is crucial for stock price movements:

- **Profitability Metrics**: Strong profits and revenue growth can boost stock prices.
- **Investment Decisions**: Positive financial results attract investors and can drive up stock prices.
- **Market Perception**: Good earnings reports improve market confidence, impacting stock prices.
- **Comparative Analysis**: Comparing financial performance with peers helps investors assess stock value.

NVIDIA's quarterly income statement affects investor perceptions and stock price through its financial performance.

[ ]:
```python
import pandas as pd
import yfinance as yf
```

```
# Define the ticker symbol for NVIDIA
ticker_symbol = 'NVDA'

# Fetch NVIDIA's financial data
nvidia_data = yf.Ticker(ticker_symbol)

# Get NVIDIA's quarterly income statement
quarterly_income_statement = nvidia_data.quarterly_financials

# Reset the index to turn the row indices into a column
pivoted_income_statement = quarterly_income_statement.reset_index()

# Pivot the DataFrame to have metrics as columns
pivoted_income_statement = pivoted_income_statement.melt(id_vars='index',␣
↪var_name='Date', value_name='Amount')
pivoted_income_statement.columns = ['Metric', 'Date', 'Amount']

# Pivot the melted DataFrame so that metrics are columns
pivoted_income_statement = pivoted_income_statement.pivot_table(index='Date',␣
↪columns='Metric', values='Amount')

# Reset index to make 'Date' a column again
pivoted_income_statement.reset_index(inplace=True)

# Print the pivoted income statement
print("NVIDIA Quarterly Income Statement:")
pivoted_income_statement  # Print the first few rows
```

```
NVIDIA Quarterly Income Statement:
```

[ ]:

```
Metric        Date  Basic Average Shares  Basic EPS  Cost Of Revenue  \
0       2023-01-31         24640000000.0      0.057              NaN
1       2023-04-30         24700000000.0      0.083     2544000000.0
2       2023-07-31         24730000000.0       0.25     4045000000.0
3       2023-10-31         24680000000.0      0.375     4720000000.0
4       2024-01-31                   NaN        NaN     5312000000.0
5       2024-04-30         24620000000.0      0.604     5638000000.0

Metric  Diluted Average Shares  Diluted EPS  Diluted NI Availto Com Stockholders  \
0                24770000000.0        0.057                                  NaN
1                24900000000.0        0.082                         2043000000.0
2                24990000000.0        0.248                         6188000000.0
3                24940000000.0        0.371                         9243000000.0
4                          NaN          NaN                        12285000000.0
5                24890000000.0        0.598                        14881000000.0
```

```
Metric           EBIT          EBITDA   Gross Profit  …  \
0                 NaN             NaN            NaN  …
1        2275000000.0    2659000000.0   4648000000.0  …
2        7046000000.0    7411000000.0   9462000000.0  …
3       10585000000.0   10957000000.0  13400000000.0  …
4       14169000000.0   14556000000.0  16791000000.0  …
5       17343000000.0   17753000000.0  20406000000.0  …

Metric Selling General And Administration Special Income Charges  \
0                                     NaN                    0.0
1                             633000000.0                    0.0
2                             622000000.0                    0.0
3                             689000000.0                    0.0
4                             712000000.0                    0.0
5                             777000000.0                    NaN

Metric Tax Effect Of Unusual Items Tax Provision Tax Rate For Calcs  \
0                              NaN           NaN                 NaN
1                              0.0   166000000.0               0.075
2                              0.0   793000000.0               0.114
3                              0.0  1279000000.0               0.122
4                              0.0  1821000000.0            0.129094
5                              0.0  2398000000.0               0.139

Metric Total Expenses Total Operating Income As Reported  Total Revenue  \
0                 NaN                                NaN            NaN
1        5052000000.0                       2140000000.0   7192000000.0
2        6707000000.0                       6800000000.0  13507000000.0
3        7703000000.0                      10417000000.0  18120000000.0
4        8489000000.0                      13614000000.0  22103000000.0
5        9135000000.0                      16909000000.0  26044000000.0

Metric Total Unusual Items Total Unusual Items Excluding Goodwill
0                      0.0                                     0.0
1                      0.0                                     0.0
2                      0.0                                     0.0
3                      0.0                                     0.0
4                      0.0                                     0.0
5                      NaN                                     NaN

[6 rows x 44 columns]
```

In the code below, the fetched income statement is joined with historical stock price data at the quarter and year level granularity. This allows for a comprehensive analysis of how income statement impacts the stock price over time.

[ ]:

```python
# Convert 'Date' in merged_data to datetime
merged_data['Date'] = pd.to_datetime(merged_data['Date'], format='%Y_%m_%d')

# Convert 'DATE' in economic_df to datetime
pivoted_income_statement['Date'] = pd.
 ↪to_datetime(pivoted_income_statement['Date'], format='%Y_%m_%d')

# Extract year and quarter from 'Date' column in merged_data
merged_data['Year'] = merged_data['Date'].dt.year
merged_data['Quarter'] = merged_data['Date'].dt.quarter

# Extract year and quarter from 'Date' column in pivoted_income_statement
pivoted_income_statement['Year'] = pivoted_income_statement['Date'].dt.year
pivoted_income_statement['Quarter'] = pivoted_income_statement['Date'].dt.
 ↪quarter

# Merge on 'Year' and 'Quarter'
merged_data = merged_data.merge(pivoted_income_statement, on=['Year',␣
 ↪'Quarter'], how='left')

# Display the first few rows of the final dataset
print("\nFinal Merged Data:")
merged_data
```

```
Final Merged Data:
```

[ ]:

```
                Datetime   nvda_open   nvda_high    nvda_low  nvda_close  \
0    2020-10-12 13:30:00   13.989500   14.175000   13.912500   14.080792
1    2020-10-12 14:30:00   14.083000   14.190500   14.030251   14.129033
2    2020-10-12 15:30:00   14.130930   14.228999   14.100999   14.227374
3    2020-10-12 16:30:00   14.229500   14.260750   14.191499   14.254750
4    2020-10-12 17:30:00   14.265375   14.345750   14.229500   14.343750
…                    …         …           …           …           …
7233 2024-07-22 16:30:00  121.775001  122.949996  121.540000  122.800003
7234 2024-07-22 17:30:00  122.809997  124.069999  122.599998  123.514999
7235 2024-07-22 18:30:00  123.517501  123.750000  122.610000  122.839996
7236 2024-07-22 19:30:00  122.864997  123.750000  122.709999  123.739997
7237 2024-07-22 20:00:00  123.540000  123.540000  123.540000  123.540000

      nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
0     132333280.0   53.549999   53.619998  53.209999    53.349998  …
1      50193760.0   53.345001   53.665000  53.279998    53.580001  …
2      37900160.0   53.574199   53.799999  53.540000    53.775001  …
3      40635000.0   53.770000   54.119998  53.764999    54.090000  …
4      44107080.0   54.139999   54.169998  53.979999    54.150001  …
…             …          …          …          …            …  …
```

```
7233   20145140.0   32.895000   33.119998  32.880001    32.994998  …
7234   27192892.0   33.000000   33.145000  32.950000    33.130001  …
7235   25875275.0   33.125598   33.359001  33.103599    33.314998  …
7236   20012085.0   33.310001   33.409999  33.270000    33.369998  …
7237          NaN   33.369998   33.369998  33.369998    33.369998  …

      Selling General And Administration  Special Income Charges  \
0                                    NaN                     NaN
1                                    NaN                     NaN
2                                    NaN                     NaN
3                                    NaN                     NaN
4                                    NaN                     NaN
…                                  …                     …
7233                                 NaN                     NaN
7234                                 NaN                     NaN
7235                                 NaN                     NaN
7236                                 NaN                     NaN
7237                                 NaN                     NaN

      Tax Effect Of Unusual Items  Tax Provision  Tax Rate For Calcs  \
0                             NaN            NaN                 NaN
1                             NaN            NaN                 NaN
2                             NaN            NaN                 NaN
3                             NaN            NaN                 NaN
4                             NaN            NaN                 NaN
…                           …            …                 …
7233                          NaN            NaN                 NaN
7234                          NaN            NaN                 NaN
7235                          NaN            NaN                 NaN
7236                          NaN            NaN                 NaN
7237                          NaN            NaN                 NaN

      Total Expenses  Total Operating Income As Reported  Total Revenue  \
0                NaN                                 NaN            NaN
1                NaN                                 NaN            NaN
2                NaN                                 NaN            NaN
3                NaN                                 NaN            NaN
4                NaN                                 NaN            NaN
…              …                                 …            …
7233             NaN                                 NaN            NaN
7234             NaN                                 NaN            NaN
7235             NaN                                 NaN            NaN
7236             NaN                                 NaN            NaN
7237             NaN                                 NaN            NaN

      Total Unusual Items  Total Unusual Items Excluding Goodwill
0                     NaN                                     NaN
```

```
1                     NaN                                      NaN
2                     NaN                                      NaN
3                     NaN                                      NaN
4                     NaN                                      NaN
…                   …                                      …
7233                  NaN                                      NaN
7234                  NaN                                      NaN
7235                  NaN                                      NaN
7236                  NaN                                      NaN
7237                  NaN                                      NaN

[7238 rows x 97 columns]
```

2.5.3 Impact of the NVIDIA Balance Sheet on Stock Price

The balance sheet is crucial for stock price movements:

- **Liquidity Metrics**: High levels of cash and liquid assets indicate strong liquidity, which can positively impact stock prices.
- **Debt Levels**: Lower debt levels and manageable debt ratios are seen as favorable, reducing financial risk and potentially boosting stock prices.
- **Asset Management**: Efficient use of assets to generate revenue and profit enhances investor confidence, influencing stock prices.
- **Equity Value**: Strong shareholder equity reflects financial stability and growth potential, which can drive up stock prices.

NVIDIA's quarterly balance sheet affects investor perceptions and stock price through its financial health and stability.

[ ]:
```python
import pandas as pd
import yfinance as yf

# Define the ticker symbol for NVIDIA
ticker_symbol = 'NVDA'

# Fetch NVIDIA's financial data
nvidia_data = yf.Ticker(ticker_symbol)

# Get NVIDIA's quarterly balance sheet
quarterly_balance_sheet = nvidia_data.quarterly_balance_sheet

# Reset the index to turn the row indices into a column
pivoted_balance_sheet = quarterly_balance_sheet.reset_index()

# Melt the DataFrame to have metrics as rows
pivoted_balance_sheet = pivoted_balance_sheet.melt(id_vars='index',␣
 ↪var_name='Date', value_name='Amount')
```

```
pivoted_balance_sheet.columns = ['Metric', 'Date', 'Amount']

# Pivot the melted DataFrame so that metrics are columns
pivoted_balance_sheet = pivoted_balance_sheet.pivot_table(index='Date',␣
↪columns='Metric', values='Amount')

# Reset index to make 'Date' a column again
pivoted_balance_sheet.reset_index(inplace=True)

# Print the pivoted balance sheet
print("NVIDIA Quarterly Balance Sheet with Metrics as Columns:")
pivoted_balance_sheet
```

```
NVIDIA Quarterly Balance Sheet with Metrics as Columns:
```

[ ]:

```
Metric       Date  Accounts Payable  Accounts Receivable  \
0      2022-10-31               NaN                  NaN
1      2023-01-31               NaN                  NaN
2      2023-04-30      1141000000.0         4080000000.0
3      2023-07-31      1929000000.0         7066000000.0
4      2023-10-31      2380000000.0         8309000000.0
5      2024-01-31      2699000000.0         9999000000.0
6      2024-04-30      2715000000.0        12365000000.0

Metric Accumulated Depreciation Additional Paid In Capital  \
0                           NaN                        NaN
1                 -2694000000.0                        NaN
2                           NaN              12453000000.0
3                           NaN              12629000000.0
4                           NaN              12991000000.0
5                 -3509000000.0              13132000000.0
6                           NaN              12651000000.0

Metric Buildings And Improvements Capital Lease Obligations Capital Stock  \
0                             NaN                       NaN           NaN
1                    1598000000.0                       NaN           NaN
2                             NaN              1126000000.0     2000000.0
3                             NaN              1249000000.0     2000000.0
4                             NaN              1321000000.0     2000000.0
5                    1816000000.0              1347000000.0     2000000.0
6                             NaN              1527000000.0     2000000.0

Metric Cash And Cash Equivalents  \
0                            NaN
1                            NaN
2                   5079000000.0
3                   5783000000.0
```

```
4                 5519000000.0
5                 7280000000.0
6                 7587000000.0

Metric Cash Cash Equivalents And Short Term Investments  …     Total Debt  \
0                                                   NaN  …            NaN
1                                                   NaN  …            NaN
2                                         15320000000.0  …  12080000000.0
3                                         16023000000.0  …  10954000000.0
4                                         18281000000.0  …  11027000000.0
5                                         25984000000.0  …  11056000000.0
6                                         31438000000.0  …  11237000000.0

Metric Total Equity Gross Minority Interest  \
0                                       NaN
1                                       NaN
2                             24520000000.0
3                             27501000000.0
4                             33265000000.0
5                             42978000000.0
6                             49142000000.0

Metric Total Liabilities Net Minority Interest Total Non Current Assets  \
0                                          NaN                      NaN
1                                          NaN                      NaN
2                                19940000000.0            19577000000.0
3                                22054000000.0            20758000000.0
4                                20883000000.0            21490000000.0
5                                22750000000.0            21383000000.0
6                                27930000000.0            23343000000.0

Metric Total Non Current Liabilities Net Minority Interest Total Tax Payable  \
0                                                     NaN               NaN
1                                                     NaN               NaN
2                                           12680000000.0      1544000000.0
3                                           11720000000.0      2803000000.0
4                                           11782000000.0       420000000.0
5                                           12119000000.0       296000000.0
6                                           12707000000.0      3881000000.0

Metric Tradeand Other Payables Non Current Treasury Shares Number  \
0                                      NaN                    NaN
1                                      NaN                    NaN
2                             1455000000.0                    NaN
3                             1477000000.0                    NaN
4                             1319000000.0                    0.0
5                             1441000000.0                    NaN
```

```
6                              1613000000.0                    NaN

Metric Work In Process Working Capital
0                  NaN             NaN
1                  NaN             NaN
2          930000000.0   17623000000.0
3         1058000000.0   18463000000.0
4         1338000000.0   23557000000.0
5         1505000000.0   33714000000.0
6         1625000000.0   38506000000.0

[7 rows x 78 columns]
```

In the code below, the fetched balance sheet is joined with historical stock price data at the quarter and year level granularity. This allows for a comprehensive analysis of how balance sheet impacts the stock price over time.

[ ]:
```python
# Convert 'DATE' in pivoted_balance_sheet to datetime
pivoted_balance_sheet['Date'] = pd.to_datetime(pivoted_balance_sheet['Date'],␣
  ↪format='%Y_%m_%d')

# Extract year and quarter from 'Date' column in pivoted_balance_sheet
pivoted_balance_sheet['Year'] = pivoted_balance_sheet['Date'].dt.year
pivoted_balance_sheet['Quarter'] = pivoted_balance_sheet['Date'].dt.quarter

# Merge on 'Year' and 'Quarter'
merged_data = merged_data.merge(pivoted_balance_sheet, on=['Year', 'Quarter'],␣
  ↪how='left')

# Display the first few rows of the final dataset
print("\nFinal Merged Data:")
merged_data
```

```
Final Merged Data:
```

[ ]:
```
                Datetime   nvda_open   nvda_high    nvda_low  nvda_close  \
0    2020-10-12 13:30:00   13.989500   14.175000   13.912500   14.080792
1    2020-10-12 14:30:00   14.083000   14.190500   14.030251   14.129033
2    2020-10-12 15:30:00   14.130930   14.228999   14.100999   14.227374
3    2020-10-12 16:30:00   14.229500   14.260750   14.191499   14.254750
4    2020-10-12 17:30:00   14.265375   14.345750   14.229500   14.343750
…                    …          …          …          …          …
7233 2024-07-22 16:30:00  121.775001  122.949996  121.540000  122.800003
7234 2024-07-22 17:30:00  122.809997  124.069999  122.599998  123.514999
7235 2024-07-22 18:30:00  123.517501  123.750000  122.610000  122.839996
7236 2024-07-22 19:30:00  122.864997  123.750000  122.709999  123.739997
7237 2024-07-22 20:00:00  123.540000  123.540000  123.540000  123.540000
```

```
      nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
0     132333280.0   53.549999   53.619998  53.209999    53.349998  …   
1      50193760.0   53.345001   53.665000  53.279998    53.580001  …   
2      37900160.0   53.574199   53.799999  53.540000    53.775001  …   
3      40635000.0   53.770000   54.119998  53.764999    54.090000  …   
4      44107080.0   54.139999   54.169998  53.979999    54.150001  …   
…             …           …           …          …            …  …   
7233   20145140.0   32.895000   33.119998  32.880001    32.994998  …   
7234   27192892.0   33.000000   33.145000  32.950000    33.130001  …   
7235   25875275.0   33.125598   33.359001  33.103599    33.314998  …   
7236   20012085.0   33.310001   33.409999  33.270000    33.369998  …   
7237          NaN   33.369998   33.369998  33.369998    33.369998  …   

      Total Debt  Total Equity Gross Minority Interest  \
0            NaN                                   NaN   
1            NaN                                   NaN   
2            NaN                                   NaN   
3            NaN                                   NaN   
4            NaN                                   NaN   
…            …                                   …   
7233         NaN                                   NaN   
7234         NaN                                   NaN   
7235         NaN                                   NaN   
7236         NaN                                   NaN   
7237         NaN                                   NaN   

      Total Liabilities Net Minority Interest  Total Non Current Assets  \
0                                         NaN                       NaN   
1                                         NaN                       NaN   
2                                         NaN                       NaN   
3                                         NaN                       NaN   
4                                         NaN                       NaN   
…                                       …                       …   
7233                                      NaN                       NaN   
7234                                      NaN                       NaN   
7235                                      NaN                       NaN   
7236                                      NaN                       NaN   
7237                                      NaN                       NaN   

      Total Non Current Liabilities Net Minority Interest  Total Tax Payable  \
0                                                   NaN                NaN   
1                                                   NaN                NaN   
2                                                   NaN                NaN   
3                                                   NaN                NaN   
4                                                   NaN                NaN   
…                                                 …                …   
```

```
7233                                                         NaN                    NaN
7234                                                         NaN                    NaN
7235                                                         NaN                    NaN
7236                                                         NaN                    NaN
7237                                                         NaN                    NaN

      Tradeand Other Payables Non Current  Treasury Shares Number  \
0                                     NaN                     NaN
1                                     NaN                     NaN
2                                     NaN                     NaN
3                                     NaN                     NaN
4                                     NaN                     NaN
…                                     …                       …
7233                                  NaN                     NaN
7234                                  NaN                     NaN
7235                                  NaN                     NaN
7236                                  NaN                     NaN
7237                                  NaN                     NaN

      Work In Process  Working Capital
0                 NaN              NaN
1                 NaN              NaN
2                 NaN              NaN
3                 NaN              NaN
4                 NaN              NaN
…                 …                …
7233              NaN              NaN
7234              NaN              NaN
7235              NaN              NaN
7236              NaN              NaN
7237              NaN              NaN

[7238 rows x 175 columns]
```

2.5.4 Impact of the NVIDIA Cash Flow Statement on Stock Price

The cash flow statement is crucial for stock price movements:

- **Operating Cash Flow**: Strong operating cash flow indicates robust core business performance, which can positively impact stock prices.
- **Investment Activities**: Cash used or generated from investment activities provides insight into future growth prospects, influencing stock prices.
- **Financing Activities**: Effective management of cash from financing activities, such as debt repayment or share repurchases, can enhance investor confidence and impact stock prices.
- **Free Cash Flow**: High free cash flow signifies the company's ability to generate surplus cash, which can be used for expansion, dividends, or reducing debt, potentially boosting stock prices.

NVIDIA's quarterly cash flow statement affects investor perceptions and stock price through its cash management and overall financial health.

[ ]:
```
import pandas as pd
import yfinance as yf

# Define the ticker symbol for NVIDIA
ticker_symbol = 'NVDA'

# Fetch NVIDIA's financial data
nvidia_data = yf.Ticker(ticker_symbol)

# Get NVIDIA's quarterly cash flow statement
quarterly_cash_flow = nvidia_data.quarterly_cashflow

# Reset the index to turn the row indices into a column
pivoted_cash_flow = quarterly_cash_flow.reset_index()

# Melt the DataFrame to have metrics as rows
pivoted_cash_flow = pivoted_cash_flow.melt(id_vars='index', var_name='Date',␣
↪value_name='Amount')
pivoted_cash_flow.columns = ['Metric', 'Date', 'Amount']

# Pivot the melted DataFrame so that metrics are columns
pivoted_cash_flow = pivoted_cash_flow.pivot_table(index='Date',␣
↪columns='Metric', values='Amount')

# Reset index to make 'Date' a column again
pivoted_cash_flow.reset_index(inplace=True)

# Print the pivoted cash flow statement
print("NVIDIA Quarterly Cash Flow Statement with Metrics as Columns:")
pivoted_cash_flow
```

```
NVIDIA Quarterly Cash Flow Statement with Metrics as Columns:
```

[ ]:
```
Metric       Date Beginning Cash Position Capital Expenditure  \
0      2022-10-31                     NaN                 NaN
1      2023-01-31                     NaN                 NaN
2      2023-04-30            3389000000.0        -248000000.0
3      2023-07-31            5079000000.0        -289000000.0
4      2023-10-31            5882000000.0        -278000000.0
5      2024-01-31            5519000000.0        -254000000.0
6      2024-04-30            7280000000.0        -369000000.0

Metric Cash Dividends Paid Cash Flow From Continuing Financing Activities  \
0                      NaN                                            NaN
```

```
1                 NaN                                             NaN
2           -99000000.0                                     -380000000.0
3          -100000000.0                                    -5099000000.0
4           -97000000.0                                    -4525000000.0
5           -99000000.0                                    -3629000000.0
6           -98000000.0                                    -9345000000.0

Metric Cash Flow From Continuing Investing Activities  \
0                                                 NaN
1                                                 NaN
2                                        -841000000.0
3                                        -446000000.0
4                                       -3170000000.0
5                                       -6109000000.0
6                                       -5693000000.0

Metric Cash Flow From Continuing Operating Activities  \
0                                                 NaN
1                                                 NaN
2                                        2911000000.0
3                                        6348000000.0
4                                        7332000000.0
5                                       11499000000.0
6                                       15345000000.0

Metric Change In Account Payable Change In Accrued Expense  \
0                           NaN                       NaN
1                           NaN                       NaN
2                    11000000.0               689000000.0
3                   778000000.0              1986000000.0
4                   461000000.0             -1722000000.0
5                   281000000.0              1072000000.0
6                   -22000000.0              4202000000.0

Metric Change In Inventory  … Operating Gains Losses Other Non Cash Items  \
0                     NaN  …                    NaN                  NaN
1                     NaN  …                    NaN                  NaN
2             566000000.0  …             14000000.0          -34000000.0
3             295000000.0  …            -59000000.0          -68000000.0
4            -456000000.0  …             69000000.0          -68000000.0
5            -503000000.0  …           -262000000.0         -108000000.0
6            -577000000.0  …            -69000000.0         -145000000.0

Metric Proceeds From Stock Option Exercised Purchase Of Business  \
0                                      NaN                  NaN
1                                      NaN                  NaN
2                              246000000.0         -304000000.0
```

```
3                                     1000000.0          221000000.0
4                                   156000000.0                  0.0
5                                           0.0                  0.0
6                                   285000000.0         -174000000.0

Metric Purchase Of Investment Purchase Of PPE Repayment Of Debt  \
0                         NaN             NaN               0.0
1                         NaN             NaN               0.0
2               -2801000000.0    -248000000.0               NaN
3               -2977000000.0    -289000000.0               NaN
4               -5782000000.0    -278000000.0               0.0
5               -7636000000.0    -254000000.0               0.0
6               -9303000000.0    -369000000.0               NaN

Metric Repurchase Of Capital Stock Sale Of Investment Stock Based Compensation
0                              NaN                NaN                      NaN
1                              NaN                NaN                      NaN
2                              0.0       2512000000.0              735000000.0
3                    -3067000000.0       2599000000.0              841000000.0
4                    -3807000000.0       2890000000.0              979000000.0
5                    -2659000000.0       1781000000.0              994000000.0
6                    -7740000000.0       4153000000.0             1011000000.0

[7 rows x 52 columns]
```

In the code below, the fetched cash flow is joined with historical stock price data at the quarter and year level granularity. This allows for a comprehensive analysis of how cash flow impacts the stock price over time.

```
[ ]: # Convert 'DATE' in pivoted_cash_flow to datetime
pivoted_cash_flow['Date'] = pd.to_datetime(pivoted_cash_flow['Date'],
 ↪format='%Y_%m_%d')

# Extract year and quarter from 'Date' column in pivoted_cash_flow
pivoted_cash_flow['Year'] = pivoted_cash_flow['Date'].dt.year
pivoted_cash_flow['Quarter'] = pivoted_cash_flow['Date'].dt.quarter

# Merge on 'Year' and 'Quarter', specifying suffixes to avoid duplicates
merged_data = merged_data.merge(pivoted_cash_flow, on=['Year', 'Quarter'],
 ↪how='left', suffixes=('_existing', '_cashflow'))

# Display the first few rows of the final dataset
print("\nFinal Merged Data:")
merged_data
```

```
Final Merged Data:
```

[ ]:
```
                 Datetime   nvda_open   nvda_high    nvda_low  nvda_close  \
0     2020-10-12 13:30:00   13.989500   14.175000   13.912500   14.080792
1     2020-10-12 14:30:00   14.083000   14.190500   14.030251   14.129033
2     2020-10-12 15:30:00   14.130930   14.228999   14.100999   14.227374
3     2020-10-12 16:30:00   14.229500   14.260750   14.191499   14.254750
4     2020-10-12 17:30:00   14.265375   14.345750   14.229500   14.343750
…                     …           …           …           …           …
7233 2024-07-22 16:30:00  121.775001  122.949996  121.540000  122.800003
7234 2024-07-22 17:30:00  122.809997  124.069999  122.599998  123.514999
7235 2024-07-22 18:30:00  123.517501  123.750000  122.610000  122.839996
7236 2024-07-22 19:30:00  122.864997  123.750000  122.709999  123.739997
7237 2024-07-22 20:00:00  123.540000  123.540000  123.540000  123.540000

       nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
0      132333280.0   53.549999   53.619998  53.209999    53.349998  …
1       50193760.0   53.345001   53.665000  53.279998    53.580001  …
2       37900160.0   53.574199   53.799999  53.540000    53.775001  …
3       40635000.0   53.770000   54.119998  53.764999    54.090000  …
4       44107080.0   54.139999   54.169998  53.979999    54.150001  …
…              …           …           …          …            …  …
7233    20145140.0   32.895000   33.119998  32.880001    32.994998  …
7234    27192892.0   33.000000   33.145000  32.950000    33.130001  …
7235    25875275.0   33.125598   33.359001  33.103599    33.314998  …
7236    20012085.0   33.310001   33.409999  33.270000    33.369998  …
7237           NaN   33.369998   33.369998  33.369998    33.369998  …

       Operating Gains Losses  Other Non Cash Items  \
0                         NaN                   NaN
1                         NaN                   NaN
2                         NaN                   NaN
3                         NaN                   NaN
4                         NaN                   NaN
…                         …                   …
7233                      NaN                   NaN
7234                      NaN                   NaN
7235                      NaN                   NaN
7236                      NaN                   NaN
7237                      NaN                   NaN

       Proceeds From Stock Option Exercised  Purchase Of Business  \
0                                       NaN                   NaN
1                                       NaN                   NaN
2                                       NaN                   NaN
3                                       NaN                   NaN
4                                       NaN                   NaN
…                                       …                   …
7233                                    NaN                   NaN
```

```
7234                                     NaN                   NaN
7235                                     NaN                   NaN
7236                                     NaN                   NaN
7237                                     NaN                   NaN

      Purchase Of Investment  Purchase Of PPE  Repayment Of Debt  \
0                        NaN              NaN                NaN
1                        NaN              NaN                NaN
2                        NaN              NaN                NaN
3                        NaN              NaN                NaN
4                        NaN              NaN                NaN
…                      …              …                …
7233                     NaN              NaN                NaN
7234                     NaN              NaN                NaN
7235                     NaN              NaN                NaN
7236                     NaN              NaN                NaN
7237                     NaN              NaN                NaN

      Repurchase Of Capital Stock  Sale Of Investment  \
0                             NaN                 NaN
1                             NaN                 NaN
2                             NaN                 NaN
3                             NaN                 NaN
4                             NaN                 NaN
…                           …                 …
7233                          NaN                 NaN
7234                          NaN                 NaN
7235                          NaN                 NaN
7236                          NaN                 NaN
7237                          NaN                 NaN

      Stock Based Compensation
0                          NaN
1                          NaN
2                          NaN
3                          NaN
4                          NaN
…                        …
7233                       NaN
7234                       NaN
7235                       NaN
7236                       NaN
7237                       NaN

[7238 rows x 227 columns]
```

2.5.5 Step 2: Data Preparation

In this step, we will focus on data quality checks followed by data cleaning tasks. As part of the data quality checks, we will list all the variables along with their descriptions and data types. We will also examine sample values from each variable. Our dataset does not contain categorical variables but includes many numerical variables. We will choose 5 numerical variables. We will check the following information for each variable:

1. Number of observations in the variable
2. Range of the variable
3. Minimum and Maximum of the variable
4. Mean and standard deviation/variance of the variable
5. Mode, median, and quartiles
6. Histogram of the variable
7. Any interesting findings

Below is the Python code to perform these checks and generate the required statistics and visualizations.

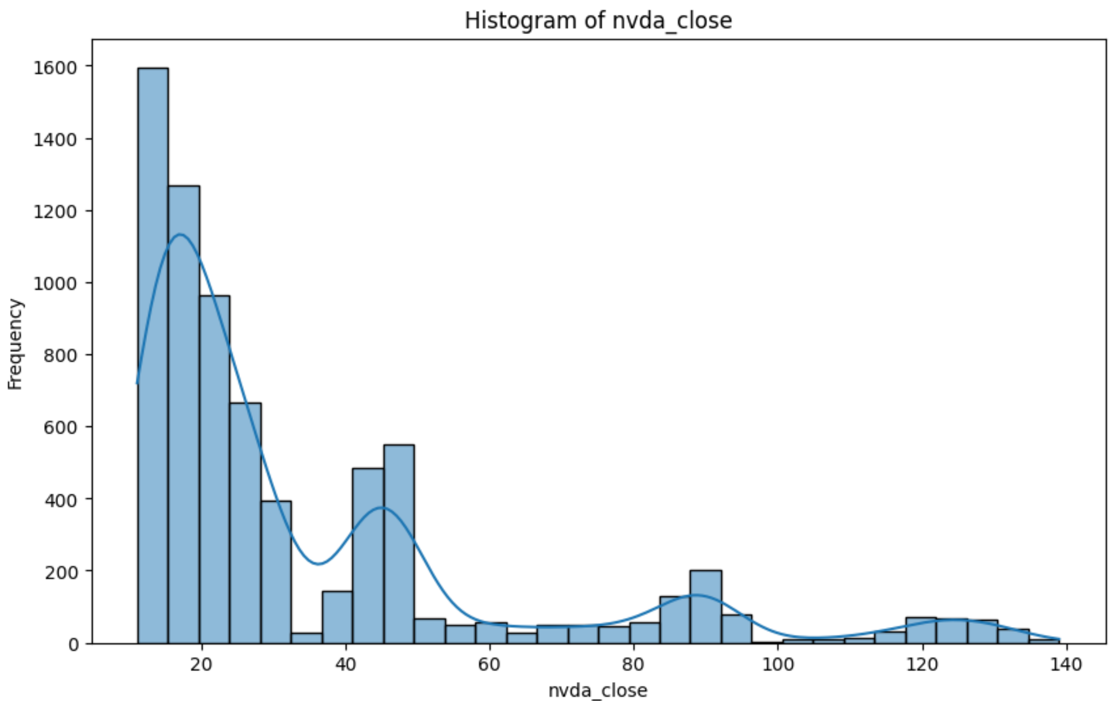
[ ]:
```
# Function to generate summary report of DataFrame variables
def summarize_dataframe(df):
    # Initialize an empty list to store summary data
    summary = []

    # Iterate over each column in the DataFrame
    for column in df.columns:
        # Get data type of the variable
        dtype = df[column].dtypes

        # Handle cases where there are no non-null values
        if df[column].dropna().size > 0:
            # Extract values and convert to list if it's a Series
            values = df[column].dropna().sample(n=5, random_state=1).squeeze()
            non_null_values = values.tolist() if isinstance(values, pd.Series)␣
↪else values
        else:
            non_null_values = "No non-null values"  # Indicate no non-null␣
↪values

        # Add the summary data to the list
        summary.append({
            'Variable': column,
            'Data Type': dtype,
            'Sample Values': non_null_values
        })

    # Create a DataFrame from the summary data
    summary_df = pd.DataFrame(summary)
```

```
    return summary_df
```

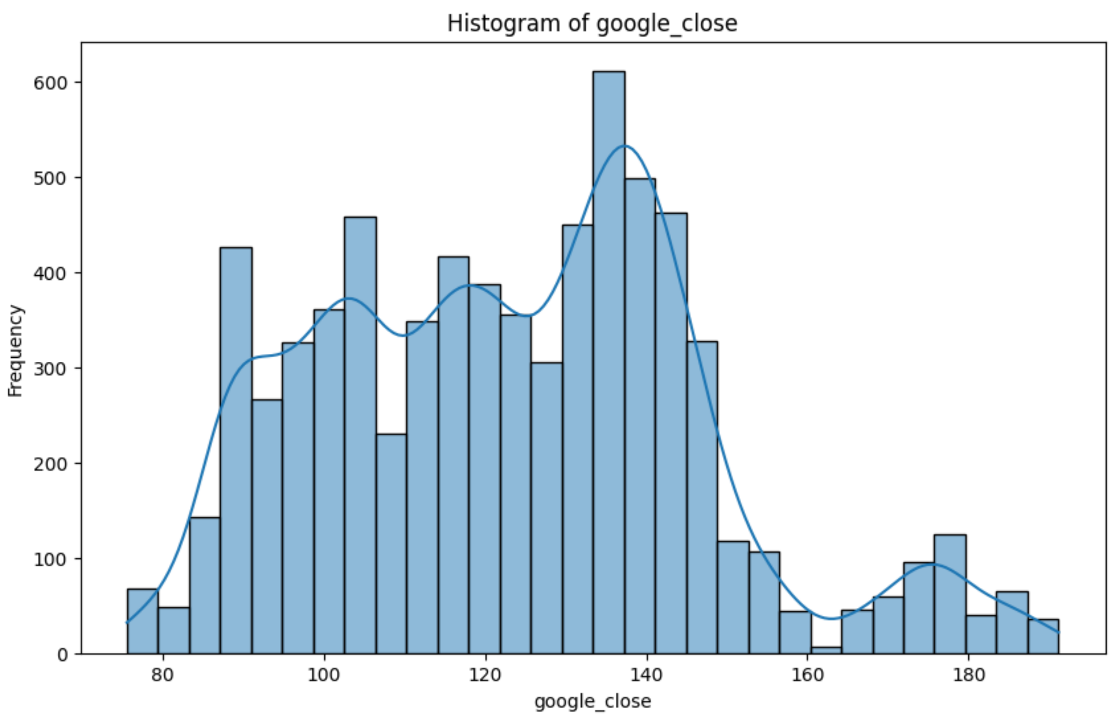
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def analyze_numeric_variable(df, column):
    if column not in df.columns:
        print(f"Column {column} does not exist in the DataFrame.")
        return

    print(f"\n### Analysis for {column} ###")

    # Number of observations
    num_obs = df[column].count()
    print(f"Number of observations: {num_obs}")

    # Range
    range_var = df[column].max() - df[column].min()
    print(f"Range: {range_var}")

    # Minimum and Maximum
    min_var = df[column].min()
    max_var = df[column].max()
    print(f"Min: {min_var}, Max: {max_var}")

    # Mean and Standard Deviation
    mean_var = df[column].mean()
    std_var = df[column].std()
    variance_var = df[column].var()
    print(f"Mean: {mean_var}, Standard Deviation: {std_var}, Variance: 
 ↪{variance_var}")

    # Mode, Median, and Quartiles
    mode_var = df[column].mode()[0]
    median_var = df[column].median()
    quartiles = df[column].quantile([0.25, 0.5, 0.75, 0.95])
    print(f"Mode: {mode_var}")
    print(f"Median: {median_var}")
    print(f"Quartiles:\n25%: {quartiles[0.25]}, 50%: {quartiles[0.5]}, 75%: 
 ↪{quartiles[0.75]}, 95%: {quartiles[0.95]}")

    # Histogram
    plt.figure(figsize=(10, 6))
    sns.histplot(df[column], bins=30, kde=True)
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
```

```python
    plt.show()

# List of numerical columns to analyze
numerical_columns = [
    'nvda_close',
    'google_close',
    'amd_close',
    'intel_close',
    'qcom_close'
]

# Analyze each specified numeric variable
for column in numerical_columns:
    analyze_numeric_variable(merged_data, column)
```

```
### Analysis for nvda_close ###
Number of observations: 7226
Range: 127.839008
Min: 11.14999, Max: 138.988998
Mean: 34.59393764079574, Standard Deviation: 27.823239977209557, Variance:
774.132682829392
Mode: 15.3720001
Median: 22.71825025
Quartiles:
25%: 16.003496875, 50%: 22.71825025, 75%: 43.978874149999996, 95%: 94.4526229
```

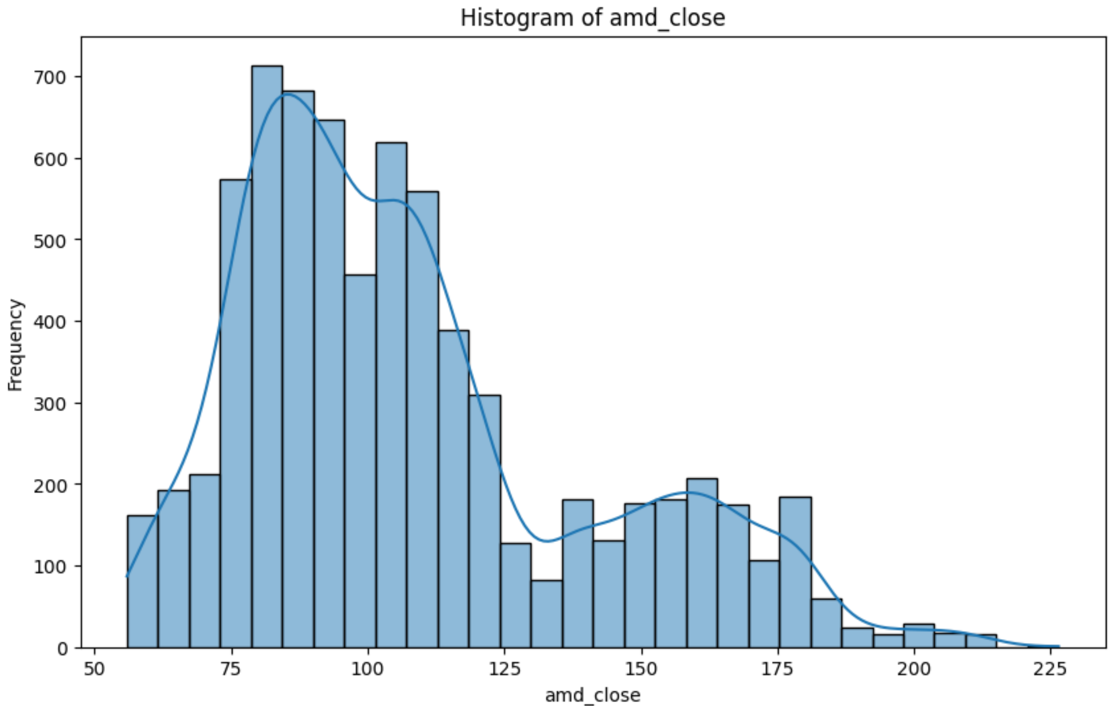


```
### Analysis for google_close ###
Number of observations: 7228
Range: 115.65799494999999
Min: 75.52199705000001, Max: 191.179992
Mean: 123.29750262587162, Standard Deviation: 23.810413041465207, Variance:
566.9357692051765
Mode: 105.569999
Median: 122.755001
Quartiles:
25%: 104.179750975, 50%: 122.755001, 75%: 138.8466232, 95%: 171.56899669999999
```

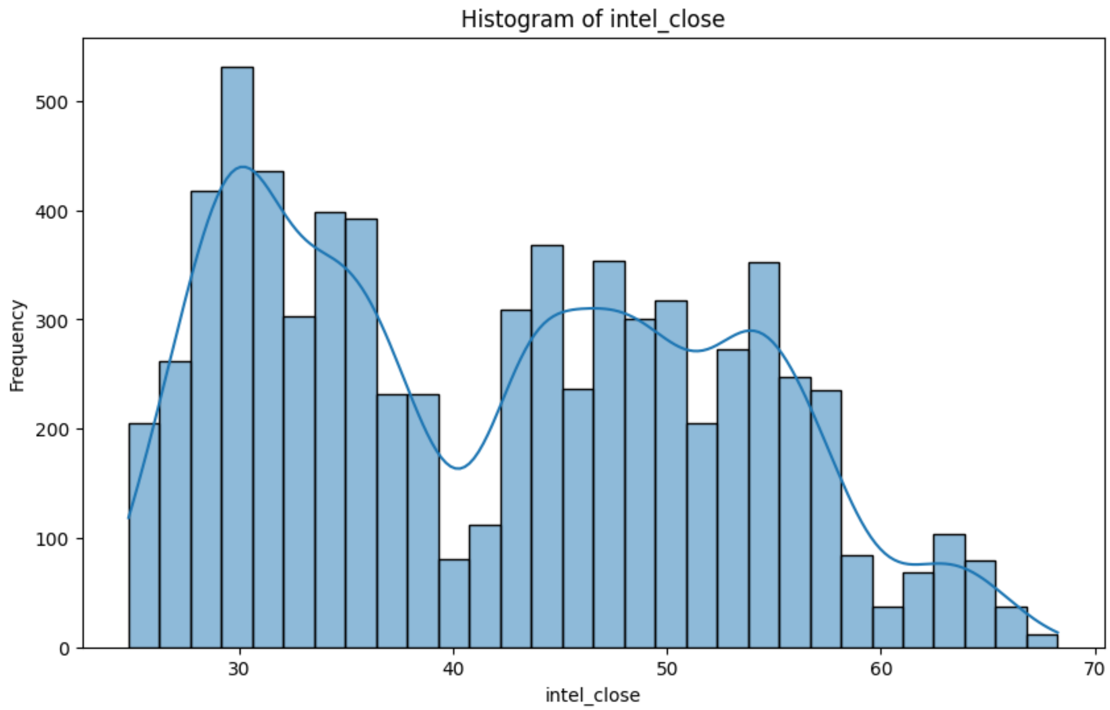



```
### Analysis for amd_close ###
Number of observations: 7230
Range: 170.510105
Min: 55.939998, Max: 226.450103
Mean: 108.13188081300137, Standard Deviation: 32.46368134244888, Variance:
1053.8906063040633
Mode: 81.050003
Median: 101.037498
Quartiles:
```

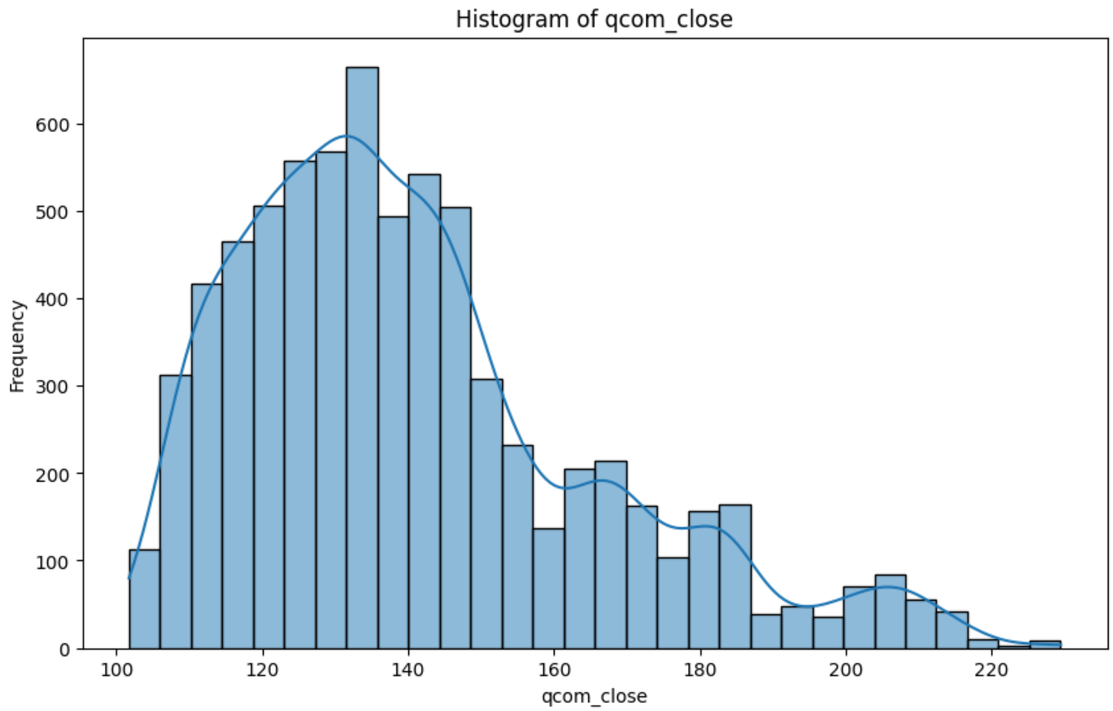
25%: 83.884998, 50%: 101.037498, 75%: 121.76257275, 95%: 174.229995



Analysis for intel_close
Number of observations: 7221
Range: 43.440101999999996
Min: 24.8199, Max: 68.260002
Mean: 41.86658582149287, Standard Deviation: 10.720923811160825, Variance: 114.93820736471514
Mode: 30.299999
Median: 42.25
Quartiles:
25%: 31.85, 50%: 42.25, 75%: 50.569999, 95%: 59.029998

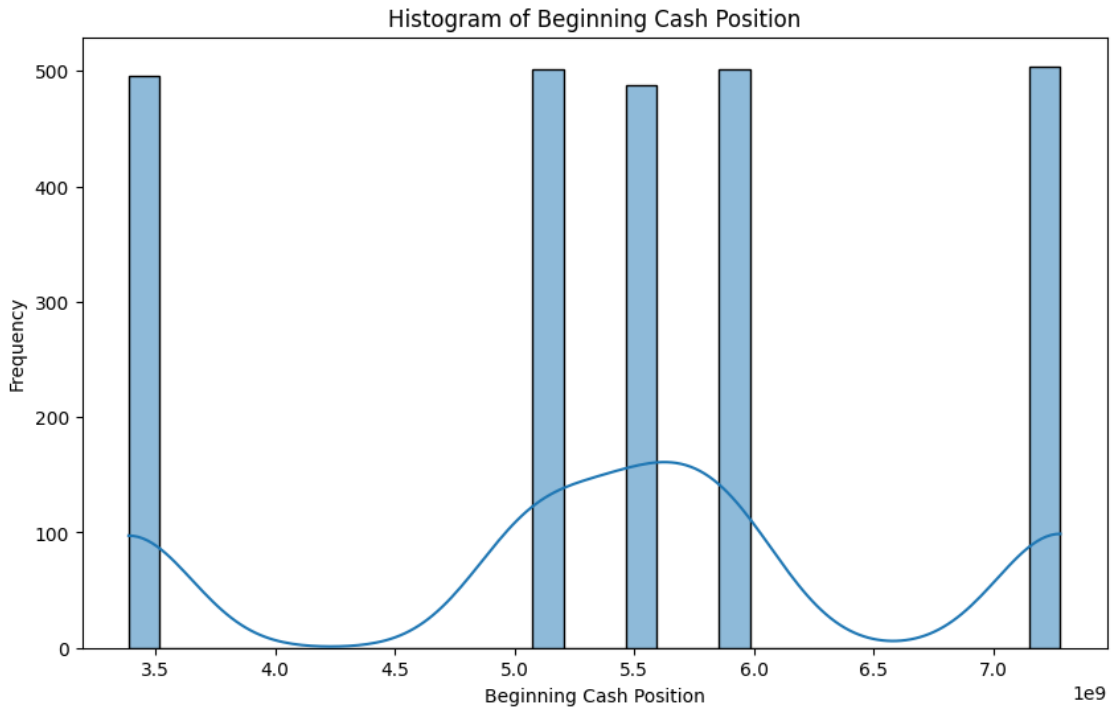



```
### Analysis for qcom_close ###
Number of observations: 7218
Range: 127.76959900000001
Min: 101.709999, Max: 229.479598
Mean: 140.87512301233028, Standard Deviation: 24.636692017235465, Variance:
606.9665935521136
Mode: 128.660003
Median: 135.8675
Quartiles:
25%: 122.97257575, 50%: 135.8675, 75%: 152.28499575, 95%: 189.72918589999998
```

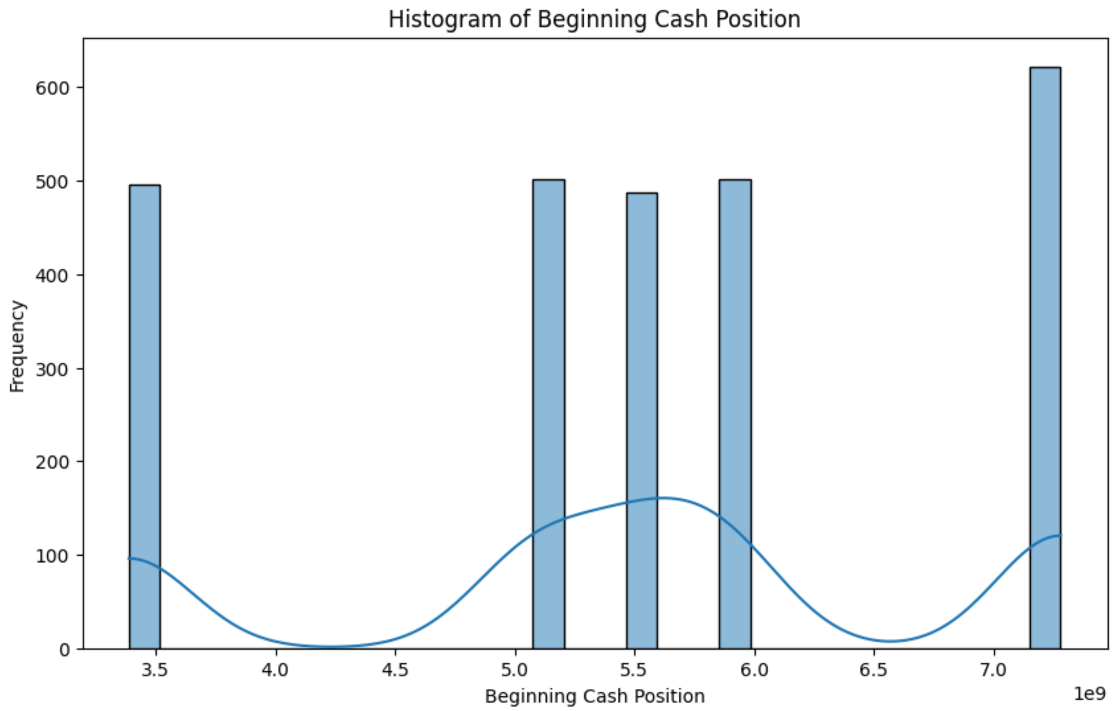


[ ]:
```
analyze_numeric_variable(merged_data, 'Beginning Cash Position')
# Forward fill NaN values
merged_data['Beginning Cash Position'] = merged_data['Beginning Cash Position'].
    ↪fillna(method='ffill')
analyze_numeric_variable(merged_data, 'Beginning Cash Position')
```

```
### Analysis for Beginning Cash Position ###
Number of observations: 2490
Range: 3891000000.0
Min: 3389000000.0, Max: 7280000000.0
Mean: 5435661445.783133, Standard Deviation: 1261186731.819916, Variance:
1.5905919725186004e+18
Mode: 7280000000.0
Median: 5519000000.0
Quartiles:
25%: 5079000000.0, 50%: 5519000000.0, 75%: 5882000000.0, 95%: 7280000000.0
```




```
### Analysis for Beginning Cash Position ###
Number of observations: 2608
Range: 3891000000.0
Min: 3389000000.0, Max: 7280000000.0
Mean: 5519109279.141105, Standard Deviation: 1290579822.8675084, Variance:
1.665596279192729e+18
Mode: 7280000000.0
Median: 5519000000.0
Quartiles:
25%: 5079000000.0, 50%: 5519000000.0, 75%: 5882000000.0, 95%: 7280000000.0
```



3 Detecting and Handling Outliers

3.1 Introduction

Outliers are data points that differ significantly from other observations in a dataset. They can arise due to variability in the data or might indicate errors or anomalies. Identifying and handling outliers is essential for ensuring the accuracy and reliability of data analysis and subsequent modeling.

3.2 Detecting Outliers

3.2.1 Statistical Methods

1. **Z-Score Method**: The Z-score measures how many standard deviations a data point is from the mean of the dataset. Data points with Z-scores that exceed a specified threshold are considered outliers. This method is useful for identifying outliers in normally distributed data.

2. **Interquartile Range (IQR) Method**: The IQR method identifies outliers based on the spread of the middle 50% of the data. It calculates the range between the first quartile (Q1) and the third quartile (Q3), and identifies values that fall below or above a defined multiple of the IQR as outliers. This method is robust against non-normal distributions.

3.2.2 Visualization Methods

1. **Box Plot**: A box plot provides a visual representation of the data distribution and highlights potential outliers. Outliers are typically shown as points outside the “whiskers” of the box

plot, which represent the range of the data within a certain percentile range.

2. **Scatter Plot**: Scatter plots can reveal outliers by showing the relationship between two variables. Outliers may appear as points that are distant from the general trend or cluster of data points.

3.3 Handling Outliers

3.3.1 Removing Outliers

1. **Statistical Thresholds**: Outliers identified using statistical methods can be removed from the dataset. This helps in avoiding distortion of statistical analyses and model performance.

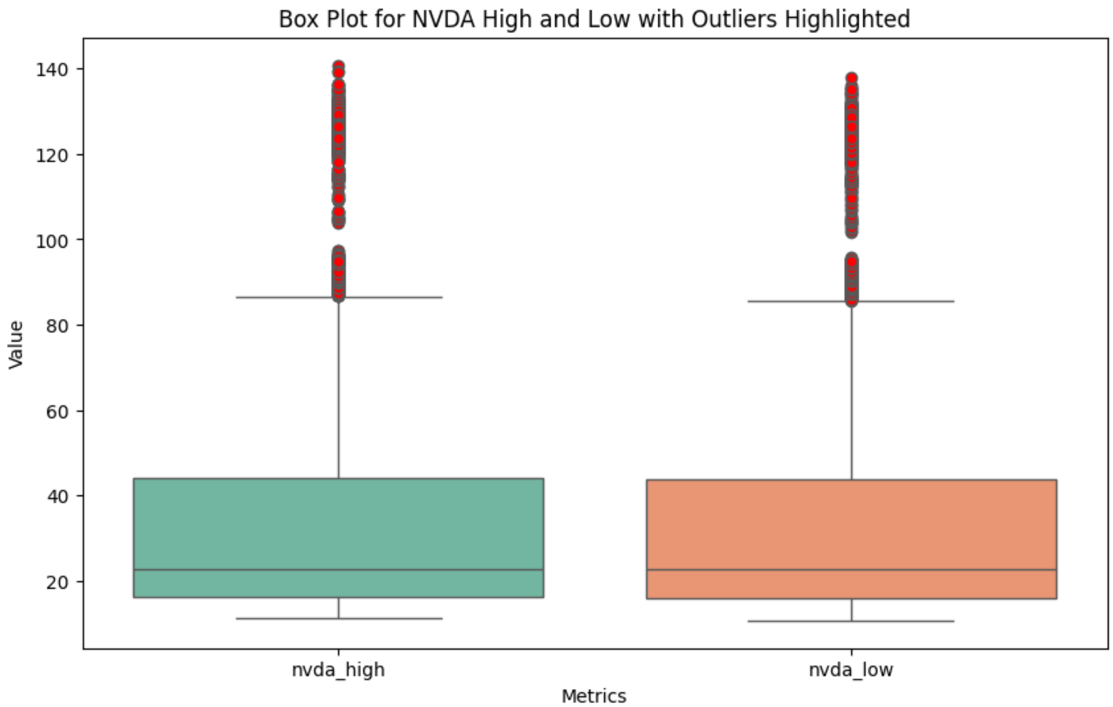
3.3.2 Transforming Data

1. **Log Transformation**: Log transformation compresses the range of data values and reduces the impact of extreme values. This method can stabilize variance and make the data more normally distributed.

2. **Winsorization**: Winsorization involves capping extreme values at a specified percentile. This reduces the influence of outliers by limiting their impact on statistical measures without completely removing them.

In the section below, we will use IQR method to remove outliers.

[ ]:
```
# Create the box plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=merged_data[['nvda_high', 'nvda_low']], palette='Set2',␣
↪flierprops=dict(markerfacecolor='r', marker='o'))

# Add titles and labels
plt.title('Box Plot for NVDA High and Low with Outliers Highlighted')
plt.xlabel('Metrics')
plt.ylabel('Value')

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



[ ]:
```
import pandas as pd

def drop_outliers(df, column):
    # Calculate Q1, Q3, and IQR
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]

    # Print information about the data before dropping outliers
    print(f"Number of rows before dropping outliers for {column}: {len(df)}")
    print(f"Sample outlier values for {column}: {outliers[column].head()}")

    # Drop outliers
    df.drop(outliers.index, inplace=True)

    # Print information about the data after dropping outliers
    print(f"Number of rows after dropping outliers for {column}: {len(df)}")

# List of columns to check for outliers
```

```
columns_to_check = ['nvda_low', 'nvda_high']

# Iterate over specified columns and remove outliers
for column in columns_to_check:
    drop_outliers(merged_data, column)
```

```
Number of rows before dropping outliers for nvda_low: 7238
Sample outlier values for nvda_low: 6467    85.801001
6468    86.453497
6469    86.551001
6479    85.964001
6480    87.030011
Name: nvda_low, dtype: float64
Number of rows after dropping outliers for nvda_low: 6563
Number of rows before dropping outliers for nvda_high: 6563
Sample outlier values for nvda_high: 6280    63.398999
6281    63.365991
6282    63.492999
6285    63.021997
6301    63.113989
Name: nvda_high, dtype: float64
Number of rows after dropping outliers for nvda_high: 6297
```

3.3.3 Handling Missing Values

In this step, we address missing values in the dataset using the **K-Nearest Neighbors (KNN) imputation method**. Specifically, we employ a KNN imputer with `n_neighbors` set to 5. This technique involves the following:

- **KNN Imputation**: For each missing value, the algorithm identifies the 5 nearest neighbors based on the distance metric (such as Euclidean distance) in the feature space.
- **Value Estimation**: The missing value is then imputed by taking the mean (or median) of the corresponding feature values from these nearest neighbors.

Using the KNN imputer helps to preserve the data's structure and relationships, providing more accurate and reliable imputations compared to simple methods like mean or median imputation. This step is crucial for maintaining the integrity of the dataset, especially in features with missing values that could significantly impact the modeling process.

[ ]:
```
target = merged_data['nvda_close']
features = merged_data.drop(columns=['nvda_close'])

# Convert all columns to numeric, non-convertible values will be set as NaN
numeric_features = features.apply(pd.to_numeric, errors='coerce')

# Check how many columns were successfully converted
print("Number of numeric columns after conversion:", numeric_features.shape[1])
```

```
Number of numeric columns after conversion: 226
```

[ ]:
```python
# Calculate the correlation matrix
correlation_matrix = numeric_features.corrwith(target).abs()

# Display the correlation matrix
print(correlation_matrix)

# Set a correlation threshold
correlation_threshold = 0.1

# Select features with correlation above the threshold
high_correlation_features = correlation_matrix[correlation_matrix >␣
↪correlation_threshold].index

# Filter the dataset to keep only high correlation features
filtered_data = features[high_correlation_features]

# Display the filtered data
# print(filtered_data)

filtered_data.info()

print(f"Original number of columns: {merged_data.shape[1]}")
print(f"Number of columns after variance thresholding: {filtered_data.
↪shape[1]}")
```

```
Datetime                        0.740416
nvda_open                       0.999823
nvda_high                       0.999898
nvda_low                        0.999901
nvda_volume                     0.059691
                                  …
Purchase Of PPE                 0.428858
Repayment Of Debt                    NaN
Repurchase Of Capital Stock     0.704566
Sale Of Investment              0.216686
Stock Based Compensation        0.758805
Length: 226, dtype: float64
<class 'pandas.core.frame.DataFrame'>
Index: 6297 entries, 0 to 6300
Columns: 191 entries, Datetime to Stock Based Compensation
dtypes: datetime64[ns](6), float64(36), int32(1), object(148)
memory usage: 9.2+ MB
Original number of columns: 227
Number of columns after variance thresholding: 191
```

[ ]:
```python
from sklearn.feature_selection import VarianceThreshold
```

```python
# Apply variance threshold
selector = VarianceThreshold(threshold=0.11)

# Exclude datetime columns from the DataFrame before applying VarianceThreshold
numeric_df = filtered_data.select_dtypes(exclude=['datetime64'])
high_variance_data = selector.fit_transform(numeric_df)

# Get the names of the features that were retained
# Use numeric_df.columns to align with the DataFrame used for VarianceThreshold
high_variance_features = numeric_df.columns[selector.get_support()]

# Create a new DataFrame with the high variance features
high_variance_data = pd.DataFrame(high_variance_data,␣
↪columns=high_variance_features)

# Display the high variance data
#print(high_variance_data.head())
```

[ ]:
```python
from sklearn.impute import KNNImputer

# Initialize KNN Imputer
knn_imputer = KNNImputer(n_neighbors=5)

# Impute missing values
imputed_data = knn_imputer.fit_transform(high_variance_data)

# Convert the imputed data back to a DataFrame
imputed_data = pd.DataFrame(imputed_data, columns=high_variance_data.columns)

# Display the imputed data
imputed_data
```

[ ]:

	nvda_open	nvda_high	nvda_low	intel_open	intel_high	intel_low	\
0	13.989500	14.175000	13.912500	53.549999	53.619998	53.209999	
1	14.083000	14.190500	14.030251	53.345001	53.665000	53.279998	
2	14.130930	14.228999	14.100999	53.574199	53.799999	53.540000	
3	14.229500	14.260750	14.191499	53.770000	54.119998	53.764999	
4	14.265375	14.345750	14.229500	54.139999	54.169998	53.979999	
…	…	…	…	…	…	…	
6292	62.100000	62.320001	61.650000	43.209999	43.409999	42.714000	
6293	62.309399	62.482001	61.757001	42.844001	42.884998	42.485000	
6294	61.926001	62.496997	61.819000	42.659999	43.200000	42.560001	
6295	62.409998	62.729999	62.321997	43.119998	43.369998	43.090000	
6296	62.675000	62.898999	62.600012	43.345001	43.569999	43.200000	

	intel_close	amd_open	amd_high	amd_low	…	\
0	53.349998	83.650001	84.940002	83.120002	…	

```
1       53.580001   84.778800   85.129997   84.569999  …
2       53.775001   84.955001   84.970100   84.150001  …
3       54.090000   84.589996   84.699996   84.230003  …
4       54.150001   84.510002   84.720001   84.379997  …
…           …           …           …           …  …
6292    42.845001  169.270004  169.800003  165.860000  …
6293    42.659999  168.089996  168.839996  166.240005  …
6294    43.119998  166.899993  169.345001  166.634399  …
6295    43.345001  168.529998  169.009994  166.779998  …
6296    43.263999  168.660003  170.740005  168.500000  …

      Operating Cash Flow  Operating Gains Losses  Other Non Cash Items  \
0            5.660600e+09             -44400000.0           -61200000.0
1            5.660600e+09             -44400000.0           -61200000.0
2            5.660600e+09             -44400000.0           -61200000.0
3            5.660600e+09             -44400000.0           -61200000.0
4            5.660600e+09             -44400000.0           -61200000.0
…                     …                     …                     …
6292         1.149900e+10            -262000000.0          -108000000.0
6293         1.149900e+10            -262000000.0          -108000000.0
6294         1.149900e+10            -262000000.0          -108000000.0
6295         1.149900e+10            -262000000.0          -108000000.0
6296         1.149900e+10            -262000000.0          -108000000.0

      Proceeds From Stock Option Exercised  Purchase Of Business  \
0                               50000000.0           116000000.0
1                               50000000.0           116000000.0
2                               50000000.0           116000000.0
3                               50000000.0           116000000.0
4                               50000000.0           116000000.0
…                                    …                     …
6292                                   0.0                   0.0
6293                                   0.0                   0.0
6294                                   0.0                   0.0
6295                                   0.0                   0.0
6296                                   0.0                   0.0

      Purchase Of Investment  Purchase Of PPE  Repurchase Of Capital Stock  \
0              -2.941800e+09     -280800000.0                -2.453600e+09
1              -2.941800e+09     -280800000.0                -2.453600e+09
2              -2.941800e+09     -280800000.0                -2.453600e+09
3              -2.941800e+09     -280800000.0                -2.453600e+09
4              -2.941800e+09     -280800000.0                -2.453600e+09
…                        …              …                          …
6292           -7.636000e+09     -254000000.0                -2.659000e+09
6293           -7.636000e+09     -254000000.0                -2.659000e+09
6294           -7.636000e+09     -254000000.0                -2.659000e+09
```

```
6295          -7.636000e+09    -254000000.0                   -2.659000e+09
6296          -7.636000e+09    -254000000.0                   -2.659000e+09

      Sale Of Investment  Stock Based Compensation
0           2.581600e+09               819800000.0
1           2.581600e+09               819800000.0
2           2.581600e+09               819800000.0
3           2.581600e+09               819800000.0
4           2.581600e+09               819800000.0
…                  …                        …
6292        1.781000e+09               994000000.0
6293        1.781000e+09               994000000.0
6294        1.781000e+09               994000000.0
6295        1.781000e+09               994000000.0
6296        1.781000e+09               994000000.0

[6297 rows x 179 columns]
```

[ ]:
```python
# Check for missing values in each column
missing_values = imputed_data.isnull().sum()

# Display the sum of missing values for each column
print("Missing values in each column:")
print(missing_values)

# Display columns with missing values
columns_with_missing_values = missing_values[missing_values > 0]
if columns_with_missing_values.empty:
    print("There are no columns with missing values.")
else:
    print("Columns with missing values:")
    print(columns_with_missing_values)
```

```
Missing values in each column:
nvda_open                      0
nvda_high                      0
nvda_low                       0
intel_open                     0
intel_high                     0
                              ..
Purchase Of Investment         0
Purchase Of PPE                0
Repurchase Of Capital Stock    0
Sale Of Investment             0
Stock Based Compensation       0
Length: 179, dtype: int64
There are no columns with missing values.
```

[ ]:
```
merged_data
```

[ ]:
```
                Datetime  nvda_open  nvda_high   nvda_low  nvda_close  \
1    2020-10-12 14:30:00  14.083000  14.190500  14.030251   14.129033
2    2020-10-12 15:30:00  14.130930  14.228999  14.100999   14.227374
3    2020-10-12 16:30:00  14.229500  14.260750  14.191499   14.254750
4    2020-10-12 17:30:00  14.265375  14.345750  14.229500   14.343750
8    2020-10-13 14:30:00  14.195250  14.317500  14.127721   14.305251
…                    …        …        …        …         …
4921 2023-05-24 16:30:00  29.984271  30.069000  29.888080   30.008990
4922 2023-05-24 17:30:00  30.004999  30.225000  29.939999   30.107001
4923 2023-05-24 18:30:00  30.107001  30.542999  29.981000   30.537979
4924 2023-05-24 19:30:00  30.537970  30.607001  30.322009   30.541000
4925 2023-05-24 20:00:00  30.538000  30.538000  30.538000   30.538000

      nvda_volume  intel_open  intel_high  intel_low  intel_close  …  \
1      50193760.0   53.345001   53.665000  53.279998    53.580001  …
2      37900160.0   53.574199   53.799999  53.540000    53.775001  …
3      40635000.0   53.770000   54.119998  53.764999    54.090000  …
4      44107080.0   54.139999   54.169998  53.979999    54.150001  …
8      43114280.0   53.860000   54.185001  53.669998    54.084999  …
…             …        …        …        …          …  …
4921   35435070.0   28.975000   29.024999  28.850000    28.934999  …
4922   43175620.0   28.930000   29.040000  28.879999    28.895000  …
4923   78406120.0   28.895000   29.010000  28.819999    29.004999  …
4924   76984590.0   29.010000   29.079999  28.899999    28.979999  …
4925          NaN   29.000000   29.000000  29.000000    29.000000  …

      Operating Gains Losses  Other Non Cash Items  \
1                        NaN                   NaN
2                        NaN                   NaN
3                        NaN                   NaN
4                        NaN                   NaN
8                        NaN                   NaN
…                     …                   …
4921              14000000.0           -34000000.0
4922              14000000.0           -34000000.0
4923              14000000.0           -34000000.0
4924              14000000.0           -34000000.0
4925              14000000.0           -34000000.0

      Proceeds From Stock Option Exercised  Purchase Of Business  \
1                                      NaN                   NaN
2                                      NaN                   NaN
3                                      NaN                   NaN
4                                      NaN                   NaN
8                                      NaN                   NaN
```

```
…                              …                   …
4921                 246000000.0        -304000000.0
4922                 246000000.0        -304000000.0
4923                 246000000.0        -304000000.0
4924                 246000000.0        -304000000.0
4925                 246000000.0        -304000000.0

      Purchase Of Investment  Purchase Of PPE  Repayment Of Debt  \
1                        NaN              NaN                NaN
2                        NaN              NaN                NaN
3                        NaN              NaN                NaN
4                        NaN              NaN                NaN
8                        NaN              NaN                NaN
…                      …              …                 …
4921           -2801000000.0     -248000000.0                NaN
4922           -2801000000.0     -248000000.0                NaN
4923           -2801000000.0     -248000000.0                NaN
4924           -2801000000.0     -248000000.0                NaN
4925           -2801000000.0     -248000000.0                NaN

      Repurchase Of Capital Stock  Sale Of Investment  \
1                             NaN                 NaN
2                             NaN                 NaN
3                             NaN                 NaN
4                             NaN                 NaN
8                             NaN                 NaN
…                           …                 …
4921                          0.0        2512000000.0
4922                          0.0        2512000000.0
4923                          0.0        2512000000.0
4924                          0.0        2512000000.0
4925                          0.0        2512000000.0

      Stock Based Compensation
1                          NaN
2                          NaN
3                          NaN
4                          NaN
8                          NaN
…                        …
4921               735000000.0
4922               735000000.0
4923               735000000.0
4924               735000000.0
4925               735000000.0

[3102 rows x 227 columns]
```

4 Heat Map

This heatmap visualizes the historical stock prices of NVIDIA company over a specified period. The data is arranged in a matrix format where each cell represents the stock price for a particular time period. The color intensity of each cell corresponds to the magnitude of the stock price, making it easy to identify fluctuations and trends at a glance.

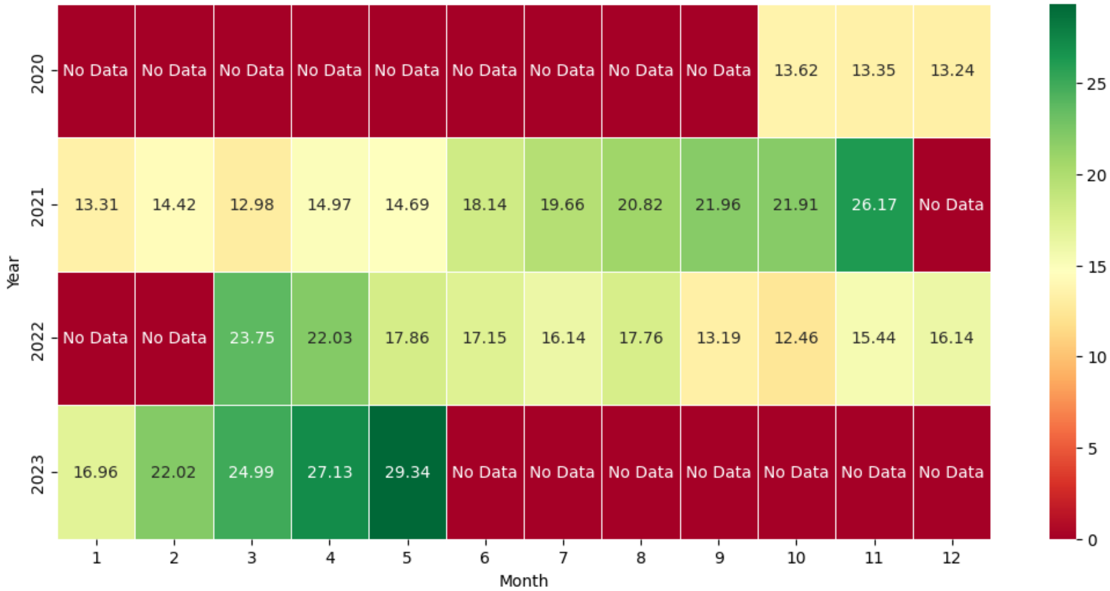
The heatmaps show :

1) The stock prices satying low as an effect of covid during 2020.
2) Started reviving during late 2021 going into 2022.
3) Reached highest as of early 2023 in the last 5 years.

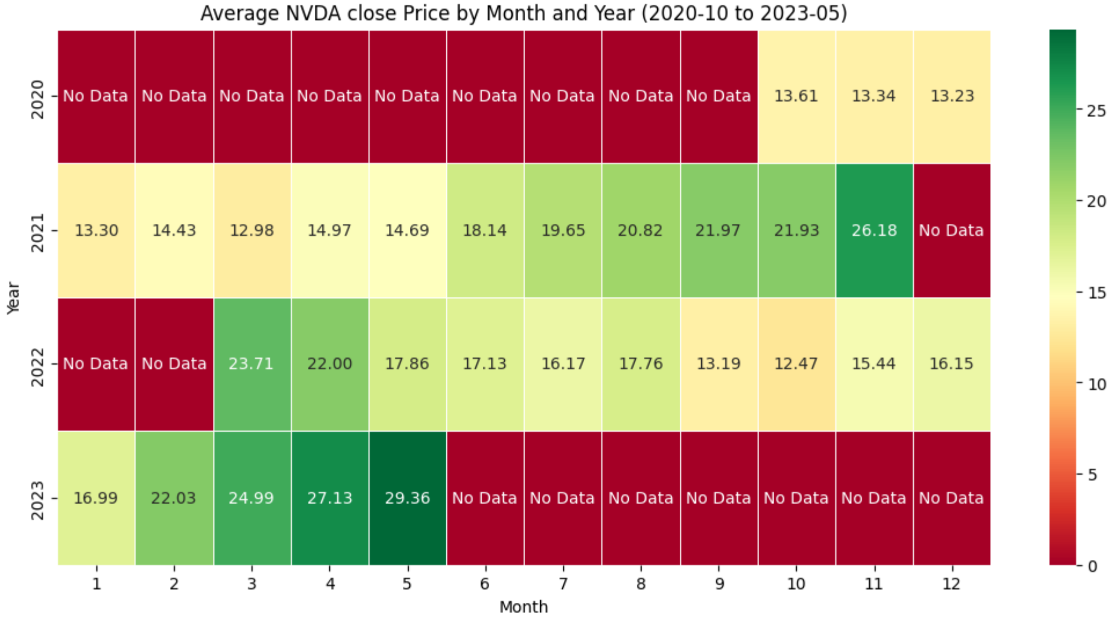
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
# Extract year and month from the 'Date' column
merged_data['Year'] = merged_data['Datetime'].dt.year
merged_data['Month'] = merged_data['Datetime'].dt.month

# Filter data for the specified period
start_date = pd.to_datetime('2020-10-01')
end_date = pd.to_datetime('2023-05-31')
filtered_data = merged_data[(merged_data['Datetime'] >= start_date) &
 ↪(merged_data['Datetime'] <= end_date)]

# Calculate the average 'nvda_open' for each month and year
heatmap_data = filtered_data.groupby(['Year', 'Month'])['nvda_open'].mean().
 ↪unstack(fill_value=0)
heatmap_data1 = filtered_data.groupby(['Year', 'Month'])['nvda_close'].mean().
 ↪unstack(fill_value=0)
heatmap_data2 = filtered_data.groupby(['Year', 'Month'])['nvda_high'].mean().
 ↪unstack(fill_value=0)
heatmap_data3 = filtered_data.groupby(['Year', 'Month'])['nvda_low'].mean().
 ↪unstack(fill_value=0)



# Create the heatmap
plt.figure(figsize=(13, 6))
ax=sns.heatmap(heatmap_data, cmap='RdYlGn', annot=True, fmt=".2f", linewidths=.
 ↪5)
# Customize annotations
for text in ax.texts:
    if text.get_text() == '0.00':
        text.set_text('No Data')
plt.title('Average NVDA Open Price by Month and Year (2020-10 to 2023-05)')
plt.xlabel('Month')
```

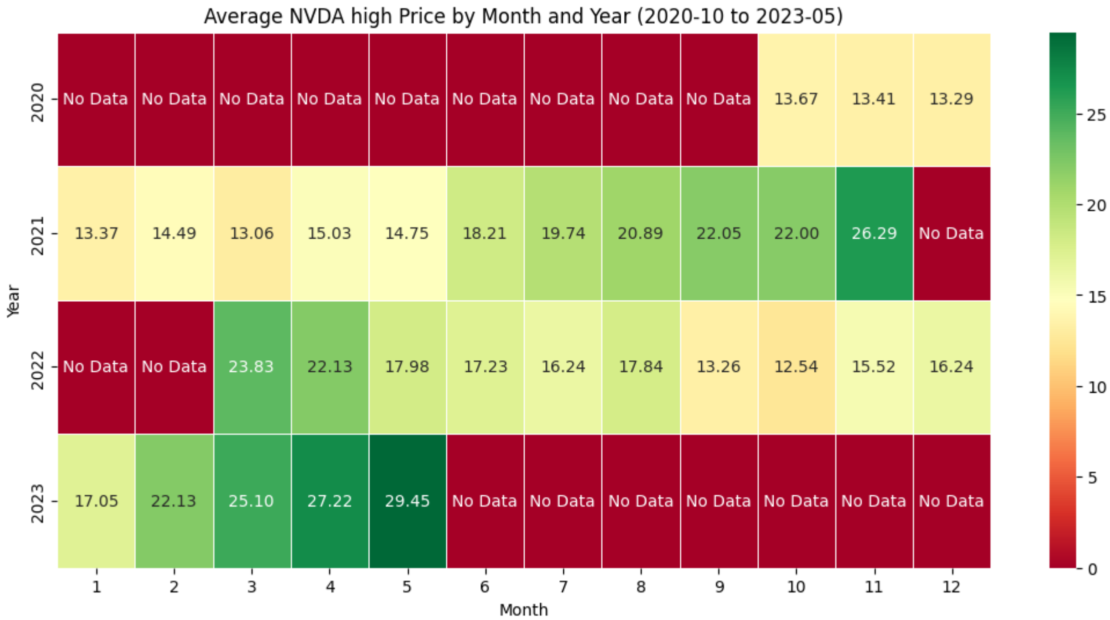
```
plt.ylabel('Year')
plt.show()

plt.figure(figsize=(13, 6))
bx=sns.heatmap(heatmap_data1, cmap='RdYlGn', annot=True, fmt=".2f", linewidths=.
  ↪5)
# Customize annotations
for text in bx.texts:
    if text.get_text() == '0.00':
        text.set_text('No Data')
plt.title('Average NVDA close Price by Month and Year (2020-10 to 2023-05)')
plt.xlabel('Month')
plt.ylabel('Year')
plt.show()

plt.figure(figsize=(13, 6))
cx=sns.heatmap(heatmap_data2, cmap='RdYlGn', annot=True, fmt=".2f", linewidths=.
  ↪5)
# Customize annotations
for text in cx.texts:
    if text.get_text() == '0.00':
        text.set_text('No Data')
plt.title('Average NVDA high Price by Month and Year (2020-10 to 2023-05)')
plt.xlabel('Month')
plt.ylabel('Year')
plt.show()

plt.figure(figsize=(13, 6))
dx=sns.heatmap(heatmap_data3, cmap='RdYlGn', annot=True, fmt=".2f", linewidths=.
  ↪5)
# Customize annotations
for text in dx.texts:
    if text.get_text() == '0.00':
        text.set_text('No Data')
plt.title('Average NVDA low Price by Month and Year (2020-10 to 2023-05)')
plt.xlabel('Month')
plt.ylabel('Year')
plt.show()
```

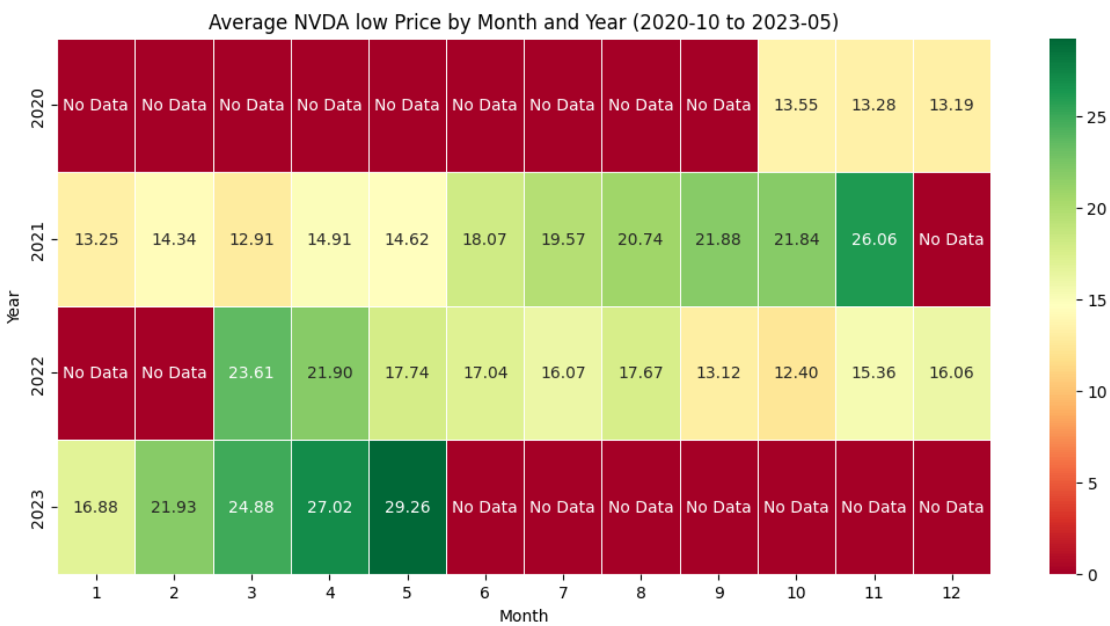
Average NVDA Open Price by Month and Year (2020-10 to 2023-05)



Average NVDA close Price by Month and Year (2020-10 to 2023-05)







5 Data Transformation and Feature Engineering Steps

In this project, we will undertake the following steps to transform and engineer features, preparing the stock data for modeling:

1. **Creating New Features**:

- Add a 'Price Change Percentage' feature to capture daily stock fluctuations.

2. **Transforming Features**:
   - Apply standardization using Z-score
3. **Binning**:
   - Categorize 'Price Change Percentage' into 'Low', 'Medium', and 'High' bins to simplify data interpretation and analysis.
4. **Feature Selection**:
   - Reduce the dataset's dimensionality by eliminating features with low variance, focusing on the most informative features for the model.

These steps are designed to refine the dataset, enhancing its suitability for effective modeling and analysis.

[ ]:
```
# Create new features
imputed_data['price_change_percentage'] = imputed_data['nvda_open'].
 ↪pct_change() * 100
imputed_data
```

[ ]:
```
      nvda_open  nvda_high   nvda_low  intel_open  intel_high  intel_low  \
0     13.989500  14.175000  13.912500   53.549999   53.619998  53.209999
1     14.083000  14.190500  14.030251   53.345001   53.665000  53.279998
2     14.130930  14.228999  14.100999   53.574199   53.799999  53.540000
3     14.229500  14.260750  14.191499   53.770000   54.119998  53.764999
4     14.265375  14.345750  14.229500   54.139999   54.169998  53.979999
…           …          …          …           …           …          …
6292  62.100000  62.320001  61.650000   43.209999   43.409999  42.714000
6293  62.309399  62.482001  61.757001   42.844001   42.884998  42.485000
6294  61.926001  62.496997  61.819000   42.659999   43.200000  42.560001
6295  62.409998  62.729999  62.321997   43.119998   43.369998  43.090000
6296  62.675000  62.898999  62.600012   43.345001   43.569999  43.200000

      intel_close    amd_open    amd_high     amd_low  …  \
0       53.349998   83.650001   84.940002   83.120002  …
1       53.580001   84.778800   85.129997   84.569999  …
2       53.775001   84.955001   84.970100   84.150001  …
3       54.090000   84.589996   84.699996   84.230003  …
4       54.150001   84.510002   84.720001   84.379997  …
…             …           …           …          …  …
6292    42.845001  169.270004  169.800003  165.860000  …
6293    42.659999  168.089996  168.839996  166.240005  …
6294    43.119998  166.899993  169.345001  166.634399  …
6295    43.345001  168.529998  169.009994  166.779998  …
6296    43.263999  168.660003  170.740005  168.500000  …

      Operating Gains Losses  Other Non Cash Items  \
0                -44400000.0           -61200000.0
1                -44400000.0           -61200000.0
2                -44400000.0           -61200000.0
```

```
3               -44400000.0          -61200000.0
4               -44400000.0          -61200000.0
…                         …                    …
6292           -262000000.0         -108000000.0
6293           -262000000.0         -108000000.0
6294           -262000000.0         -108000000.0
6295           -262000000.0         -108000000.0
6296           -262000000.0         -108000000.0

      Proceeds From Stock Option Exercised  Purchase Of Business  \
0                               50000000.0           116000000.0
1                               50000000.0           116000000.0
2                               50000000.0           116000000.0
3                               50000000.0           116000000.0
4                               50000000.0           116000000.0
…                                        …                     …
6292                                   0.0                   0.0
6293                                   0.0                   0.0
6294                                   0.0                   0.0
6295                                   0.0                   0.0
6296                                   0.0                   0.0

      Purchase Of Investment  Purchase Of PPE  Repurchase Of Capital Stock  \
0              -2.941800e+09     -280800000.0                -2.453600e+09
1              -2.941800e+09     -280800000.0                -2.453600e+09
2              -2.941800e+09     -280800000.0                -2.453600e+09
3              -2.941800e+09     -280800000.0                -2.453600e+09
4              -2.941800e+09     -280800000.0                -2.453600e+09
…                          …                …                            …
6292           -7.636000e+09     -254000000.0                -2.659000e+09
6293           -7.636000e+09     -254000000.0                -2.659000e+09
6294           -7.636000e+09     -254000000.0                -2.659000e+09
6295           -7.636000e+09     -254000000.0                -2.659000e+09
6296           -7.636000e+09     -254000000.0                -2.659000e+09

      Sale Of Investment  Stock Based Compensation  price_change_percentage
0           2.581600e+09               819800000.0                      NaN
1           2.581600e+09               819800000.0                 0.668357
2           2.581600e+09               819800000.0                 0.340335
3           2.581600e+09               819800000.0                 0.697550
4           2.581600e+09               819800000.0                 0.252117
…                      …                         …                        …
6292        1.781000e+09               994000000.0                 0.931295
6293        1.781000e+09               994000000.0                 0.337197
6294        1.781000e+09               994000000.0                -0.615314
6295        1.781000e+09               994000000.0                 0.781573
6296        1.781000e+09               994000000.0                 0.424615
```

```
[6297 rows x 180 columns]
```

```
[ ]: # Standardization using Z-score
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_df = pd.DataFrame(scaler.fit_transform(imputed_data),␣
 ↪columns=imputed_data.columns)
scaled_df
```

```
[ ]:      nvda_open  nvda_high  nvda_low  intel_open  intel_high  intel_low  \
0     -0.908251  -0.901481 -0.906054    0.985476    0.972614   0.973864
1     -0.900662  -0.900228 -0.896459    0.966952    0.976667   0.980211
2     -0.896771  -0.897115 -0.890694    0.987663    0.988827   1.003786
3     -0.888771  -0.894548 -0.883319    1.005355    1.017649   1.024187
4     -0.885859  -0.887676 -0.880223    1.038788    1.022153   1.043682
…           …          …         …          …           …          …
6292   2.996654   2.990886  2.983989    0.051154    0.052997   0.022170
6293   3.013650   3.003983  2.992709    0.018082    0.005710   0.001406
6294   2.982532   3.005195  2.997761    0.001456    0.034082   0.008206
6295   3.021815   3.024033  3.038749    0.043021    0.049394   0.056262
6296   3.043324   3.037696  3.061404    0.063353    0.067408   0.066236

      intel_close  amd_open  amd_high   amd_low  …  Operating Gains Losses  \
0        0.967508 -0.665160 -0.631402 -0.665846  …               -0.179308
1        0.988296 -0.616112 -0.623205 -0.602379  …               -0.179308
2        1.005920 -0.608455 -0.630104 -0.620763  …               -0.179308
3        1.034390 -0.624315 -0.641758 -0.617261  …               -0.179308
4        1.039813 -0.627791 -0.640895 -0.610696  …               -0.179308
…             …         …         …         …  …                       …
6292     0.018053  3.055158  3.030002  2.955720  …               -4.303119
6293     0.001332  3.003885  2.988581  2.972353  …               -4.303119
6294     0.042907  2.952178  3.010370  2.989616  …               -4.303119
6295     0.063244  3.023004  2.995916  2.995989  …               -4.303119
6296     0.055922  3.028653  3.070560  3.071274  …               -4.303119

      Other Non Cash Items  Proceeds From Stock Option Exercised  \
0                -0.082575                             -0.344303
1                -0.082575                             -0.344303
2                -0.082575                             -0.344303
3                -0.082575                             -0.344303
4                -0.082575                             -0.344303
…                        …                                     …
6292             -3.504806                             -0.955048
6293             -3.504806                             -0.955048
6294             -3.504806                             -0.955048
6295             -3.504806                             -0.955048
```

```
6296            -3.504806                            -0.955048

      Purchase Of Business  Purchase Of Investment  Purchase Of PPE  \
0                 0.343126                0.317941        -0.330700
1                 0.343126                0.317941        -0.330700
2                 0.343126                0.317941        -0.330700
3                 0.343126                0.317941        -0.330700
4                 0.343126                0.317941        -0.330700
…                    …                      …               …
6292             -0.348441               -4.111454         1.650499
6293             -0.348441               -4.111454         1.650499
6294             -0.348441               -4.111454         1.650499
6295             -0.348441               -4.111454         1.650499
6296             -0.348441               -4.111454         1.650499

      Repurchase Of Capital Stock  Sale Of Investment  \
0                       -0.140820            0.027929
1                       -0.140820            0.027929
2                       -0.140820            0.027929
3                       -0.140820            0.027929
4                       -0.140820            0.027929
…                          …                 …
6292                    -0.332493           -4.977727
6293                    -0.332493           -4.977727
6294                    -0.332493           -4.977727
6295                    -0.332493           -4.977727
6296                    -0.332493           -4.977727

      Stock Based Compensation  price_change_percentage
0                    -0.130181                      NaN
1                    -0.130181                 0.289390
2                    -0.130181                 0.137291
3                    -0.130181                 0.302926
4                    -0.130181                 0.096386
…                       …                      …
6292                  2.628901                 0.411311
6293                  2.628901                 0.135836
6294                  2.628901                -0.305828
6295                  2.628901                 0.341886
6296                  2.628901                 0.176371

[6297 rows x 180 columns]
```

[ ]:
```
# Calculate the correlation matrix
corr_matrix = scaled_df.corr()
corr_matrix
```

[ ]:

```
                              nvda_open  nvda_high  nvda_low  intel_open  \
nvda_open                      1.000000   0.999903  0.999890   -0.176964
nvda_high                      0.999903   1.000000  0.999810   -0.177261
nvda_low                       0.999890   0.999810  1.000000   -0.177199
intel_open                    -0.176964  -0.177261 -0.177199    1.000000
intel_high                    -0.177046  -0.177224 -0.177297    0.999825
…                                  …          …         …          …
Purchase Of PPE                0.206947   0.206951  0.207592   -0.328038
Repurchase Of Capital Stock   -0.142650  -0.142285 -0.142587   -0.308847
Sale Of Investment            -0.082817  -0.083126 -0.082872   -0.017755
Stock Based Compensation       0.505107   0.504449  0.505747    0.181362
price_change_percentage        0.018928   0.018807  0.018559   -0.009429

                              intel_high  intel_low  intel_close  amd_open  \
nvda_open                      -0.177046  -0.176791    -0.176640  0.721673
nvda_high                      -0.177224  -0.177104    -0.176850  0.722375
nvda_low                       -0.177297  -0.176901    -0.176782  0.720465
intel_open                      0.999825   0.999829     0.999696  0.260532
intel_high                      1.000000   0.999700     0.999826  0.260142
…                                    …          …           …        …
Purchase Of PPE                -0.327430  -0.328454    -0.327795  0.096403
Repurchase Of Capital Stock    -0.308350  -0.309252    -0.308794 -0.145846
Sale Of Investment             -0.017804  -0.017218    -0.017551 -0.210672
Stock Based Compensation        0.181166   0.181608     0.181547  0.383632
price_change_percentage        -0.009043  -0.009678    -0.009538  0.021473

                              amd_high   amd_low  …  Operating Gains Losses  \
nvda_open                     0.719462  0.724059  …                0.035447
nvda_high                     0.720470  0.724760  …                0.035130
nvda_low                      0.718255  0.723207  …                0.035790
intel_open                    0.259905  0.260208  …               -0.231530
intel_high                    0.259784  0.259785  …               -0.231191
…                                  …         …  …                     …
Purchase Of PPE               0.096418  0.097285  …                0.213172
Repurchase Of Capital Stock  -0.144747 -0.146493  …                0.210931
Sale Of Investment           -0.211449 -0.209979  …                0.806801
Stock Based Compensation      0.381653  0.385838  …               -0.150339
price_change_percentage       0.021379  0.020943  …               -0.002283

                              Other Non Cash Items  \
nvda_open                                -0.261761
nvda_high                                -0.261495
nvda_low                                 -0.261854
intel_open                               -0.310318
intel_high                               -0.309880
…                                             …
Purchase Of PPE                           0.602949
```

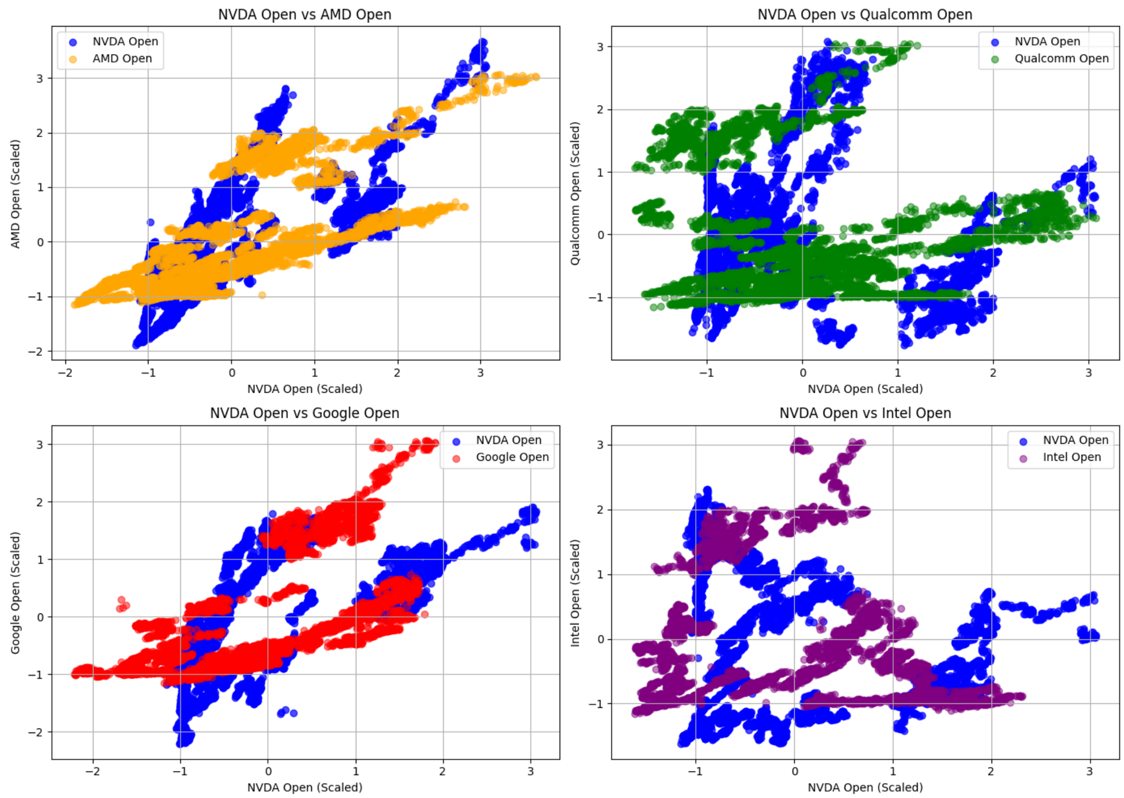
```
Repurchase Of Capital Stock              0.830577
Sale Of Investment                       0.238402
Stock Based Compensation                -0.818246
price_change_percentage                  0.003321

                             Proceeds From Stock Option Exercised  \
nvda_open                                                0.179866
nvda_high                                                0.179699
nvda_low                                                 0.180538
intel_open                                              -0.366275
intel_high                                              -0.365641
…                                                           …
Purchase Of PPE                                          0.862766
Repurchase Of Capital Stock                              0.757172
Sale Of Investment                                       0.112762
Stock Based Compensation                                -0.377223
price_change_percentage                                  0.007222

                             Purchase Of Business  Purchase Of Investment  \
nvda_open                               -0.187500               -0.661899
nvda_high                               -0.187435               -0.661216
nvda_low                                -0.188161               -0.662937
intel_open                               0.359653               -0.023720
intel_high                               0.359013               -0.023834
…                                           …                       …
Purchase Of PPE                         -0.967882               -0.120022
Repurchase Of Capital Stock             -0.854111                0.388354
Sale Of Investment                       0.162970                0.165368
Stock Based Compensation                 0.426761               -0.854523
price_change_percentage                 -0.009573               -0.003420

                             Purchase Of PPE  Repurchase Of Capital Stock  \
nvda_open                           0.206947                    -0.142650
nvda_high                           0.206951                    -0.142285
nvda_low                            0.207592                    -0.142587
intel_open                         -0.328038                    -0.308847
intel_high                         -0.327430                    -0.308350
…                                      …                            …
Purchase Of PPE                     1.000000                     0.867139
Repurchase Of Capital Stock         0.867139                     1.000000
Sale Of Investment                 -0.405091                    -0.333880
Stock Based Compensation           -0.408789                    -0.808959
price_change_percentage             0.011151                     0.008833

                             Sale Of Investment  Stock Based Compensation  \
nvda_open                             -0.082817                  0.505107
nvda_high                             -0.083126                  0.504449
```

```
nvda_low                                -0.082872                  0.505747
intel_open                              -0.017755                  0.181362
intel_high                              -0.017804                  0.181166
…                                           …                         …
Purchase Of PPE                         -0.405091                 -0.408789
Repurchase Of Capital Stock             -0.333880                 -0.808959
Sale Of Investment                       1.000000                  0.119303
Stock Based Compensation                 0.119303                  1.000000
price_change_percentage                 -0.008858                 -0.002953

                             price_change_percentage
nvda_open                                   0.018928
nvda_high                                   0.018807
nvda_low                                    0.018559
intel_open                                 -0.009429
intel_high                                 -0.009043
…                                               …
Purchase Of PPE                             0.011151
Repurchase Of Capital Stock                 0.008833
Sale Of Investment                         -0.008858
Stock Based Compensation                   -0.002953
price_change_percentage                     1.000000

[180 rows x 180 columns]
```

[ ]:

```
# Plotting
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
# Scatter plot for NVDA vs AMD
axes[0, 0].scatter(scaled_df['nvda_open'], scaled_df['amd_open'], color='blue',␣
  ↪alpha=0.7, label='NVDA Open')
axes[0, 0].scatter(scaled_df['amd_open'], scaled_df['nvda_open'],␣
  ↪color='orange', alpha=0.5, label='AMD Open')
axes[0, 0].set_title('NVDA Open vs AMD Open')
axes[0, 0].set_xlabel('NVDA Open (Scaled)')
axes[0, 0].set_ylabel('AMD Open (Scaled)')
axes[0, 0].legend()
axes[0, 0].grid(True)

# Scatter plot for NVDA vs Qualcomm
axes[0, 1].scatter(scaled_df['nvda_open'], scaled_df['qcom_open'],␣
  ↪color='blue', alpha=0.7, label='NVDA Open')
axes[0, 1].scatter(scaled_df['qcom_open'], scaled_df['nvda_open'],␣
  ↪color='green', alpha=0.5, label='Qualcomm Open')
axes[0, 1].set_title('NVDA Open vs Qualcomm Open')
axes[0, 1].set_xlabel('NVDA Open (Scaled)')
axes[0, 1].set_ylabel('Qualcomm Open (Scaled)')
axes[0, 1].legend()
```

```
axes[0, 1].grid(True)

# Scatter plot for NVDA vs Google
axes[1, 0].scatter(scaled_df['nvda_open'], scaled_df['google_open'],␣
 ↪color='blue', alpha=0.7, label='NVDA Open')
axes[1, 0].scatter(scaled_df['google_open'], scaled_df['nvda_open'],␣
 ↪color='red', alpha=0.5, label='Google Open')
axes[1, 0].set_title('NVDA Open vs Google Open')
axes[1, 0].set_xlabel('NVDA Open (Scaled)')
axes[1, 0].set_ylabel('Google Open (Scaled)')
axes[1, 0].legend()
axes[1, 0].grid(True)

# Scatter plot for NVDA vs Intel
axes[1, 1].scatter(scaled_df['nvda_open'], scaled_df['intel_open'],␣
 ↪color='blue', alpha=0.7, label='NVDA Open')
axes[1, 1].scatter(scaled_df['intel_open'], scaled_df['nvda_open'],␣
 ↪color='purple', alpha=0.5, label='Intel Open')
axes[1, 1].set_title('NVDA Open vs Intel Open')
axes[1, 1].set_xlabel('NVDA Open (Scaled)')
axes[1, 1].set_ylabel('Intel Open (Scaled)')
axes[1, 1].legend()
axes[1, 1].grid(True)

plt.tight_layout()
plt.show()
```



[ ]:
```
import numpy as np
# Create an upper triangle matrix of the correlation matrix
upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).
 ↪astype(bool))
upper_triangle
```

[ ]:
```
                          nvda_open  nvda_high  nvda_low  intel_open  \
nvda_open                       NaN   0.999903   0.99989   -0.176964
nvda_high                       NaN        NaN   0.99981   -0.177261
nvda_low                        NaN        NaN       NaN   -0.177199
intel_open                      NaN        NaN       NaN         NaN
intel_high                      NaN        NaN       NaN         NaN
…                                 …          …         …           …
Purchase Of PPE                 NaN        NaN       NaN         NaN
Repurchase Of Capital Stock     NaN        NaN       NaN         NaN
Sale Of Investment              NaN        NaN       NaN         NaN
Stock Based Compensation        NaN        NaN       NaN         NaN
price_change_percentage         NaN        NaN       NaN         NaN

                          intel_high  intel_low  intel_close  amd_open  \
nvda_open                  -0.177046  -0.176791    -0.176640  0.721673
nvda_high                  -0.177224  -0.177104    -0.176850  0.722375
```

```
nvda_low                     -0.177297  -0.176901    -0.176782  0.720465
intel_open                    0.999825   0.999829     0.999696  0.260532
intel_high                         NaN   0.999700     0.999826  0.260142
…                                    …          …          …         …
Purchase Of PPE                    NaN        NaN          NaN       NaN
Repurchase Of Capital Stock        NaN        NaN          NaN       NaN
Sale Of Investment                 NaN        NaN          NaN       NaN
Stock Based Compensation           NaN        NaN          NaN       NaN
price_change_percentage            NaN        NaN          NaN       NaN

                             amd_high   amd_low  …  Operating Gains Losses  \
nvda_open                    0.719462  0.724059  …                0.035447
nvda_high                    0.720470  0.724760  …                0.035130
nvda_low                     0.718255  0.723207  …                0.035790
intel_open                   0.259905  0.260208  …               -0.231530
intel_high                   0.259784  0.259785  …               -0.231191
…                                   …         …  …                       …
Purchase Of PPE                   NaN       NaN  …                     NaN
Repurchase Of Capital Stock       NaN       NaN  …                     NaN
Sale Of Investment                NaN       NaN  …                     NaN
Stock Based Compensation          NaN       NaN  …                     NaN
price_change_percentage           NaN       NaN  …                     NaN

                             Other Non Cash Items  \
nvda_open                               -0.261761
nvda_high                               -0.261495
nvda_low                                -0.261854
intel_open                              -0.310318
intel_high                              -0.309880
…                                               …
Purchase Of PPE                               NaN
Repurchase Of Capital Stock                   NaN
Sale Of Investment                            NaN
Stock Based Compensation                      NaN
price_change_percentage                       NaN

                             Proceeds From Stock Option Exercised  \
nvda_open                                                0.179866
nvda_high                                                0.179699
nvda_low                                                 0.180538
intel_open                                              -0.366275
intel_high                                              -0.365641
…                                                               …
Purchase Of PPE                                               NaN
Repurchase Of Capital Stock                                   NaN
Sale Of Investment                                            NaN
Stock Based Compensation                                      NaN
```

```
price_change_percentage                                    NaN

                         Purchase Of Business  Purchase Of Investment  \
nvda_open                           -0.187500               -0.661899
nvda_high                           -0.187435               -0.661216
nvda_low                            -0.188161               -0.662937
intel_open                           0.359653               -0.023720
intel_high                           0.359013               -0.023834
…                                         …                       …
Purchase Of PPE                           NaN                     NaN
Repurchase Of Capital Stock               NaN                     NaN
Sale Of Investment                        NaN                     NaN
Stock Based Compensation                  NaN                     NaN
price_change_percentage                   NaN                     NaN

                         Purchase Of PPE  Repurchase Of Capital Stock  \
nvda_open                       0.206947                    -0.142650
nvda_high                       0.206951                    -0.142285
nvda_low                        0.207592                    -0.142587
intel_open                     -0.328038                    -0.308847
intel_high                     -0.327430                    -0.308350
…                                    …                            …
Purchase Of PPE                      NaN                     0.867139
Repurchase Of Capital Stock          NaN                          NaN
Sale Of Investment                   NaN                          NaN
Stock Based Compensation             NaN                          NaN
price_change_percentage              NaN                          NaN

                         Sale Of Investment  Stock Based Compensation  \
nvda_open                         -0.082817                  0.505107
nvda_high                         -0.083126                  0.504449
nvda_low                          -0.082872                  0.505747
intel_open                        -0.017755                  0.181362
intel_high                        -0.017804                  0.181166
…                                       …                         …
Purchase Of PPE                   -0.405091                 -0.408789
Repurchase Of Capital Stock       -0.333880                 -0.808959
Sale Of Investment                      NaN                  0.119303
Stock Based Compensation                NaN                       NaN
price_change_percentage                 NaN                       NaN

                         price_change_percentage
nvda_open                               0.018928
nvda_high                               0.018807
nvda_low                                0.018559
intel_open                             -0.009429
intel_high                             -0.009043
```

```
…                                                   …
Purchase Of PPE                              0.011151
Repurchase Of Capital Stock                  0.008833
Sale Of Investment                          -0.008858
Stock Based Compensation                    -0.002953
price_change_percentage                           NaN

[180 rows x 180 columns]
```

[ ]:
```
# Define the correlation threshold
threshold = 0.9

# Identify features with a correlation higher than the threshold
to_drop = [column for column in upper_triangle.columns if␣
 ↪any(upper_triangle[column] > threshold)]
to_drop
```

[ ]:
```
['nvda_high',
 'nvda_low',
 'intel_high',
 'intel_low',
 'intel_close',
 'amd_high',
 'amd_low',
 'amd_close',
 'qcom_high',
 'qcom_low',
 'qcom_close',
 'google_high',
 'google_low',
 'google_close',
 'NVDA_SMA_20',
 'NVDA_EMA_20',
 'NVDA_BBL_5_2.0',
 'NVDA_BBM_5_2.0',
 'NVDA_BBU_5_2.0',
 'NVDA_OBV',
 'NVDA_EMA_Upper',
 'NVDA_EMA_Lower',
 'GDP',
 'Diluted Average Shares',
 'Diluted NI Availto Com Stockholders',
 'EBIT',
 'EBITDA',
 'Gross Profit',
 'Interest Expense Non Operating',
 'Interest Income',
```

```
'Interest Income Non Operating',
'Net Income',
'Net Income Common Stockholders',
'Net Income Continuous Operations',
'Net Income From Continuing And Discontinued Operation',
'Net Income From Continuing Operation Net Minority Interest',
'Net Income Including Noncontrolling Interests',
'Net Interest Income',
'Net Non Operating Interest Income Expense',
'Normalized EBITDA',
'Normalized Income',
'Operating Expense',
'Operating Income',
'Operating Revenue',
'Other Non Operating Income Expenses',
'Pretax Income',
'Reconciled Cost Of Revenue',
'Research And Development',
'Tax Provision',
'Total Expenses',
'Total Operating Income As Reported',
'Total Revenue',
'Accounts Payable',
'Accounts Receivable',
'Additional Paid In Capital',
'Capital Lease Obligations',
'Cash Cash Equivalents And Short Term Investments',
'Common Stock Equity',
'Construction In Progress',
'Current Accrued Expenses',
'Current Assets',
'Current Capital Lease Obligation',
'Current Debt',
'Current Debt And Capital Lease Obligation',
'Current Deferred Liabilities',
'Current Deferred Revenue',
'Current Provisions',
'Goodwill And Other Intangible Assets',
'Gross PPE',
'Inventory',
'Invested Capital',
'Investments And Advances',
'Long Term Capital Lease Obligation',
'Long Term Debt And Capital Lease Obligation',
'Machinery Furniture Equipment',
'Net Debt',
'Net PPE',
```

```
'Net Tangible Assets',
'Non Current Deferred Assets',
'Non Current Deferred Liabilities',
'Non Current Deferred Revenue',
'Non Current Deferred Taxes Assets',
'Non Current Deferred Taxes Liabilities',
'Ordinary Shares Number',
'Other Current Assets',
'Other Current Borrowings',
'Other Current Liabilities',
'Other Equity Adjustments',
'Other Intangible Assets',
'Other Investments',
'Other Non Current Assets',
'Other Non Current Liabilities',
'Other Short Term Investments',
'Payables',
'Payables And Accrued Expenses',
'Raw Materials',
'Receivables',
'Retained Earnings',
'Share Issued',
'Stockholders Equity',
'Tangible Book Value',
'Total Assets',
'Total Capitalization',
'Total Debt',
'Total Equity Gross Minority Interest',
'Total Liabilities Net Minority Interest',
'Total Non Current Assets',
'Total Non Current Liabilities Net Minority Interest',
'Total Tax Payable',
'Work In Process',
'Working Capital',
'Beginning Cash Position',
'Capital Expenditure',
'Cash Flow From Continuing Financing Activities',
'Cash Flow From Continuing Investing Activities',
'Cash Flow From Continuing Operating Activities',
'Change In Account Payable',
'Change In Inventory',
'Change In Other Current Liabilities',
'Change In Payable',
'Change In Receivables',
'Change In Working Capital',
'Changes In Account Receivables',
'Common Stock Payments',
```

```
 'Deferred Income Tax',
 'Deferred Tax',
 'Depreciation Amortization Depletion',
 'Depreciation And Amortization',
 'End Cash Position',
 'Financing Cash Flow',
 'Free Cash Flow',
 'Gain Loss On Investment Securities',
 'Investing Cash Flow',
 'Net Business Purchase And Sale',
 'Net Common Stock Issuance',
 'Net Income From Continuing Operations',
 'Net Investment Purchase And Sale',
 'Net Other Financing Charges',
 'Net PPE Purchase And Sale',
 'Operating Cash Flow',
 'Operating Gains Losses',
 'Other Non Cash Items',
 'Proceeds From Stock Option Exercised',
 'Purchase Of Business',
 'Purchase Of Investment',
 'Purchase Of PPE',
 'Repurchase Of Capital Stock',
 'Stock Based Compensation']
```

[ ]:

```
# Print number of columns before reduction
print(f'Number of columns before reduction: {scaled_df.shape[1]}')

# Drop the identified columns
df_reduced = scaled_df.drop(columns=to_drop)

# Print the number of columns after reduction
num_columns_after_reduction = df_reduced.shape[1]

# Add 14 to this number
num_columns_with_addition = num_columns_after_reduction + 14

# Print the result
print(f'Number of columns after reduction: {num_columns_with_addition }')


# Define the columns you want to keep from scaled_df
columns_to_keep = [
    'nvda_high', 'nvda_low', 'intel_high', 'intel_low', 'intel_close',
    'amd_high', 'amd_low', 'amd_close', 'qcom_high', 'qcom_low', 'qcom_close',
    'google_high', 'google_low', 'google_close'
]
```

```python
# Extract these columns from scaled_df
scaled_df_subset = scaled_df[columns_to_keep]

# Concatenate with df_reduced
df_reduced = pd.concat([scaled_df_subset, df_reduced], axis=1)



# Print the first few rows of df_reduced to inspect
print(df_reduced.head())
```

```
Number of columns before reduction: 180
Number of columns after reduction: 46
   nvda_high  nvda_low  intel_high  intel_low  intel_close  amd_high  \
0  -0.901481 -0.906054    0.972614   0.973864     0.967508 -0.631402
1  -0.900228 -0.896459    0.976667   0.980211     0.988296 -0.623205
2  -0.897115 -0.890694    0.988827   1.003786     1.005920 -0.630104
3  -0.894548 -0.883319    1.017649   1.024187     1.034390 -0.641758
4  -0.887676 -0.880223    1.022153   1.043682     1.039813 -0.640895

    amd_low  amd_close  qcom_high  qcom_low  …  Non Current Prepaid Assets  \
0 -0.665846  -0.616502  -0.409378 -0.498253  …                    0.276922
1 -0.602379  -0.608681  -0.470334 -0.496237  …                    0.276922
2 -0.620763  -0.624322  -0.447730 -0.442178  …                    0.276922
3 -0.617261  -0.628232  -0.418293 -0.417634  …                    0.276922
4 -0.610696  -0.626711  -0.418819 -0.400524  …                    0.276922

   Other Properties  Tradeand Other Payables Non Current  Cash Dividends Paid  \
0          0.171639                             0.359494            -0.412743
1          0.171639                             0.359494            -0.412743
2          0.171639                             0.359494            -0.412743
3          0.171639                             0.359494            -0.412743
4          0.171639                             0.359494            -0.412743

   Change In Prepaid Assets  Changes In Cash  Common Stock Dividend Paid  \
0                 -0.176717         0.019228                    0.822055
1                 -0.176717         0.019228                    0.822055
2                 -0.176717         0.019228                    0.822055
3                 -0.176717         0.019228                    0.822055
4                 -0.176717         0.019228                    0.822055

   Income Tax Paid Supplemental Data  Sale Of Investment  \
0                          -0.218643            0.027929
1                          -0.218643            0.027929
2                          -0.218643            0.027929
3                          -0.218643            0.027929
4                          -0.218643            0.027929
```

```
   price_change_percentage
0                      NaN
1                 0.289390
2                 0.137291
3                 0.302926
4                 0.096386

[5 rows x 46 columns]
```

[ ]:
```
df_reduced
```

[ ]:
```
      nvda_high  nvda_low  intel_high  intel_low  intel_close  amd_high  \
0     -0.901481 -0.906054    0.972614   0.973864     0.967508 -0.631402
1     -0.900228 -0.896459    0.976667   0.980211     0.988296 -0.623205
2     -0.897115 -0.890694    0.988827   1.003786     1.005920 -0.630104
3     -0.894548 -0.883319    1.017649   1.024187     1.034390 -0.641758
4     -0.887676 -0.880223    1.022153   1.043682     1.039813 -0.640895
…           …         …           …          …            …         …
6292   2.990886  2.983989    0.052997   0.022170     0.018053  3.030002
6293   3.003983  2.992709    0.005710   0.001406     0.001332  2.988581
6294   3.005195  2.997761    0.034082   0.008206     0.042907  3.010370
6295   3.024033  3.038749    0.049394   0.056262     0.063244  2.995916
6296   3.037696  3.061404    0.067408   0.066236     0.055922  3.070560

       amd_low  amd_close  qcom_high  qcom_low  …  \
0    -0.665846  -0.616502  -0.409378 -0.498253  …
1    -0.602379  -0.608681  -0.470334 -0.496237  …
2    -0.620763  -0.624322  -0.447730 -0.442178  …
3    -0.617261  -0.628232  -0.418293 -0.417634  …
4    -0.610696  -0.626711  -0.418819 -0.400524  …
…          …          …          …        …  …
6292  2.955720   3.001245   0.442743  0.311178  …
6293  2.972353   2.952151   0.403318  0.415447  …
6294  2.989616   3.021664   0.388073  0.390733  …
6295  2.995989   3.027695   0.351277  0.352352  …
6296  3.071274   3.084761   0.335507  0.360907  …

      Non Current Prepaid Assets  Other Properties  \
0                       0.276922          0.171639
1                       0.276922          0.171639
2                       0.276922          0.171639
3                       0.276922          0.171639
4                       0.276922          0.171639
…                            …                 …
6292                   -5.467225         -5.964780
6293                   -5.467225         -5.964780
```

```
6294                  -5.467225         -5.964780
6295                  -5.467225         -5.964780
6296                  -5.467225         -5.964780

      Tradeand Other Payables Non Current  Cash Dividends Paid  \
0                                0.359494            -0.412743
1                                0.359494            -0.412743
2                                0.359494            -0.412743
3                                0.359494            -0.412743
4                                0.359494            -0.412743
…                                     …                   …
6292                            -0.399651             0.586697
6293                            -0.399651             0.586697
6294                            -0.399651             0.586697
6295                            -0.399651             0.586697
6296                            -0.399651             0.586697

      Change In Prepaid Assets  Changes In Cash  Common Stock Dividend Paid  \
0                    -0.176717         0.019228                    0.822055
1                    -0.176717         0.019228                    0.822055
2                    -0.176717         0.019228                    0.822055
3                    -0.176717         0.019228                    0.822055
4                    -0.176717         0.019228                    0.822055
…                          …               …                         …
6292                 -3.977563         1.594498                   -1.616805
6293                 -3.977563         1.594498                   -1.616805
6294                 -3.977563         1.594498                   -1.616805
6295                 -3.977563         1.594498                   -1.616805
6296                 -3.977563         1.594498                   -1.616805

      Income Tax Paid Supplemental Data  Sale Of Investment  \
0                             -0.218643            0.027929
1                             -0.218643            0.027929
2                             -0.218643            0.027929
3                             -0.218643            0.027929
4                             -0.218643            0.027929
…                                   …                  …
6292                           1.033027           -4.977727
6293                           1.033027           -4.977727
6294                           1.033027           -4.977727
6295                           1.033027           -4.977727
6296                           1.033027           -4.977727

      price_change_percentage
0                         NaN
1                    0.289390
2                    0.137291
```

```
3                     0.302926
4                     0.096386
…                        …
6292                  0.411311
6293                  0.135836
6294                 -0.305828
6295                  0.341886
6296                  0.176371

[6297 rows x 46 columns]
```

[ ]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Sample DataFrame with 32 columns (replace with actual data)
# df_reduced = pd.DataFrame(...)  # Your actual DataFrame

# Define the relevant subset of columns
columns_of_interest = ['nvda_open', 'amd_open', 'qcom_open', 'google_open',␣
 ↪'intel_open', 'nvda_high', 'nvda_low']

# Add the 'nvda_change' column based on the original DataFrame
df_reduced['nvda_change'] = df_reduced['nvda_open'].diff().apply(lambda x:␣
 ↪'Increase' if x > 0 else 'Decrease' )

# Select relevant columns and handle NaNs
subset_df = df_reduced[columns_of_interest + ['nvda_change']].dropna()

# Initialize and apply StandardScaler
scaler = StandardScaler()
scaled_subset_df = pd.DataFrame(scaler.fit_transform(subset_df.
 ↪drop(columns=['nvda_change'])), columns=columns_of_interest)
scaled_subset_df['nvda_change'] = subset_df['nvda_change']  # Add the␣
 ↪'nvda_change' column

# Convert 'nvda_change' to categorical type
scaled_subset_df['nvda_change'] = pd.
 ↪Categorical(scaled_subset_df['nvda_change'])

# Create pair plot
sns.pairplot(scaled_subset_df, hue='nvda_change', palette={'Increase': 'green',␣
 ↪'Decrease': 'red'})
plt.suptitle('Pair Plot with NVDA Change Indicator', y=1.02)
plt.show()
```

Pair Plot with NVDA Change Indicator