Introduction

This Python script adeptly combines data manipulation with pandas, visualization with Matplotlib and Seaborn, statistical modeling with SciPy, and interactive plotting with Plotly to execute comprehensive financial analysis. The data is imported from a CSV file and thoroughly examined through exploratory analysis, which includes summary statistics and the handling of null values. Date conversions are seamlessly performed, and the dataset is enriched with derived features such as the day of the week. Advanced preprocessing techniques like linear interpolation and mean imputation are employed to address missing values, while string representations of numbers are converted into floats for precise numerical analysis. The culmination of this process is a set of rich visualizations of stock prices and volumes over time and the application of polynomial regression to model and forecast stock price trends, all structured into a pandas DataFrame for robust and insightful data processing.The use of the requests library to interact with the Alpha Vantage API, retrieving daily time-series data for notable stocks like Amazon (AMZN), Microsoft (MSFT), and Apple (AAPL).

In [1]:
```
import pandas as pd
import numpy as np
stock = pd.read_csv("C:/Users/shubh/Desktop/MSBA/Projects/Stocks_python/Stock Market Dataset.csv")
stock.head()
```

Out[1]:

	Unnamed: 0	Date	Natural_Gas_Price	Natural_Gas_Vol.	Crude_oil_Price	Crude_oil_Vol.	Copper_Price	Copper_Vol.	Bitcoin_Price	Bitcoin_Vol.	...	Berkshire_Price	Berkshire_Vol.	Netflix_Price	Netflix_Vol.	Amazon_Price	Amazon_Vol.	Meta_Price	Meta_Vol.	Gold_Price	Gold_Vol.
0	0	02-02-2024	2.079	NaN	72.28	NaN	3.8215	NaN	43,194.70	42650.0	...	5,89,498	10580.0	564.64	4030000.0	171.81	117220000.0	474.99	84710000.0	2,053.70	NaN
1	1	01-02-2024	2.050	161340.0	73.82	577940.0	3.8535	NaN	43,081.40	47690.0	...	5,81,600	9780.0	567.51	3150000.0	159.28	66360000.0	394.78	25140000.0	2,071.10	260920.0
2	2	31-01-2024	2.100	142860.0	75.85	344490.0	3.9060	NaN	42,580.50	56480.0	...	5,78,020	9720.0	564.11	4830000.0	155.20	49690000.0	390.14	20010000.0	2,067.40	238370.0
3	3	30-01-2024	2.077	139750.0	77.82	347240.0	3.9110	NaN	42,946.20	55130.0	...	5,84,680	9750.0	562.85	6120000.0	159.00	42290000.0	400.06	18610000.0	2,050.90	214590.0
4	4	29-01-2024	2.490	3590.0	76.78	331930.0	3.8790	NaN	43,299.80	45230.0	...	5,78,800	13850.0	575.79	6880000.0	161.26	42840000.0	401.02	17790000.0	2,034.90	1780.0

5 rows × 39 columns

In [2]:
```
df = pd.DataFrame(stock)
summary = df.describe()
print(summary)

df.isnull().values.sum()
```

```
       Unnamed: 0  Natural_Gas_Price  Natural_Gas_Vol.  Crude_oil_Price  \
count  1243.000000        1243.000000       1239.000000      1243.000000
mean    621.000000           3.494714     131624.116223        67.577064
std     358.967501           1.822540      64385.141749        20.465500
min       0.000000           1.482000       1200.000000       -37.630000
25%     310.500000           2.347500      91900.000000        55.095000
50%     621.000000           2.702000     127370.000000        69.230000
75%     931.500000           4.055500     169460.000000        80.455000
max    1242.000000           9.647000     381970.000000       123.700000

       Crude_oil_Vol.  Copper_Price    Copper_Vol.  Bitcoin_Vol.  \
count    1.220000e+03   1243.000000    1206.000000  1.243000e+03
mean     3.989038e+05      3.541957   35406.616915  4.033918e+07
std      2.161619e+05      0.702819   38415.448731  2.940889e+08
min      1.702000e+04      2.100500      10.000000  2.600000e+02
25%      2.835975e+05      2.858750     370.000000  7.907500e+04
50%      3.668850e+05      3.666000   10180.000000  2.153100e+05
75%      5.072425e+05      4.137250   68340.000000  6.151050e+05
max      1.770000e+06      4.937500  176040.000000  4.470000e+09

       Platinum_Vol.  Ethereum_Vol.  ...  Nvidia_Price   Nvidia_Vol.  \
count     636.000000   1.243000e+03  ...   1243.000000  1.243000e+03
mean     9082.515723   1.801563e+07  ...    187.285841  4.560298e+07
std      8876.538587   1.326933e+08  ...    134.679941  1.869107e+07
min         0.000000   7.518000e+04  ...     33.450000  9.790000e+06
25%      1120.000000   5.883600e+05  ...     73.905000  3.245000e+07
50%      6070.000000   1.570000e+06  ...    151.590000  4.279000e+07
75%     15287.500000   9.365000e+06  ...    242.140000  5.511500e+07
max     42830.000000   1.790000e+09  ...    661.600000  1.534600e+08

       Berkshire_Vol.  Netflix_Price  Netflix_Vol.  Amazon_Price  \
count     1243.000000    1243.000000  1.243000e+03   1243.000000
mean      2426.524537     404.839541  7.057401e+06    128.683234
std       2660.497572     114.989473  6.384187e+06     30.808631
min         80.000000     166.370000  1.140000e+06     79.410000
25%        345.000000     323.010000  3.990000e+06     96.260000
50%       1510.000000     384.150000  5.610000e+06    128.730000
75%       3225.000000     495.365000  7.910000e+06    158.110000
max      13850.000000     691.690000  1.333900e+08    186.570000

        Amazon_Vol.   Meta_Price     Meta_Vol.      Gold_Vol.
count  1.243000e+03  1243.000000  1.243000e+03    1241.000000
mean   7.413005e+07   239.728134  2.325851e+07  211127.671233
std    3.245753e+07    71.015427  1.555486e+07  115006.351292
min    1.763000e+07    88.910000  5.470000e+06       0.000000
25%    5.264500e+07   183.355000  1.478500e+07  152200.000000
50%    6.520000e+07   224.430000  1.934000e+07  197970.000000
75%    8.674500e+07   301.650000  2.711500e+07  257920.000000
max    3.113500e+08   474.990000  2.304100e+08  813410.000000

[8 rows x 31 columns]
721
```

In [3]:
```
stock.tail()
```

Out[3]:

	Unnamed: 0	Date	Natural_Gas_Price	Natural_Gas_Vol.	Crude_oil_Price	Crude_oil_Vol.	Copper_Price	Copper_Vol.	Bitcoin_Price	Bitcoin_Vol.	...	Berkshire_Price	Berkshire_Vol.	Netflix_Price	Netflix_Vol.	Amazon_Price	Amazon_Vol.	Meta_Price	Meta_Vol.	Gold_Price	Gold_Vol.
1238	1238	08-02-2019	2.583	147880.0	52.72	621000.0	2.8140	270.0	3,661.70	699230.0	...	3,00,771	240.0	347.57	7560000.0	79.41	113150000.0	167.33	12560000.0	1,318.50	150610.0
1239	1239	07-02-2019	2.551	211790.0	52.64	749010.0	2.8320	320.0	3,397.70	471360.0	...	3,02,813	240.0	344.71	7860000.0	80.72	92530000.0	166.38	17520000.0	1,314.20	166760.0
1240	1240	06-02-2019	2.662	98330.0	54.01	606720.0	2.8400	100.0	3,404.30	514210.0	...	3,08,810	120.0	352.19	6720000.0	82.01	78800000.0	170.49	13280000.0	1,314.40	137250.0
1241	1241	05-02-2019	2.662	82250.0	53.66	609760.0	2.8205	90.0	3,468.40	460950.0	...	3,10,700	360.0	355.81	9050000.0	82.94	89060000.0	171.16	22560000.0	1,319.20	129010.0
1242	1242	04-02-2019	2.660	116490.0	54.56	622470.0	2.7975	490.0	3,462.80	503920.0	...	3,12,000	310.0	351.34	9050000.0	81.67	98580000.0	169.25	20040000.0	1,319.30	159560.0

5 rows × 39 columns

In [4]:
```
stock.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1243 entries, 0 to 1242
Data columns (total 39 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         1243 non-null   int64
 1   Date               1243 non-null   object
 2   Natural_Gas_Price  1243 non-null   float64
 3   Natural_Gas_Vol.   1239 non-null   float64
 4   Crude_oil_Price    1243 non-null   float64
 5   Crude_oil_Vol.     1220 non-null   float64
 6   Copper_Price       1243 non-null   float64
 7   Copper_Vol.        1206 non-null   float64
 8   Bitcoin_Price      1243 non-null   object
 9   Bitcoin_Vol.       1243 non-null   float64
 10  Platinum_Price     1243 non-null   object
 11  Platinum_Vol.      636 non-null    float64
 12  Ethereum_Price     1243 non-null   object
 13  Ethereum_Vol.      1243 non-null   float64
 14  S&P_500_Price      1243 non-null   object
 15  Nasdaq_100_Price   1243 non-null   object
 16  Nasdaq_100_Vol.    1242 non-null   float64
 17  Apple_Price        1243 non-null   float64
 18  Apple_Vol.         1243 non-null   float64
 19  Tesla_Price        1243 non-null   float64
 20  Tesla_Vol.         1243 non-null   float64
 21  Microsoft_Price    1243 non-null   float64
 22  Microsoft_Vol.     1243 non-null   float64
 23  Silver_Price       1243 non-null   float64
 24  Silver_Vol.        1196 non-null   float64
 25  Google_Price       1243 non-null   float64
 26  Google_Vol.        1243 non-null   float64
 27  Nvidia_Price       1243 non-null   float64
 28  Nvidia_Vol.        1243 non-null   float64
 29  Berkshire_Price    1243 non-null   object
 30  Berkshire_Vol.     1243 non-null   float64
 31  Netflix_Price      1243 non-null   float64
 32  Netflix_Vol.       1243 non-null   float64
 33  Amazon_Price       1243 non-null   float64
 34  Amazon_Vol.        1243 non-null   float64
 35  Meta_Price         1243 non-null   float64
 36  Meta_Vol.          1243 non-null   float64
 37  Gold_Price         1243 non-null   object
 38  Gold_Vol.          1241 non-null   float64
dtypes: float64(30), int64(1), object(8)
memory usage: 378.9+ KB
```

In [ ]:
```
```

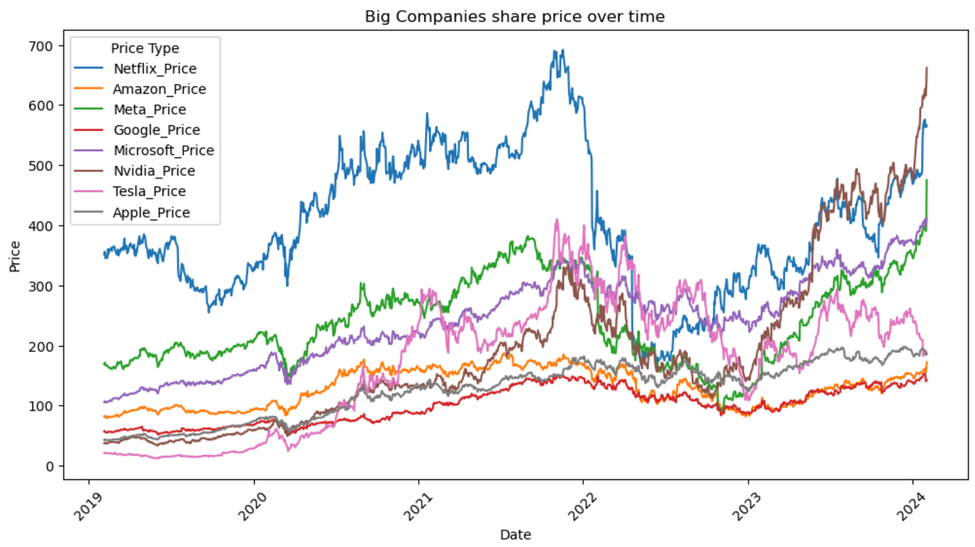
In [12]:
```
## To handle missing values in my dataset with a robust approach, I'll perform the following steps:

##For price columns: Use the previous day's price if the missing value falls on a Saturday or Sunday.
#Use linear interpolation for other days.

##For volume columns: Fill with 0 for Saturday and Sunday.
#For other days, fill missing values with the mean of the previous and next available day's volume.)
```

In [5]:
```
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')  # For "DD-MM-YYYY" format

df['DayOfWeek'] = df['Date'].dt.dayofweek  # Monday=0, Sunday=6


# Identifying price and volume columns
price_columns = [col for col in df.columns if 'Price' in col]
volume_columns = [col for col in df.columns if 'Vol' in col]

# Fill price columns
for col in price_columns:
    # Linear interpolation for initial filling
    df[col] = df[col].interpolate(method='linear')

    # Adjust for weekends
    for i in range(len(df)):
        if pd.isnull(df.loc[i, col]):
            if df.loc[i, 'DayOfWeek'] in [5, 6]:  # Saturday or Sunday
                # Use previous day's price if available
                if i > 0 and pd.notnull(df.loc[i-1, col]):
                    df.loc[i, col] = df.loc[i-1, col]

# Fill volume columns
for col in volume_columns:
    for i in range(len(df)):
        if pd.isnull(df.loc[i, col]):
            if df.loc[i, 'DayOfWeek'] in [5, 6]:  # Saturday or Sunday
                df.loc[i, col] = 0
            else:
                # Use mean of previous and next day's volume if available
                prev_val = next_val = None
                if i > 0:
                    prev_val = df.loc[i-1, col]
                if i < len(df) - 1:
                    next_val = df.loc[i+1, col]
                if pd.notnull(prev_val) and pd.notnull(next_val):
                    df.loc[i, col] = (prev_val + next_val) / 2
                elif pd.notnull(prev_val):
                    df.loc[i, col] = prev_val
                elif pd.notnull(next_val):
                    df.loc[i, col] = next_val

# Drop the 'DayOfWeek' column if no longer needed
df.drop('DayOfWeek', axis=1, inplace=True)
```

In [6]:
```
numerical_columns = df.select_dtypes(include=['object']).columns

# Convert columns with numeric strings to float, removing commas
for col in numerical_columns:
    try:
        df[col] = df[col].str.replace(',', '').astype(float)
    except ValueError:
        # This handles cases where conversion is not possible due to non-numeric values
        # If a column contains genuinely non-numeric data (e.g., dates or text), it won't be converted
        continue
```

In [7]:
```
# Reshape the DataFrame to Long format
import matplotlib.pyplot as plt
import seaborn as sns
df_long = pd.melt(df, id_vars=['Date'], value_vars=['Netflix_Price' , 'Amazon_Price' , 'Meta_Price' , 'Google_Price' , 'Microsoft_Price' , 'Nvidia_Price' , 'Tesla_Price','Apple_Price'],
                  var_name='Type', value_name='Price')

# Plotting
plt.figure(figsize=(12, 6))
```

```
sns.lineplot(data=df_long, x='Date', y='Price', hue='Type')
plt.title('Big Companies share price over time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.legend(title='Price Type')
plt.show()
```



The stock price of Netflix shows significant volatility, with a sharp increase peaking around 2021, followed by a steep decline and then a gradual recovery.

Amazon's stock price exhibits a steady upward trend with less volatility compared to Netflix.

Meta's stock price also shows growth, with some fluctuations. It has periods of rapid increase as well as declines.

Google's stock shows an overall increasing trend with some periods of volatility. The stock price appears to be recovering after a decline.

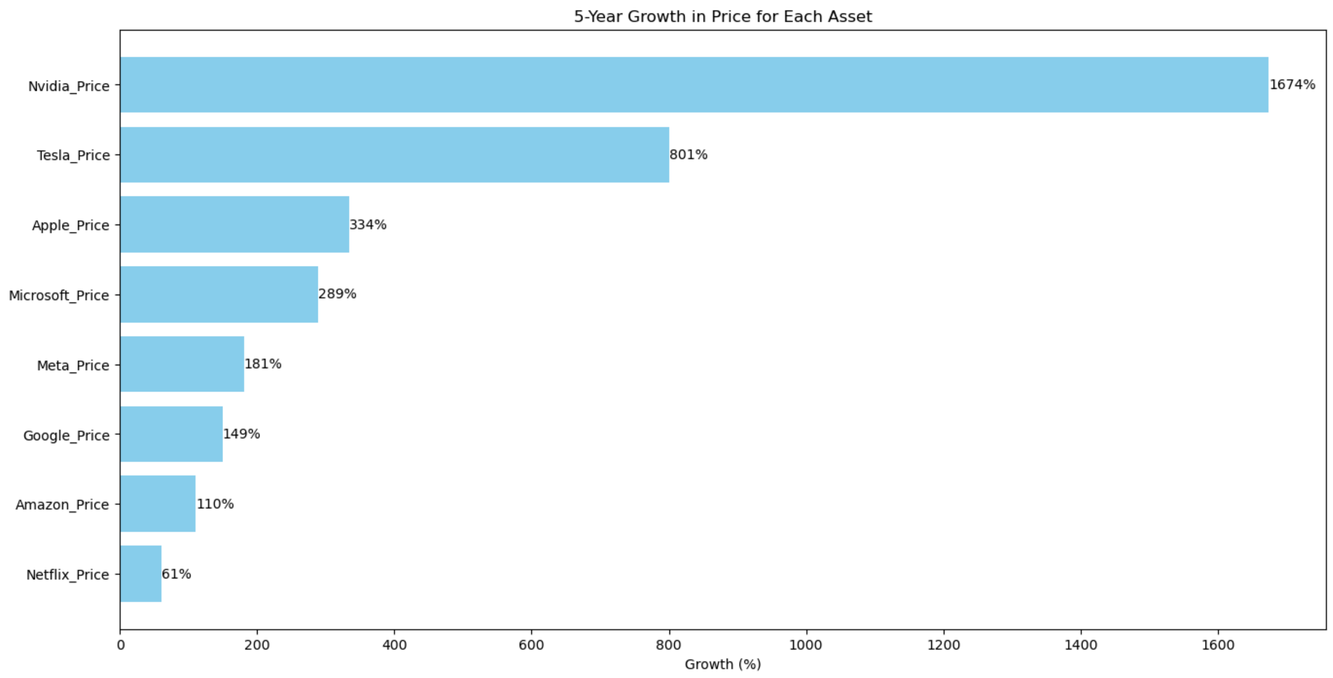
Microsoft displays a relatively stable and consistent upward trend in stock price. This kind of trend is often indicative of a company with solid fundamentals and a strong market position.

Nvidia's stock price shows a very steep increase, especially from around late 2020 through 2021, which may be due to the high demand for GPUs for gaming and data centers, as well as interest in the company's developments in AI and deep learning.

Tesla's stock has experienced substantial growth, particularly noticeable after 2020.

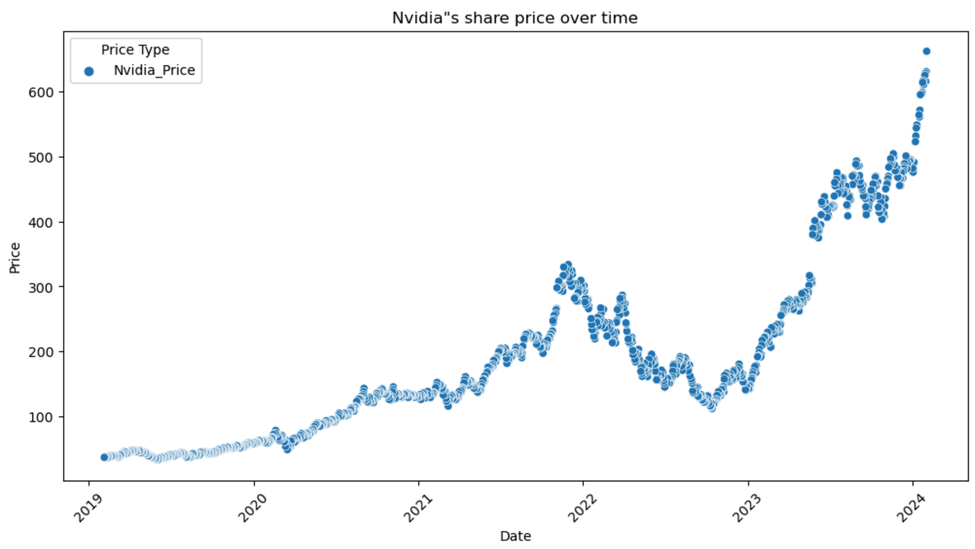
Apple's stock price shows steady growth with some periods of quicker increases. The company's strong brand, product ecosystem, and financials likely contribute to investor confidence.

In [8]:
```
# Percent change over last 5 years

# Explicitly list the columns for price data
price_columns = [
    'Apple_Price', 'Netflix_Price', 'Amazon_Price',
    'Meta_Price', 'Google_Price', 'Microsoft_Price', 'Nvidia_Price',
    'Tesla_Price'
]

# Ensure only the columns present in df are processed
price_columns = [col for col in price_columns if col in df.columns]

# Calculate initial and final prices
initial_prices = df[price_columns].iloc[0]  # Assuming the DataFrame is sorted by date, get the first row
final_prices = df[price_columns].iloc[-1]  # Get the last row

# Calculate growth percentage for each price column
growth = ((initial_prices - final_prices) / final_prices) * 100

# Prepare the data for plotting
growth_df = growth.reset_index()
growth_df.columns = ['Asset', 'Growth']
growth_df.sort_values('Growth', ascending=True, inplace=True)

# Plotting
plt.figure(figsize=(16, 8))
bars = plt.barh(growth_df['Asset'], growth_df['Growth'], color='skyblue')
plt.xlabel('Growth (%)')
plt.title('5-Year Growth in Price for Each Asset')
# Adding data labels
for bar in bars:
    plt.text(
        bar.get_width(),  # x-coordinate position of text
        bar.get_y() + bar.get_height() / 2,  # y-coordinate position of text
        f"{bar.get_width():.0f}%",  # text
        va='center'  # center alignment
    )
plt.show()
```



The bar chart illustrates the 5-year stock price growth for several leading tech companies, highlighting Nvidia's market-leading surge at 1674% and Tesla's significant 801% increase, which can be attributed to their innovation and market expansion. Apple and Microsoft also show robust growth of 334% and 289%, respectively, reflecting their strong product ecosystems and strategic pivots towards high-demand sectors like cloud computing. At the lower end, Amazon and Netflix display more modest growths of 110% and 61%, pointing to intense market competition and evolving industry landscapes.

In [9]:
```
# Plotting
df_long1 = pd.melt(df, id_vars=['Date'], value_vars=['Nvidia_Price'],
                   var_name='Type', value_name='Price')
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df_long1, x='Date', y='Price', hue='Type')
plt.title('Nvidia"s share price over time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.legend(title='Price Type')
plt.show()
```

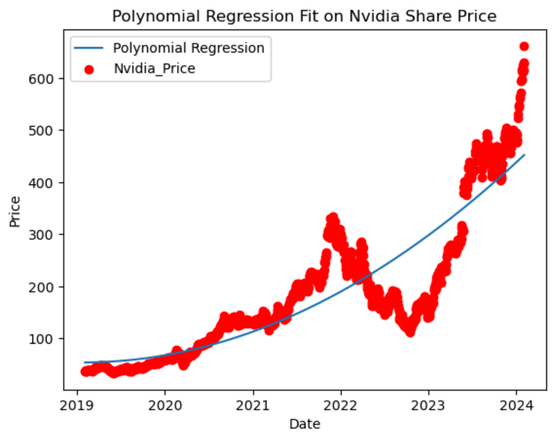


The scatter plot traces Nvidia's share price from 2019 to 2024, highlighting a significant growth trajectory with marked volatility. Initially, the price rises modestly until early 2021, when it experiences a steep ascent, indicating robust market performance. A notable downturn occurs later in 2021, possibly reflecting market corrections or company-specific events. The price recovers sharply thereafter, entering a phase of rapid growth that suggests strong investor confidence and possibly successful business initiatives. The plot's upward trend, interspersed with fluctuations, underscores the dynamic nature of the tech market and Nvidia's position within it. The price's peak in 2024 indicates a high point in Nvidia's valuation, potentially aligning with technological advancements and market demand.

In [10]:
```
from scipy import stats
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression


# Convert 'Date' to datetime
df_long1['Date'] = pd.to_datetime(df_long1['Date'])
# Convert 'Date' to a numerical value, e.g., days since the minimum date
df_long1['DateNumeric'] = (df_long1['Date'] - df_long1['Date'].min()).dt.days

# Prepare X and y for fitting the model
X = df_long1[['DateNumeric']]  # Features matrix must be 2D
y = df_long1['Price']  # Target variable

# Build a polynomial regression pipeline
pipeline = make_pipeline(PolynomialFeatures(2), LinearRegression())

# Use the pipeline to build the model
pipeline.fit(X, y)

# Predict using the model
pred = pipeline.predict(X)
print(pred)
# For plotting, sort the DataFrame by 'DateNumeric' to ensure the line plot makes sense
df_sorted = df_long1.sort_values(by='DateNumeric')
plt.plot(df_sorted['Date'], pipeline.predict(df_sorted[['DateNumeric']]), label='Polynomial Regression')
plt.scatter(df_sorted['Date'], df_sorted['Price'], color='red', label='Nvidia_Price')
plt.legend()
plt.xlabel('Date')
```

```
plt.ylabel('Price')
plt.title('Polynomial Regression Fit on Nvidia Share Price')
plt.show()
```

```
[451.94950954 451.51530523 451.08133789 ...  53.94204933  53.93959988
  53.93738739]
```



The graph presents a polynomial regression analysis of Nvidia's stock price from 2019 to 2024. The regression line smooths out the fluctuations to model the underlying trend of the data, demonstrating Nvidia's overall price growth trajectory. Initial growth is moderate, with a notable dip in 2021, followed by a sharp recovery and a steep increase into 2024. The polynomial nature of the fitted line captures the non-linear pattern of price changes, potentially reflecting market cycles, investor sentiment, and company performance over time. The uptick towards the end of the period suggests a bullish outlook for Nvidia. The graph also indicates a growing divergence between the regression curve and the actual data points as time progresses, hinting at increasing volatility or speculative trading influencing the stock price.

In [11]:
```
import requests

# List of stock symbols to analyze
stocks = ['AMZN','MSFT', 'AAPL'] #Amazon, Microsoft, Apple

# Alpha Vantage API key
api_key = '0DJ81O0URKV5W8TR'

time_series_data = []

for stock in stocks:
    api_url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={stock}&apikey={api_key}'

    # Make API request and The response/status from the server is stored in the response variable.
    response = requests.get(api_url)
    if response.status_code == 200:#chacing status if successful

        # Extract the time series data in json format from the response
        data = response.json()['Time Series (Daily)']
        #extracts the daily time series data from the JSON response returned by the Alpha Vantage API.
        #response.json() converts the JSON response into a Python dictionary
        #['Time Series (Daily)'] accesses the portion of the dictionary that contains the daily time series data.


        for date, values in data.items():
            row = {'date': date, 'ticker': stock}#Dictionary
            row.update(values)
            time_series_data.append(row)#row dictionary appends to Time_series_data list populate it daily
    else:
        print(f"Error fetching data for {stock}: {response.status_code}")

# Normalize the list into a DataFrame(DataFrame organizes the stock data into a structured format)
df = pd.DataFrame(time_series_data)

#df.to_csv('df.csv', index = False)
```

In [12]:
```
df
```

Out[12]:

	date	ticker	1. open	2. high	3. low	4. close	5. volume
0	2024-03-28	AMZN	180.1700	181.7000	179.2600	180.3800	38051588
1	2024-03-27	AMZN	179.8800	180.0000	177.3099	179.8300	33272551
2	2024-03-26	AMZN	180.1500	180.4500	177.9500	178.3000	29658982
3	2024-03-25	AMZN	178.0100	180.9900	177.2400	179.7100	29815464
4	2024-03-22	AMZN	177.7520	179.2550	176.7500	178.8700	27995378
...	...	...	...	...	...	...	...
295	2023-11-09	AAPL	182.9600	184.1200	181.8100	182.4100	53763540
296	2023-11-08	AAPL	182.3500	183.4500	181.5900	182.8900	49340282
297	2023-11-07	AAPL	179.1800	182.4400	178.9700	181.8200	70529966
298	2023-11-06	AAPL	176.3800	179.4300	176.2100	179.2300	63841310
299	2023-11-03	AAPL	174.2400	176.8200	173.3500	176.6500	79829246

300 rows × 7 columns

In [13]:
```
type(df['date'][0])
```

Out[13]:
```
str
```

In [14]:
```
df.columns = list(df.columns.str.replace('[0-9.]','',regex = True))
#removing numbers from the column names in data frame
df.columns
```

Out[14]:
```
Index(['date', 'ticker', ' open', ' high', ' low', ' close', ' volume'], dtype='object')
```

In [15]:
```
list(df.columns.str.strip())
#Strip function removes the white spaces at start or at end here from column names
```

Out[15]:
```
['date', 'ticker', 'open', 'high', 'low', 'close', 'volume']
```

In [16]:
```
df.columns = df.columns.str.strip()
df
```

Out[16]:

	date	ticker	open	high	low	close	volume
0	2024-03-28	AMZN	180.1700	181.7000	179.2600	180.3800	38051588
1	2024-03-27	AMZN	179.8800	180.0000	177.3099	179.8300	33272551
2	2024-03-26	AMZN	180.1500	180.4500	177.9500	178.3000	29658982
3	2024-03-25	AMZN	178.0100	180.9900	177.2400	179.7100	29815464
4	2024-03-22	AMZN	177.7520	179.2550	176.7500	178.8700	27995378
...	...	...	...	...	...	...	...
295	2023-11-09	AAPL	182.9600	184.1200	181.8100	182.4100	53763540
296	2023-11-08	AAPL	182.3500	183.4500	181.5900	182.8900	49340282
297	2023-11-07	AAPL	179.1800	182.4400	178.9700	181.8200	70529966
298	2023-11-06	AAPL	176.3800	179.4300	176.2100	179.2300	63841310
299	2023-11-03	AAPL	174.2400	176.8200	173.3500	176.6500	79829246

300 rows × 7 columns

In [17]:
```
# Convert 'date' column to datetime type for Analysis
df['date'] = pd.to_datetime(df['date'])
```

In [18]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Group the DataFrame by 'ticker'
grouped = df.groupby('ticker')

# Iterate over each group
for ticker, data in grouped:
    # Create a figure with secondary y-axis for each ticker
    fig = make_subplots(specs=[[{"secondary_y": True}]])

    # Add closing price trace
    fig.add_trace(
        go.Scatter(x=data['date'], y=data['close'].astype(float), name=f"{ticker} Close Price", line=dict(color='royalblue')),
        secondary_y=False,
    )

    # Add volume trace
    fig.add_trace(
        go.Scatter(x=data['date'], y=data['volume'].astype(float), name=f"{ticker} Volume", line=dict(color='tomato', dash='dot')),
        secondary_y=True,
    )

    # Add figure title and adjust layout for each ticker
    fig.update_layout(
        title_text=f"{ticker} Stock Closing Prices and Volume",
        xaxis_title="Date",
        template="plotly_white",
    )

    # Set x-axis properties
    fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True)

    # Set y-axes titles and make them start from 0
    fig.update_yaxes(title_text=f"<b>{ticker} Close Price</b>", secondary_y=False, showline=True, linewidth=2, linecolor='blue', mirror=True, rangemode='tozero')
    fig.update_yaxes(title_text=f"<b>{ticker} Volume</b>", secondary_y=True, showline=True, linewidth=2, linecolor='red', mirror=True, rangemode='tozero')

    # Show plot for each ticker
    fig.show()
```

The three graphs illustrate the closing prices and trading volumes of Apple (AAPL), Amazon (AMZN), and Microsoft (MSFT) stocks over a period from November 2023 to March 2024.

For AAPL, the closing price remains relatively stable with slight fluctuations, while the volume shows more variability with several spikes, which may indicate specific events that caused a significant number of shares to be traded.

AMZN's graph shows a more volatile closing price with a sharp drop and subsequent recovery around the new year, which is not as prominently reflected in the trading volume.

MSFT's chart exhibits a general uptrend in closing price with minor dips. The trading volume for Microsoft appears less volatile compared to AMZN, but like AAPL, it has a few noticeable peaks, which suggest days of heavy trading possibly triggered by news releases, earnings reports, or market rumors.

While MSFT's stock demonstrates a stable and positive price trend suggesting consistent investor confidence, AMZN's fluctuating prices reflect market uncertainties, and AAPL's steady prices amid fluctuating volumes indicate stable valuations despite variable trading activity.

Observation

The visualizations derived provides a clear picture of the market behaviors of prominent companies like Apple, Amazon, and Microsoft, showcasing Apple's price stability, Amazon's noticeable volatility, and Microsoft's steady ascension, reflecting the distinct market conditions and corporate developments. Additionally, the analysis of trading volumes for these stocks reveals notable spikes, hinting at the impact of market news or significant corporate events. Utilizing polynomial regression on Nvidia's stock price, the script uncovers intricate, non-linear trends that govern market movements, emphasizing the broader tendencies that emerge from daily price volatility. This analytical prowess is powered by real-time financial data, seamlessly accessed through the Alpha V antage API and elegantly transformed into an accessible pandas DataFrame format for further analysis. Such insights underscore Python's powerful capabilities in financial data processing and visualization, providing a deep understanding of stock performance dynamics within the fluid and complex market environment.

In [ ]: