Big Data Processing on NVIDEA Stock

Part 1: Big Data Processing:

Task 1: Data Cleaning and Exploration:

1. Introduction:

- The objective of the task is to work with stock market data (in this case, NVDA) and apply data preprocessing, regression, and classification techniques to understand how the stock behaves.
- You will focus on the NVDA stock price dataset and use machine learning models to predict trends, stock prices, and other relevant information. The provided dataset includes various features like Open, High, Low, Close, Volume, etc.

2. **Loading and Exploring the Data**:

- The NVDA dataset must be loaded into a Pandas DataFrame before its structure can be examined.
- Here, the CSV file was loaded using pd.read_csv(), and the data types and missing values were checked using df.info(). This makes it easier to determine which columns include non-numeric data or missing values, which we'll need to deal with in the following steps.
- **Results**: The dataset will contain multiple columns like Open, High, Low, Close, Volume, Date, etc. The info() function shows the number of non-null entries in each column.

```
Column Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 523 entries, 0 to 522
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       522 non-null    object
 1   Open       521 non-null    float64
 2   High       522 non-null    float64
 3   Low        523 non-null    float64
 4   Close      522 non-null    float64
 5   Adj Close  523 non-null    float64
 6   Volume     521 non-null    float64
dtypes: float64(6), object(1)
memory usage: 28.7+ KB
None
        Date    Open       High     Low   Close  Adj Close       Volume
0  2022/7/1  14.899  15.063000  14.392  14.523  14.506663  577610000.0
1  2022/7/5  14.175  14.971000  14.055  14.964  14.947166  651397000.0
2  2022/7/6  15.010  15.319000  14.789  15.130  15.112980  529066000.0
3  2022/7/7  15.456  15.945000  15.389  15.858  15.840160  492903000.0
4  2022/7/8  15.430  16.037001  15.389  15.838  15.820185  467972000.0
```

3. **Handling Missing Values**:

- Missing data can have a significant impact on model performance. So, we checked for missing values using df.isnull().sum(), which gives a count of missing values in each column.
- The strategy used for missing value imputation is to forward-fill (df.ffill()), meaning if any value is missing, it gets filled with the previous day's value. This is a common method when dealing with time-series data.

```
Values missing before cleaning:
Date         1
Open         2
High         1
Low          0
Close        1
Adj Close    0
Volume       2
dtype: int64

Missing values after cleaning:
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64

First few rows after handling missing values:
        Date    Open       High     Low   Close  Adj Close       Volume
0  2022/7/1  14.899  15.063000  14.392  14.523  14.506663  577610000.0
1  2022/7/5  14.175  14.971000  14.055  14.964  14.947166  651397000.0
2  2022/7/6  15.010  15.319000  14.789  15.130  15.112980  529066000.0
3  2022/7/7  15.456  15.945000  15.389  15.858  15.840160  492903000.0
4  2022/7/8  15.430  16.037001  15.389  15.838  15.820185  467972000.0
```

- **Results**: After applying the missing value handling, we observe that the data for "Date" is clean, and the forward-fill method fills the missing numeric values across all rows.

4. Convert the date coloumn to a different datetime format:

```
df = df.assign(Date=pd.to_datetime(df['Date']))
first_rows = df.head()
print(first_rows)
```

```
        Date    Open       High     Low   Close  Adj Close       Volume
0 2022-07-01  14.899  15.063000  14.392  14.523  14.506663  577610000.0
1 2022-07-05  14.175  14.971000  14.055  14.964  14.947166  651397000.0
2 2022-07-06  15.010  15.319000  14.789  15.130  15.112980  529066000.0
3 2022-07-07  15.456  15.945000  15.389  15.858  15.840160  492903000.0
4 2022-07-08  15.430  16.037001  15.389  15.838  15.820185  467972000.0
```

5. Compute Basic statistics:

- Here, I computed the basic statistics like min, max, median, standard deviation for each numerical feature.

```
stats = df.describe(include='all')
print(stats)
```

```
                                Date        Open        High         Low  \
count                            522  522.000000  522.000000  522.000000
mean   2023-07-15 15:29:39.310344704   46.763362   47.616234   45.853726
min              2022-07-01 00:00:00   10.971000   11.735000   10.813000
25%              2023-01-06 18:00:00   18.165500   18.736000   17.895249
50%              2023-07-17 12:00:00   42.287498   42.948999   41.651998
75%              2024-01-22 18:00:00   59.929250   60.225751   58.948750
max              2024-07-31 00:00:00  139.800003  140.759995  132.419998
std                              NaN   32.827653   33.407052   32.044071

            Close   Adj Close        Volume
count  522.000000  522.000000  5.220000e+02
mean    46.788335   46.773798  4.838382e+08
min     11.227000   11.217702  1.679340e+08
25%     18.361499   18.340843  3.841098e+08
50%     42.309500   42.296837  4.574970e+08
75%     59.818251   59.810533  5.513095e+08
max    135.580002  135.580002  1.543911e+09
std     32.726253   32.724067  1.574563e+08
```

6. Plot the closing price over time using Matplotlib:

- Here, based on what we did in the above steps, we plot the graph for the same.

5) Plot the closing price over time using Matplotlib.,

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(df['Date'], df['Close'], label='Close Price', color='green')

ax.set_title('Closing Price Over Time', fontsize=14)
ax.set_xlabel('Date', fontsize=12)
ax.set_ylabel('Closing Price (USD)', fontsize=12)
ax.tick_params(axis='x', rotation=45)
ax.grid(True)
ax.legend()
plt.tight_layout()
plt.show()
```

<u>Task 2: Feature Engineering:</u>

1. Here, I had to create a coloumn for daily returns based on the adjusted closing price and print the top 10 dates with the highest daily return. It was done as shown in the below picture.,

```
[43]: df['Daily Return'] = pd.to_numeric(df['Adj Close'].pct_change(), errors='coe
      df.at[0, 'Daily Return'] = 0
      top_10_returns = df.loc[:, ['Date', 'Daily Return']].nlargest(10, 'Daily Ret
      print("Dates with Highest Daily Returns:")
      print(top_10_returns)
```

```
Dates with Highest Daily Returns:
          Date  Daily Return
226 2023-05-25      0.243696
412 2024-02-22      0.164009
92  2022-11-10      0.143293
162 2023-02-23      0.140214
522 2024-07-31      0.128121
476 2024-05-23      0.093197
285 2023-08-21      0.084713
105 2022-11-30      0.082379
17  2022-07-27      0.076030
140 2023-01-23      0.075901
```

2) # Calculate the 7-day moving average of the closing price

2. The next step, I had to calculate the 7-day moving average of the closing price and plot the graph and it is explained in the picture below.,

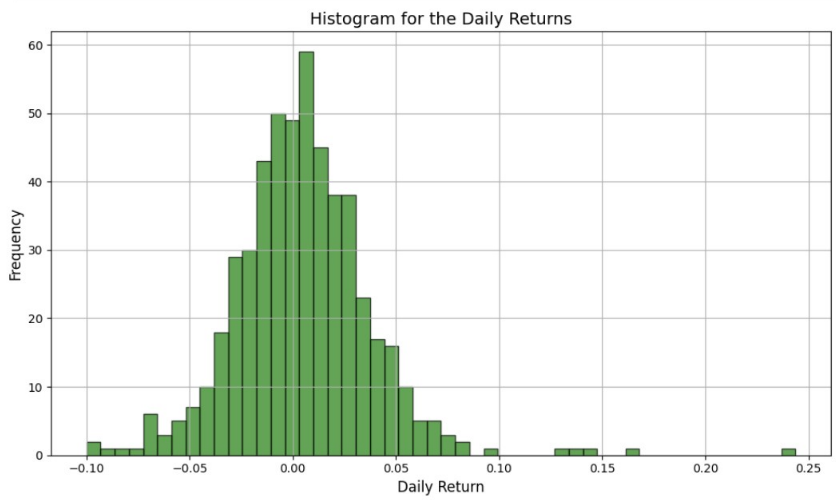
3. The last step in Feature Engineering was to normalize the trading volume coloumn using Min-Max Scaling and print the top 10 dates with the highest volume. Once the code was implemented, we got result like the picture below.,

```
print(top_10_volume)

Top 10 Dates with Highest Normalized Trading Volume:
          Date  Normalized Volume
226 2023-05-25           1.000000
43  2022-09-01           0.734701
288 2023-08-24           0.718115
423 2024-03-08           0.708104
162 2023-02-23           0.690463
229 2023-05-31           0.606584
25  2022-08-08           0.591525
264 2023-07-21           0.578378
289 2023-08-25           0.550450
228 2023-05-30           0.549040
```

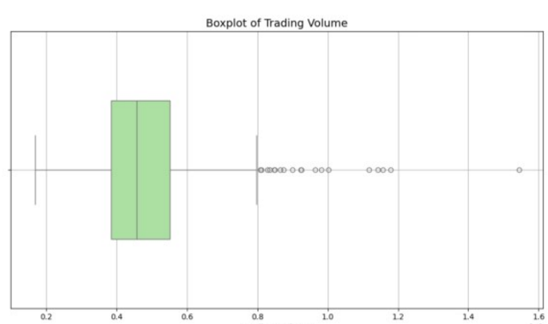
Task 3: Data Visualization:

1. Create a histogram of daily returns.,



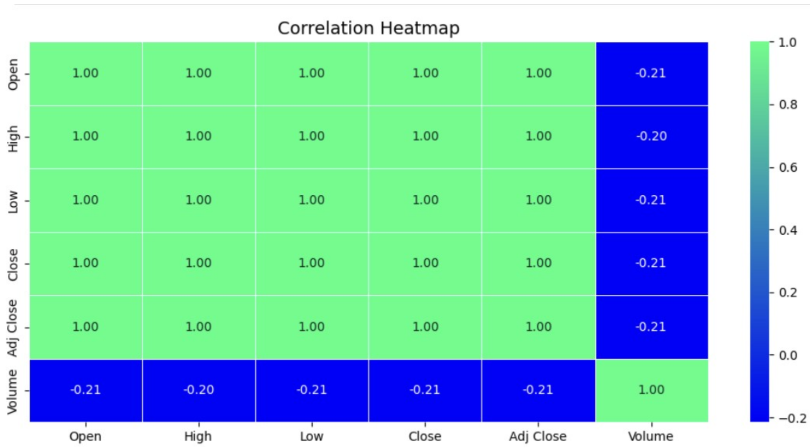
The histogram shows NVDA's daily returns, with most price changes being small, as indicated by the largest bar near zero. Fewer days had larger price swings, which are represented by smaller bars farther from the center. This is typical in stock markets where small changes are common, but occasional larger moves occur.

2. Generate a boxplot of the trading volume.,



This boxplot shows the distribution of NVDA's trading volume, with most values concentrated between 0.2 and 0.8. The whiskers indicate the range, and outliers are shown as individual points outside the whiskers, reflecting unusual trading days.

3. Display a correlation heatmap of all numerical features.,



This heatmap shows the correlation between stock features. Most features like 'Open', 'High', 'Low', 'Close', and 'Adj Close' are positively correlated, while 'Volume' has a weak negative correlation with the others. This indicates that price movements and volume tend to have an inverse relationship.

Part 2: Machine Learning:

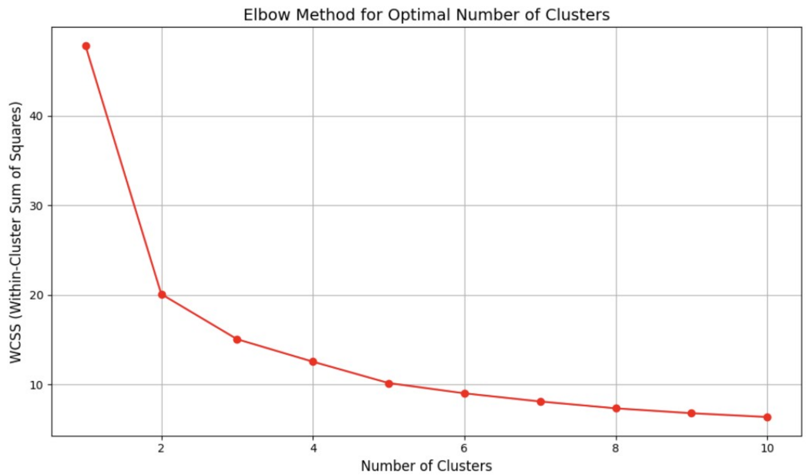
Task 1: Clustering with KMeans:

1. Select relevant features for clustering.,

I selected Daily Returns, Normalized Volume and Adjusted Closed price as mentioned in the question., after selecting this, we get.,

	Normalized Daily Return	Normalized Volume	Normalized Adj Close
0	0.291049	0.297735	0.026447
1	0.379388	0.351360	0.029989
2	0.323322	0.262455	0.031322
3	0.431027	0.236173	0.037169
4	0.287381	0.218055	0.037009

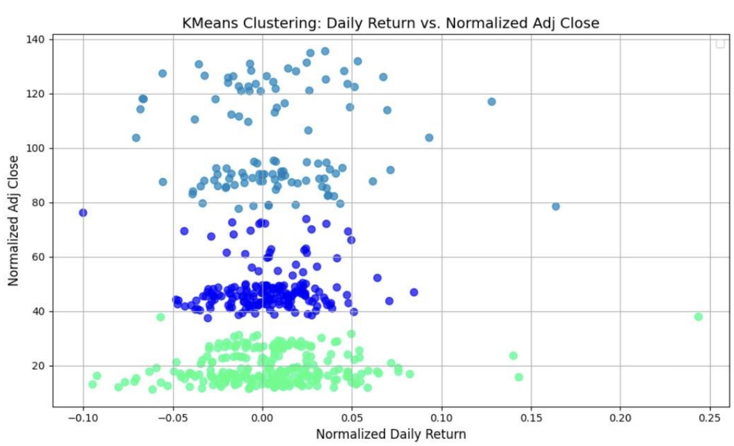
2. Here, we must determine the optimal number of clusters using the elbow method and once the code has been implemented, we get the plotted graph as shown in the picture.,



This is the Elbow Method plot used to determine the optimal number of clusters for KMeans clustering. The plot shows how the WCSS (Within-Cluster Sum of Squares) decreases as the number of clusters increases, and the "elbow" point, where the rate of decrease slows, suggests the best number of clusters, which here appears to be 3.

3. Apply Kmeans clustering and visualized the resulting clusters using a scatter plot as explained in the picture below.,

This scatter plot shows the results of KMeans clustering, where the data points are grouped into three clusters based on their normalized daily returns and adjusted closing prices, represented by different colors.



4. Interpret the clusters and describe the potential insights.,

a) Green cluster points may represent low-volatile stocks with moderate daily returns, they have low adjusted closing prices, and they appear steady.

b) The prices of stocks with a dark blue cluster are modest; they may be mid-cap stocks with slightly greater daily returns.

c) The light blue cluster's equities have the highest adjusted closing prices and the most spared daily return, which indicates that they may be more volatile and have the fastest- growing stocks.

d) Lightblue clusters have more risk but higher returns than green clusters, which appear to have lower risk.

e) A scatter plot demonstrates how well-diversified and trend-capturing the portfolio is.

Task 2: Other Machine Learning Methods:

1. Stock Price Prediction:

I used Linear Regression for prediction and used Ridge Regression Model as the bonus ML methods.

Linear Regression:

- After training and evaluation, the data, I got a RMSE score of 0.0168 and R squared score of 1.000.
- This shows that the model's predictions are extremely close to the actual data, with very little error, as seen by the RMSE of 0.0168. With an R-squared value of 1.000, the model appears to be a perfect match, explaining 100% of the variance in the data.

```
Linear Regression RMSE: 0.0168
Linear Regression R² Score: 1.0000
```

Ridge Regression Model:

- Following training and data evaluation, I received a R squared score of 0.997 and an RMSE score of 0.6060.
  • With an average error of roughly 0.6060, the model's predictions are typically accurate, according to the RMSE score of 0.6060. A very good fit is indicated by the model's R-squared score of 0.997, which indicates that it explains 99.7% of the variation in the data.

```
Ridge Regression Model RMSE: 0.6060
R² Score: 0.9997
```

Even though the first model has a perfect R-squared (1.000) and a very small error (RMSE of 0.0168), the second model fits the data better. The second model's R-squared of 0.997 indicates that it explains 99.7% of the data, which is excellent, although having a slightly greater error (RMSE of 0.6060). The perfect R-squared of the first model may indicate overfitting, which means it may work well with existing data but not with fresh data. Consequently, the second model is more dependable overall since it effectively balances error and data explanation.

2. Trends Classification:

For Trends Classification, I used Support Vector Machine as the classification model.,

- After evaluating and training the data, I got an accuracy of 0.5333 and rest is explained in the picture below.

```
Accuracy: 0.5333
Confusion Matrix:
[[ 5 44]
 [ 5 51]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.10      0.17        49
           1       0.54      0.91      0.68        56

    accuracy                           0.53       105
   macro avg       0.52      0.51      0.42       105
weighted avg       0.52      0.53      0.44       105
```