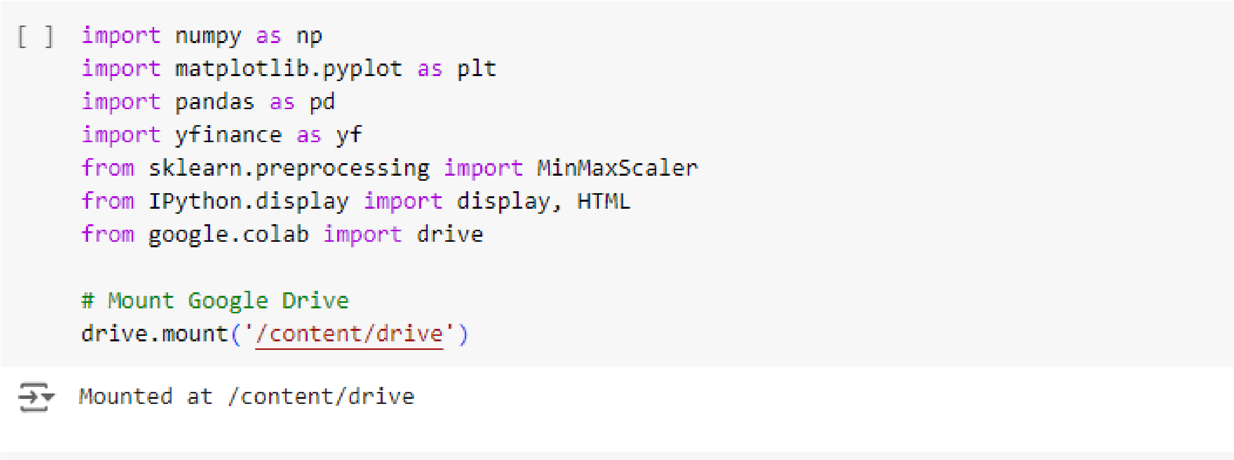
Task Report Cos30018 Option B

B.2: Data Processing 1

Name: Le Bao Nguyen

Student Id: 104169837

I. Importing libraries:



```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import yfinance as yf
    from sklearn.preprocessing import MinMaxScaler
    from IPython.display import display, HTML
    from google.colab import drive

    # Mount Google Drive
    drive.mount('/content/drive')
```

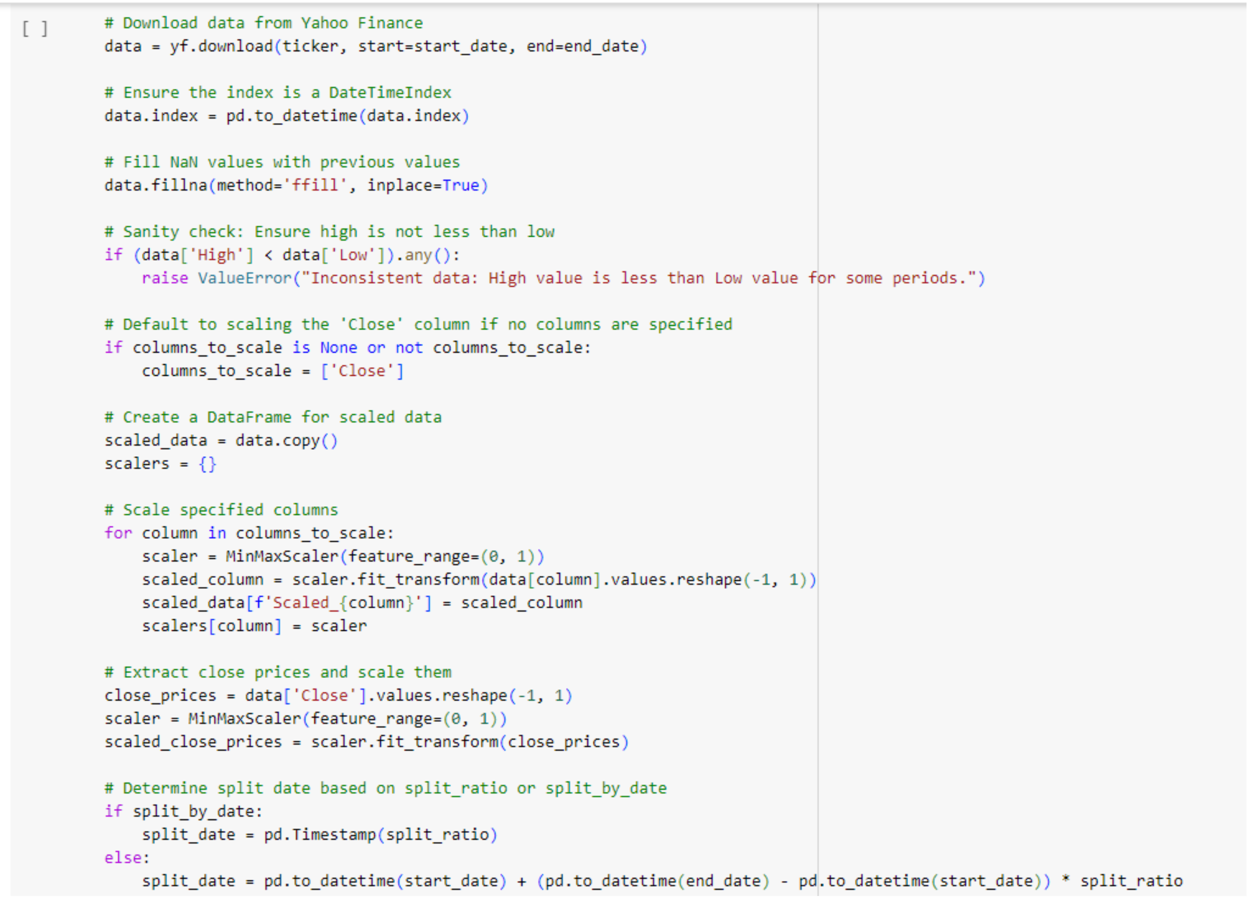
```
Mounted at /content/drive
```

Figure 1: Importing libraries to run the code.

- The script import several libraries:
- "numpy": For effective mathematical operations, especially when working with arrays, it is imported as "np".
- "matplotlib.pyplot": In order to create plots and data visualizations, it is imported as "plt".
- "pandas": It is imported as "pd" to provide strong data analysis and modification features.
- "yfinance": Importing it as "yf" makes it simple to download Yahoo Finance stock market data.
- "MinMaxScaler" from "sklearn.preprocessing": In order to ensure that all values are scaled between 0 and 1, it is used to normalize the data, which is essential for neural network performance.

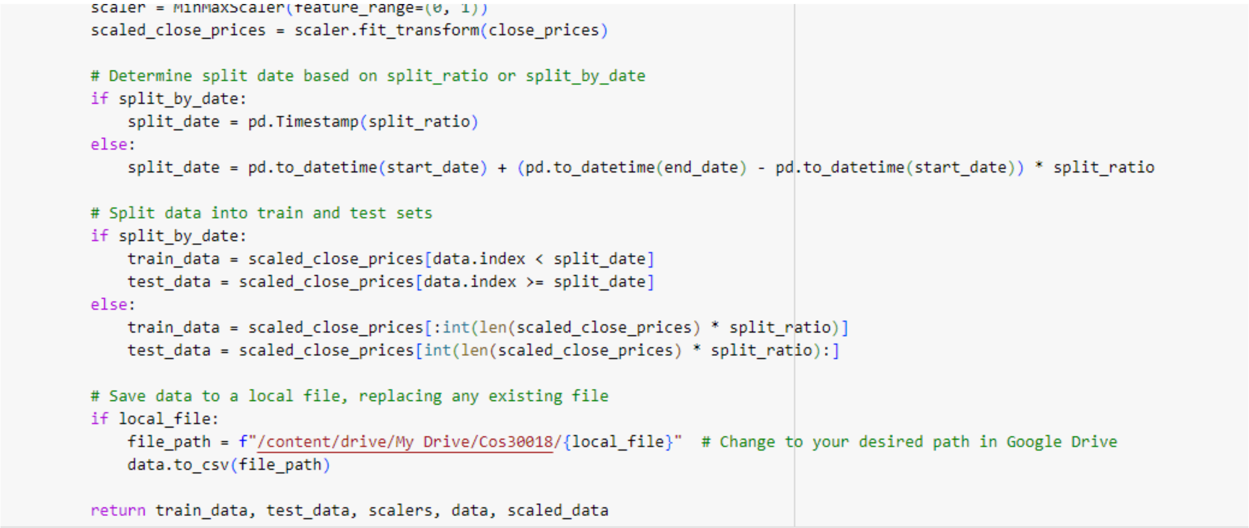
- "IPython.display": It's used to present HTML content and dataframes in Jupyter notebooks in an organized manner.
- "Drive" from "google.collab": Import drive from google collab to connect and save file to google drive.

II. Data loading and processing:



```
# Download data from Yahoo Finance
data = yf.download(ticker, start=start_date, end=end_date)

# Ensure the index is a DateTimeIndex
data.index = pd.to_datetime(data.index)

# Fill NaN values with previous values
data.fillna(method='ffill', inplace=True)

# Sanity check: Ensure high is not less than low
if (data['High'] < data['Low']).any():
    raise ValueError("Inconsistent data: High value is less than Low value for some periods.")

# Default to scaling the 'Close' column if no columns are specified
if columns_to_scale is None or not columns_to_scale:
    columns_to_scale = ['Close']

# Create a DataFrame for scaled data
scaled_data = data.copy()
scalers = {}

# Scale specified columns
for column in columns_to_scale:
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_column = scaler.fit_transform(data[column].values.reshape(-1, 1))
    scaled_data[f'Scaled_{column}'] = scaled_column
    scalers[column] = scaler

# Extract close prices and scale them
close_prices = data['Close'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_close_prices = scaler.fit_transform(close_prices)

# Determine split date based on split_ratio or split_by_date
if split_by_date:
    split_date = pd.Timestamp(split_ratio)
else:
    split_date = pd.to_datetime(start_date) + (pd.to_datetime(end_date) - pd.to_datetime(start_date)) * split_ratio
```

Figure 2: Loading and processing data (1).

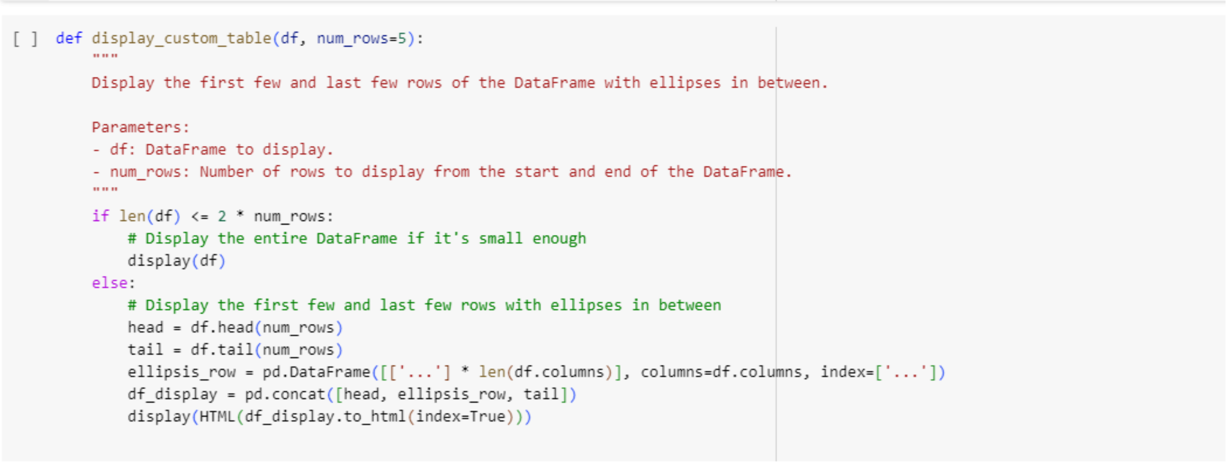


```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_close_prices = scaler.fit_transform(close_prices)

# Determine split date based on split_ratio or split_by_date
if split_by_date:
    split_date = pd.Timestamp(split_ratio)
else:
    split_date = pd.to_datetime(start_date) + (pd.to_datetime(end_date) - pd.to_datetime(start_date)) * split_ratio

# Split data into train and test sets
if split_by_date:
    train_data = scaled_close_prices[data.index < split_date]
    test_data = scaled_close_prices[data.index >= split_date]
else:
    train_data = scaled_close_prices[:int(len(scaled_close_prices) * split_ratio)]
    test_data = scaled_close_prices[int(len(scaled_close_prices) * split_ratio):]

# Save data to a local file, replacing any existing file
if local_file:
    file_path = f"/content/drive/My Drive/Cos30018/{local_file}"  # Change to your desired path in Google Drive
    data.to_csv(file_path)

return train_data, test_data, scalers, data, scaled_data
```

Figure 3: Loading and processing data (2).

- For the specified ticker and date range, the "load_and_process_data" function is intended to retrieve and preprocess stock market data. First, use "yfinance" to download the data from Yahoo Finance. The downloaded data is saved to a CSV file if a "local_file" path is provided.
- Any missing values (NaNs) in the dataset are filled using the forward-fill approach, which replaces the last valid observation for missing values, in order to maintain data integrity.
- The "Close" prices are then extracted by the function, which then reshapes them into a column vector and uses "MinMaxScaler" to normalize them to a range of 0 to 1. The neural network's performance depends on this scaling. Also, it can scale to a specified column instead of scaling all. If there is no column to specified column is scaled, it auto scales the close column.
- A specified ratio or split date are then used for splitting the data into training and testing sets. Splitting data based on a certain date is applied if "split_by_date" is set to "True"; if not, the ratio is calculated.

III. Displaying data in a custom table:

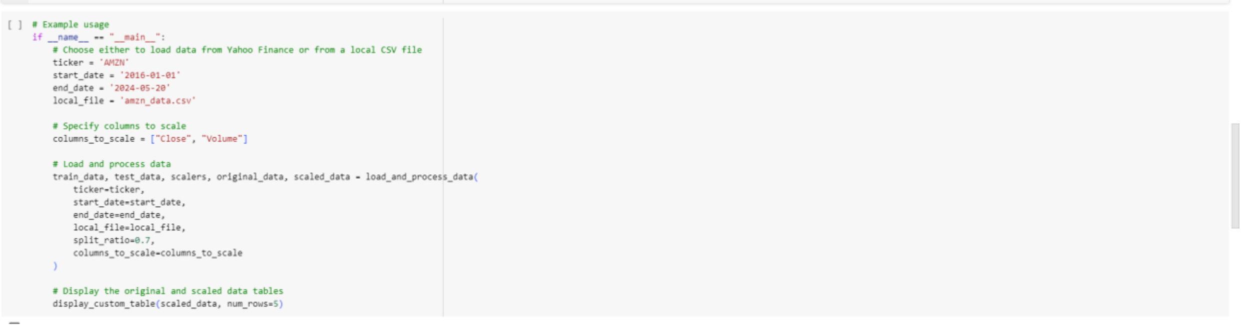


```
[ ] def display_custom_table(df, num_rows=5):
        """
        Display the first few and last few rows of the DataFrame with ellipses in between.

        Parameters:
        - df: DataFrame to display.
        - num_rows: Number of rows to display from the start and end of the DataFrame.
        """
        if len(df) <= 2 * num_rows:
            # Display the entire DataFrame if it's small enough
            display(df)
        else:
            # Display the first few and last few rows with ellipses in between
            head = df.head(num_rows)
            tail = df.tail(num_rows)
            ellipsis_row = pd.DataFrame([['...'] * len(df.columns)], columns=df.columns, index=['...'])
            df_display = pd.concat([head, ellipsis_row, tail])
            display(HTML(df_display.to_html(index=True)))
```

Figure 4: Displaying the data from csv file.

- The "display_custom_table" function aims to display a DataFrame's contents in an organized manner. It shows only the first and final few rows, with ellipses splitting them to show that the middle part of the DataFrame is hidden. This method is especially helpful for handling big datasets since it provides an overview of the start and finish of the data without overloaded the user with details.
- The function decides whether to display the DataFrame in entirety or in sections, with an ellipsis row placed between the head and tail, according on how long it is. The HTML rendering capabilities of "IPython.display'" are then used to display the resulting DataFrame.

IV. Main script run:



```
[ ] # Example usage
    if __name__ == "__main__":
        # Choose either to load data from Yahoo Finance or from a local CSV file
        ticker = 'AMZN'
        start_date = '2016-01-01'
        end_date = '2024-05-20'
        local_file = 'amzn_data.csv'

        # Specify columns to scale
        columns_to_scale = ["Close", "Volume"]

        # Load and process data
        train_data, test_data, scalers, original_data, scaled_data = load_and_process_data(
            ticker=ticker,
            start_date=start_date,
            end_date=end_date,
            local_file=local_file,
            split_ratio=0.7,
            columns_to_scale=columns_to_scale
        )

        # Display the original and scaled data tables
        display_custom_table(scaled_data, num_rows=5)
```

Figure 5: The script to run the code and change the date.

- The process of predicting stock prices is managed by the main script. The stock ticker (AMZN), the data collecting date range, and the location to a local CSV file for data storage are set first.
- We can add specify columns to scale the value.
- To download and preprocess the stock data and split it into training and testing sets, the "load_and_process_data" function is invoked. The original stock data is then shown succinctly using the "display_custom_table" function

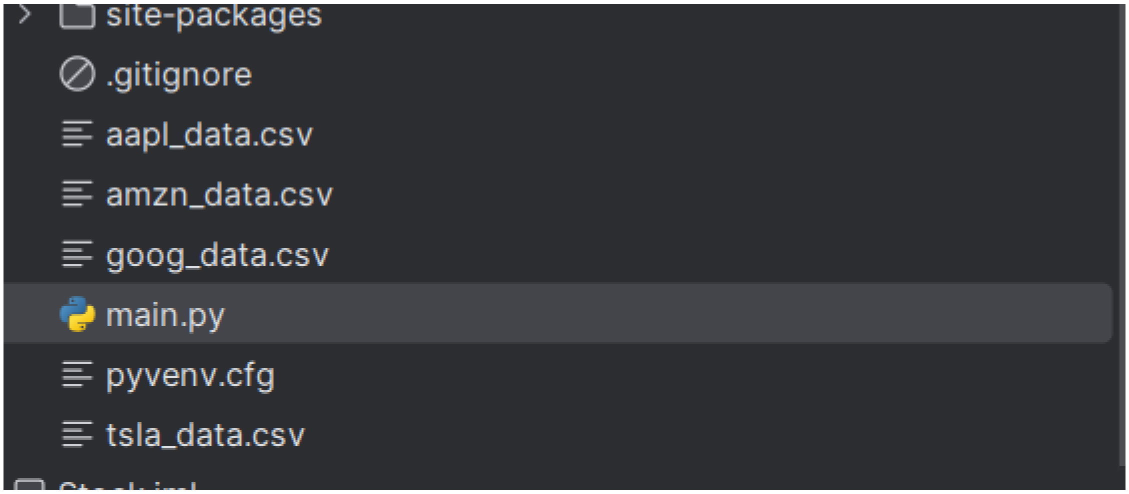
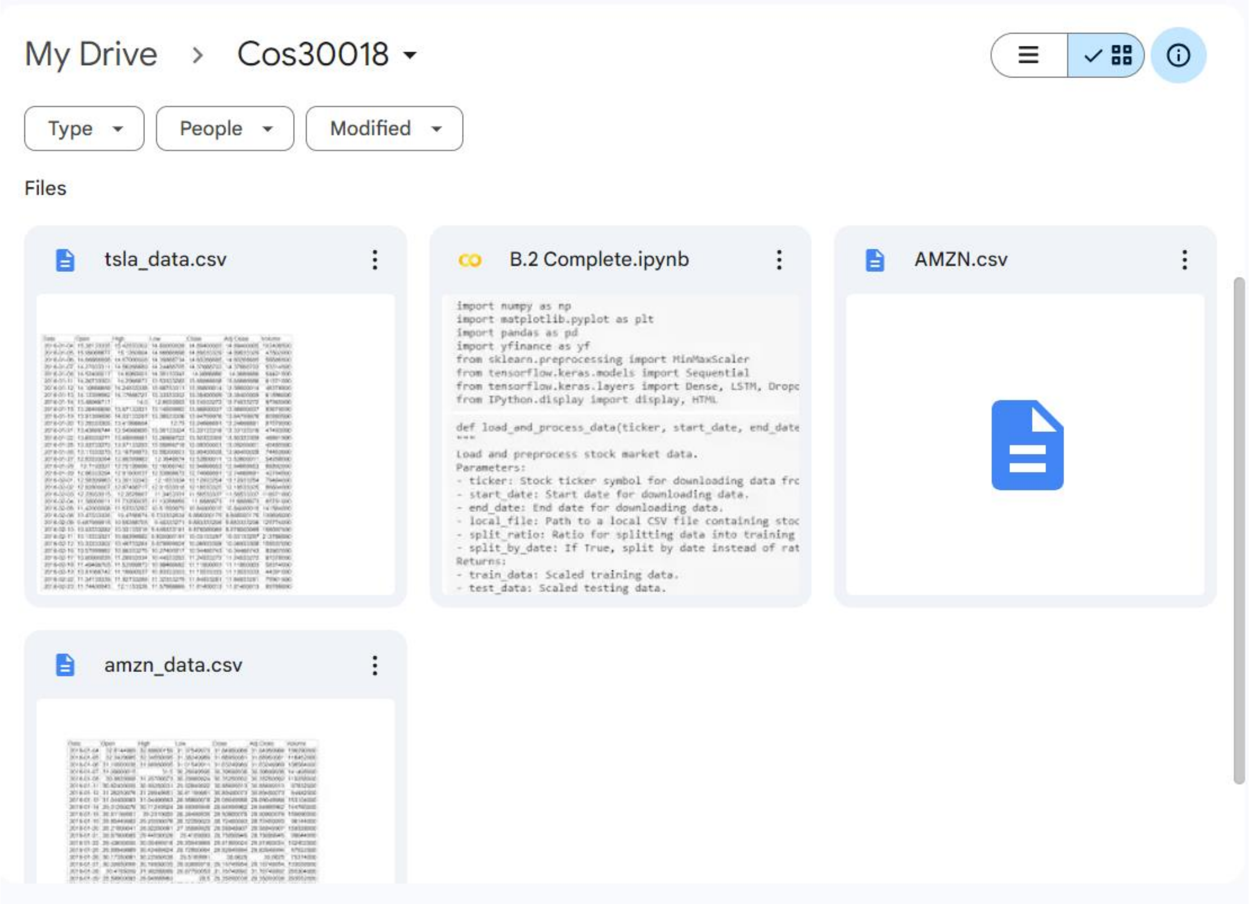
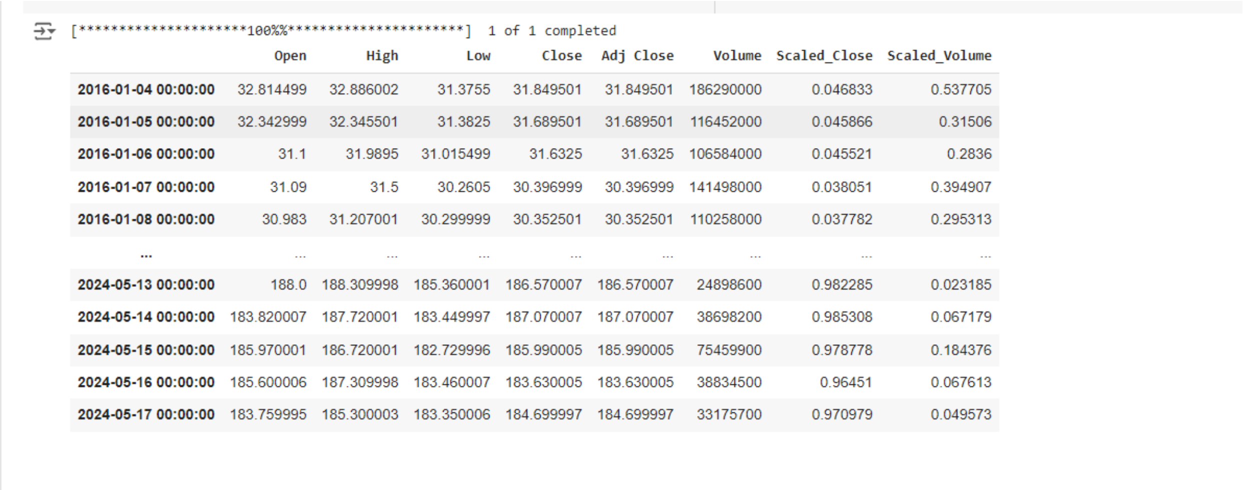


Figure 6: The csv file is loading and saving after running the code (if file_path = local_file).



```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropc
from IPython.display import display, HTML

def load_and_process_data(ticker, start_date, end_date
"""
Load and preprocess stock market data.
Parameters:
- ticker: Stock ticker symbol for downloading data frc
- start_date: Start date for downloading data.
- end_date: End date for downloading data.
- local_file: Path to a local CSV file containing stoc
- split_ratio: Ratio for splitting data into training
- split_by_date: If True, split by date instead of rat
Returns:
- train_data: Scaled training data.
- test_data: Scaled testing data.
```

Figure 7: The csv file is loading and saving after running the code (if file_path = f"/content/drive/My Drive/Cos30018/{local_file}").



```
[*********************100%%**********************]  1 of 1 completed
```

	Open	High	Low	Close	Adj Close	Volume	Scaled_Close	Scaled_Volume
2016-01-04 00:00:00	32.814499	32.886002	31.3755	31.849501	31.849501	186290000	0.046833	0.537705
2016-01-05 00:00:00	32.342999	32.345501	31.3825	31.689501	31.689501	116452000	0.045866	0.31506
2016-01-06 00:00:00	31.1	31.9895	31.015499	31.6325	31.6325	106584000	0.045521	0.2836
2016-01-07 00:00:00	31.09	31.5	30.2605	30.396999	30.396999	141498000	0.038051	0.394907
2016-01-08 00:00:00	30.983	31.207001	30.299999	30.352501	30.352501	110258000	0.037782	0.295313
...	...	...	...	...	...	...	...	...
2024-05-13 00:00:00	188.0	188.309998	185.360001	186.570007	186.570007	24898600	0.982285	0.023185
2024-05-14 00:00:00	183.820007	187.720001	183.449997	187.070007	187.070007	38698200	0.985308	0.067179
2024-05-15 00:00:00	185.970001	186.720001	182.729996	185.990005	185.990005	75459900	0.978778	0.184376
2024-05-16 00:00:00	185.600006	187.309998	183.460007	183.630005	183.630005	38834500	0.96451	0.067613
2024-05-17 00:00:00	183.759995	185.300003	183.350006	184.699997	184.699997	33175700	0.970979	0.049573

Figure 8: The loading table from csv file.