COS30018 - Option B - Task 2: Data processing 1

Student Name: Nguyen Duc Le Nguyen

Id: 104224493

Summary of Effort

This report details the development of a function to load and process stock market data with various features, including handling NaN values, splitting data into train/test sets, scaling features, and saving/loading data locally.

Code Breakdown and Explanation

Below is a detailed explanation of the less straightforward lines of code within the function.

```
import os
import pandas as pd
import yfinance as yf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import joblib
```

Figure 1 - Import Libraries

os: Provides a way of using operating system-dependent functionality, such as reading or writing to the filesystem.

pandas: A powerful data manipulation library.

yfinance: A library to fetch financial data from Yahoo Finance.

sklearn.model_selection.train_test_split: A utility function to split data into train and test sets.

sklearn.preprocessing.StandardScaler and MinMaxScaler: Tools for feature scaling.

joblib: A library for saving and loading Python objects.

```
def load_and_process_data(ticker, start_date, end_date, na_method='drop',
                          split_method='ratio', train_ratio=0.8, split_date=None,
                          random_state=42, scale=False, scaler_type='standard',
                          save_data=False, load_data=False, data_path='data.csv',
                          scaler_path='scaler.pkl'):
    """
```

Figure 2 - Load and Process Data

This function initializes with several parameters, allowing flexibility in data loading, processing, and saving.

```
# Load data from a local file if specified
if load_data and os.path.exists(data_path):
    df = pd.read_csv(data_path, index_col='Date', parse_dates=True)
else:
    # Download data from Yahoo Finance
    df = yf.download(ticker, start=start_date, end=end_date)
    if save_data:
        df.to_csv(data_path)
```

Figure 3 - Save data

Checks if data should be loaded from a local file. If not, it downloads the data from Yahoo Finance and saves it if required.

```
# Handle NaN values
if na_method == 'drop':
    df = df.dropna()
elif na_method == 'fill':
    df = df.fillna(method='ffill').fillna(method='bfill')
```

Figure 4 - Handle NaN

Handles NaN values by either dropping them or filling them. Forward fill (ffill) and backward fill (bfill) ensure no NaN values remain.

```
# Split the data into features and target
X = df.drop(columns=['Adj Close'])
y = df['Adj Close']
```

Figure 5 - Split Data

Splits the dataframe into features (X) and target (y). Here, 'Adj Close' is assumed to be the target variable.

```python
# Split the data into train and test sets
if split_method == 'ratio':
    train_data_len = math.ceil(len(X) * train_ratio)
    X_train = X.iloc[:train_data_len]
    y_train = y.iloc[:train_data_len]

    X_test = X.iloc[train_data_len:]
    y_test = y.iloc[train_data_len:]
    print(f"Train data length: {len(X_train)}")
    print(f"Test data length: {len(X_test)}")


elif split_method == 'date' and split_date:
    train_data = df[df.index < split_date]
    test_data = df[df.index >= split_date]
    X_train, y_train = train_data.drop(columns=['Close']), train_data['Close']
    X_test, y_test = test_data.drop(columns=['Close']), test_data['Close']
    print(f"Train data from {train_data.index.min()} to {train_data.index.max()}")
    print(f"Test data from {test_data.index.min()} to {test_data.index.max()}")
else:
    raise ValueError("Invalid split method or missing split date.")
```

Figure 6 - Split Methods

Depending on the split_method, the data is split either to a specified ratio or by a specific date.

```python
# Scale the feature columns if specified
scaler = None
if scale:
    if scaler_type == 'standard':
        scaler = StandardScaler()
    elif scaler_type == 'minmax':
        scaler = MinMaxScaler()
    else:
        raise ValueError("Invalid scaler type.")

    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Save the scaler if specified
    if save_data:
        joblib.dump(scaler, scaler_path)
```

Figure 7 - Scaler

If scaling is requested, the function applies either StandardScaler or MinMaxScaler to the feature columns. It also saves the scaler if specified.

```
return X_train, X_test, y_train, y_test, scaler
```

Figure 8 - Data Returned

Returns the processed data splits and the scaler (if applied).

Output

AAPL_data.csv stock_pre • AAPL_data.csv\ data

```
Date,Open,High,Low,Close,Adj Close,Volume
2018-12-17,41.36249923706055,42.087501525878906,40.682498931884766,40.98500061035156,39.2344970703125,177151600
2018-12-18,41.345001220703125,41.88249969482422,41.09749984741211,41.51750183105469,39.7442512512207,135366000
2018-12-19,41.5,41.86249923706055,39.772499084472656,40.22249984741211,38.50457000732422,196189200
2018-12-20,40.099998474121094,40.52750015258789,38.82500076293945,39.20750045776367,37.53292465209961,259092000
2018-12-21,39.21500015258789,39.540000915527344,37.40750122070312,37.682498931884766,36.07305908203125,382978400
2018-12-24,37.037498474121094,37.88750076293945,36.647499084472656,36.70750045776367,35.13969802856445,148676800
2018-12-26,37.07500076293945,39.307498931884766,36.68000030517578,39.29249954223633,37.61428451538086,234330000
2018-12-27,38.959999084472656,39.192501068115234,37.51750183105469,39.037498474121094,37.37018585205078,212468400
2018-12-28,39.375,39.630001068115234,38.63750076293945,39.057498931884766,37.389320373535156,169165600
2018-12-31,39.63249969482422,39.84000015258789,39.119998931884766,39.435001373291016,37.750701904296875,140014000
2019-01-02,38.72249984741211,39.712501525878906,38.557498931884766,39.47999954223633,37.79377365112305,148158800
```

Figure 9 - Stock Data in CSV

```
PS D:\cos30018\stock_pre\B2> python B2.py
[**********************100%%**********************]  1 of 1 completed
       Open      High       Low     Close    Volume
0  0.036152  0.039739  0.039338  0.044690  0.253276
1  0.107885  0.110776  0.100741  0.101868  0.477122
2  0.686468  0.695570  0.690340  0.702822  0.250719
3  0.668508  0.662357  0.639176  0.632572  0.160387
4  0.271980  0.272091  0.273123  0.270850  0.304350
          Open      High       Low     Close    Volume
1009  0.726956  0.722471  0.715299  0.709404  0.079408
1010  0.797377  0.795882  0.798145  0.788758  0.061287
1011  0.909582  0.906673  0.911988  0.912895  0.083026
1012  0.696590  0.694528  0.692020  0.699377  0.134890
1013  0.875822  0.884919  0.883605  0.892103  0.065016
PS D:\cos30018\stock_pre\B2>
```

Figure 10 - Scaled Data

```
[**********************100%%**********************]  1 of 1 completed
Train data from 2018-12-17 00:00:00 to 2021-12-31 00:00:00
Test data from 2022-01-03 00:00:00 to 2024-03-01 00:00:00
```

Figure 11 - Split Data by Date

```
[**********************100%%**********************]  1 of 1 completed
Train data length: 1048
Test data length: 262
```

Figure 12 - Split Data by Ratio (0.8)

This report covered the key lines of the load_and_process_data function, explaining each part to ensure clarity. Further inquiries about any specific line of code are welcome.