Stock Price Prediction System

(Option 3)

Weekly Report (Tasks C.2 -Data Processing 1)

COS30018 Intelligent Systems

Class: Thursday 2:30pm to 4:30pm (2 hours)

Name: Suyogya Raj Joshi
Student ID: 105098233

Contents

Check List of given weekly tasks:

No.	Weekly Tasks	Progress
1.	Writing a function that load and processes a dataset with multiple features	Complete
2.	Summary of Implemented function and features selected for processing	Complete

Initial Testing with the given code base.

In the given starter code base, there were a lot of comments with instructions on various ways of implementing the techniques required for the second assessment. The first instruction was to fetch the data for the stock using yahoo with **web.Datareader** . However, this method of fetching the data had some reliability issue with the yahoo API. Hence, decision to fetch the data using yahoo finance was made.

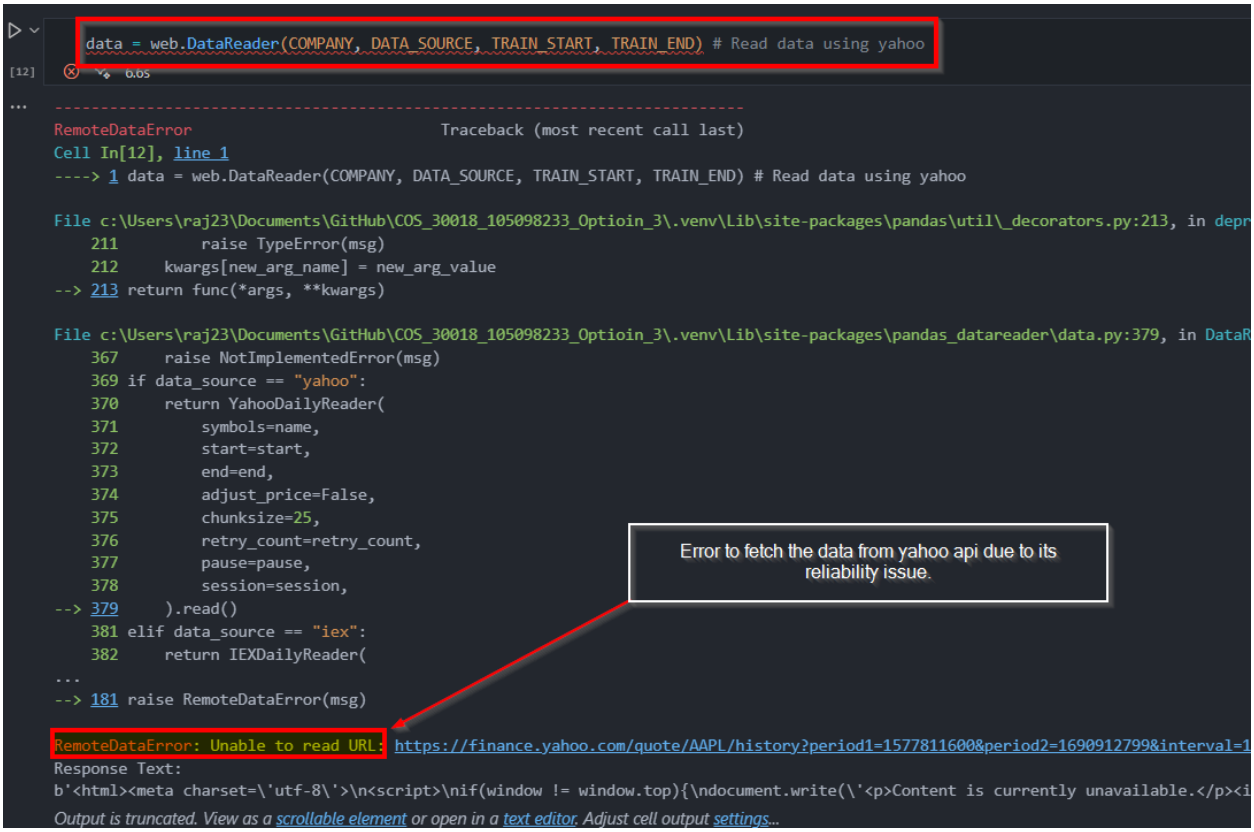


Fig: Reliability issue with the yahoo API.

Defining a function for loading and processing the data.

Before defining the function, Dataset was loaded from yahoo finance to see the different attributes of the dataset which will help to define function and how to process the dataset.

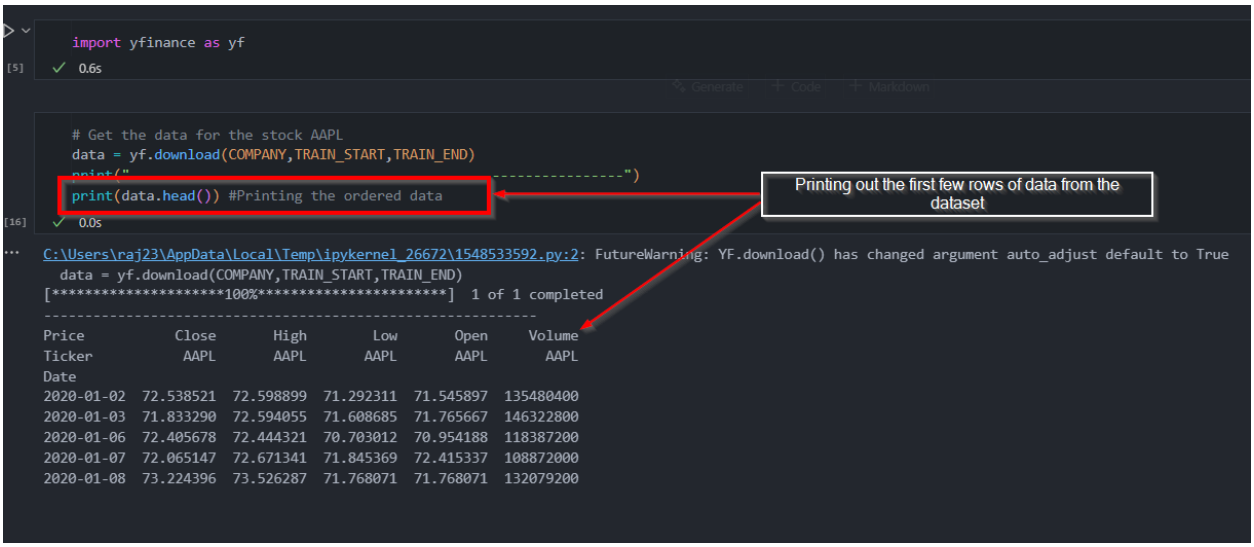


Fig: Printing out the data using head () function from pandas' library.

Here dataset had 6 attributes:

- Close
- High
- Low
- Open
- Volume

Now using this information, function was defined which loads and processes the dataset:

```python
# Defining a function that loads the data
def load_data(COMPANY: str, TRAIN_START: str, TRAIN_END: str, test_size: float = 0.2, data_dir: str = "data"):
    # Ensuring data directory exists
    os.makedirs(data_dir, exist_ok=True)
    DATA_PATH = os.path.join(data_dir, f"{COMPANY}_{TRAIN_START}_{TRAIN_END}.csv")

    # Loading data if present, else downloading and saving
    if os.path.exists(DATA_PATH):
        data = pd.read_csv(DATA_PATH, index_col=0, parse_dates=True)
    else:
        data = yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END)
        data.to_csv(DATA_PATH)

    print("------------------------------------------------------------")
    print("Initial output with the NaN")
    print(data.head(10))  # First 10 rows
    print("------------------------------------------------------------")
    print(data.isna().sum())  # Checking NaN Or missing values.
    print("============================================================================================================")

    # Removing NaN values
    data = data.dropna()

    # Quick check
    print("------------------------------------------------------------")
    print("Final output without the NaN")
    print(data.isna().sum())  # Checking the NaN after removing it

    # Ensure columns are numeric before calculation
    for col in ["Open", "High", "Low", "Close"]:
        data[col] = pd.to_numeric(data[col], errors='coerce')

    # Add OHLC average column and set PRICE_VALUE
    data["OHLC_Avg"] = (data["Open"] + data["High"] + data["Low"] + data["Close"]) / 4
    PRICE_VALUE = "OHLC_Avg"

    # Scaling the data
    # Choosing the price column (i.e OHLC_Avg) and reshape it into 2D
    values = data[PRICE_VALUE].values.reshape(-1, 1)
    # Creating scaler
    scaler = MinMaxScaler(feature_range=(0, 1))
    # Fitting and transforming the scaler.
    scaled_data = scaler.fit_transform(values)

    # Splitting the dataset into training and testing set.
    train, test = train_test_split(scaled_data, test_size=test_size, shuffle=False)

    return data, train, test, scaler, PRICE_VALUE, scaled_data
```

✓ 0.0s

Fig: Defining the function that loads and processes data.

The function allows for the reusability of code as one can easily tune the data processing and loading by changing few variables.

For the above function, the below given input as shown in the image:

```python
TRAIN_START = '2020-01-01'     # Start date to read
TRAIN_END = '2023-08-01'       # End date to read
COMPANY = "AAPL" #Ticker for apple

# Calling the function to load the dataset and save the return value to specific variable
data, train, test, scaler, PRICE_VALUE, scaled_data = load_data(
    COMPANY, TRAIN_START, TRAIN_END, test_size=0.2
)

print("Dataset shape:", data.shape)
print("Train shape:", train.shape)
print("Test shape:", test.shape)
print("Using PRICE_VALUE:", PRICE_VALUE)
```

✓ 0.0s

Summary of the function along with output.

The function accepts five parameters: ticker, start_date, end_date, test_size, and folder name. Its purpose is to collect (or load) historical prices, clean and convert them.

```
# Defining a function that loads the data
def load_data(COMPANY: str, TRAIN_START: str, TRAIN_END: str, test_size: float = 0.2, data_dir: str = "data"):
```

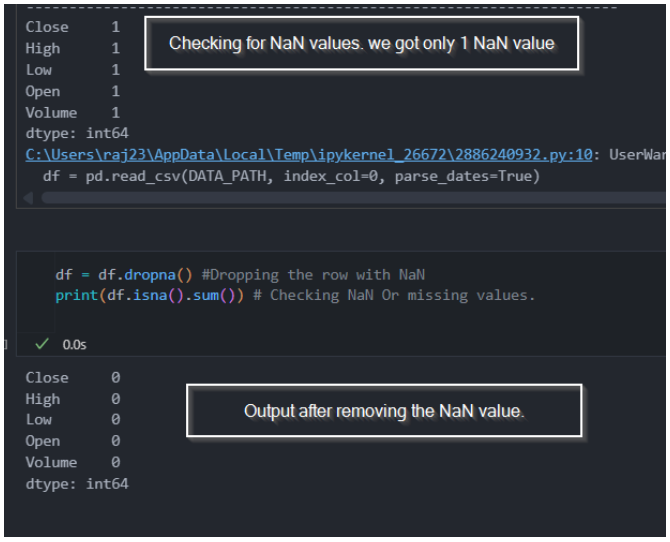
The OS module ensures the setup of a data directory. A DATA_PATH variable stores this location, ensuring that datasets are properly stored and reused in future.

```
# Ensuring data directory exists
os.makedirs(data_dir, exist_ok=True)
DATA_PATH = os.path.join(data_dir, f"{COMPANY}_{TRAIN_START}_{TRAIN_END}.csv")
```

After this process, the function had an if else loop which checked If the CSV already exists, it is loaded from disc for faster results. Otherwise, historical price data is collected from Yahoo Finance, saved to CSV at DATA_PATH, and then imported into a DataFrame named data.

```
# Loading data if present, else downloading and saving
if os.path.exists(DATA_PATH):
    data = pd.read_csv(DATA_PATH, index_col=0, parse_dates=True)
else:
    data = yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END)
    data.to_csv(DATA_PATH)
```

Missing values are determined using **data.isna().sum().** Because only one row was missing in this experiment, the technique row dropping **(data = data.dropna())** was used. Dropping avoids adding false patterns into the target series.



Instead of capturing a single attribute like the Close, the function calculates the day's entire movement:

$$OHLC\ \ avg = \frac{Open + High + Low + Close}{4}$$

The target selection variable is set, and this new column is added to the data: "OHLC_Avg" is PRICE_VALUE.

```
# Ensure columns are numeric before calculation
for col in ["Open", "High", "Low", "Close"]:
    data[col] = pd.to_numeric(data[col], errors='coerce')

# Add OHLC average column and set PRICE_VALUE
data["OHLC_Avg"] = (data["Open"] + data["High"] + data["Low"] + data["Close"]) / 4
PRICE_VALUE = "OHLC_Avg"
```

The selected series is extracted as **data[PRICE_VALUE]** and reshaped into a 2D array using**. reshape(-1, 1).** This shape is required by scikit-learn transformers and numerous deep neural network layers.

A **MinMaxScaler** is applied to the target series and transformed into the [0, 1] range, resulting in scaled_data. Scaling maintains training and usually improves consistency in LSTM-style models.

```
# Scaling the data
# Choosing the price column (i.e OHLC_Avg) and reshape it into 2D
values = data[PRICE_VALUE].values.reshape(-1, 1)
# Creating scaler
scaler = MinMaxScaler(feature_range=(0, 1))
# Fitting and transforming the scaler.
scaled_data = scaler.fit_transform(values)
```

Using given test_size (0.2), the series is divided sequentially into train (first 80%) and test (last 20%). This ensures time order while preventing forecasting bias.

```
# Splitting the dataset into training and testing set.
train, test = train_test_split(scaled_data, test_size=test_size, shuffle=False)

return data, train, test, scaler, PRICE_VALUE, scaled_data
```

Here the final output looks like this:

Observed output

- Dataset shape: (901, 6)
- Train shape: (720, 1)
- Test shape: (181, 1)
- Using PRICE_VALUE: OHLC_Avg