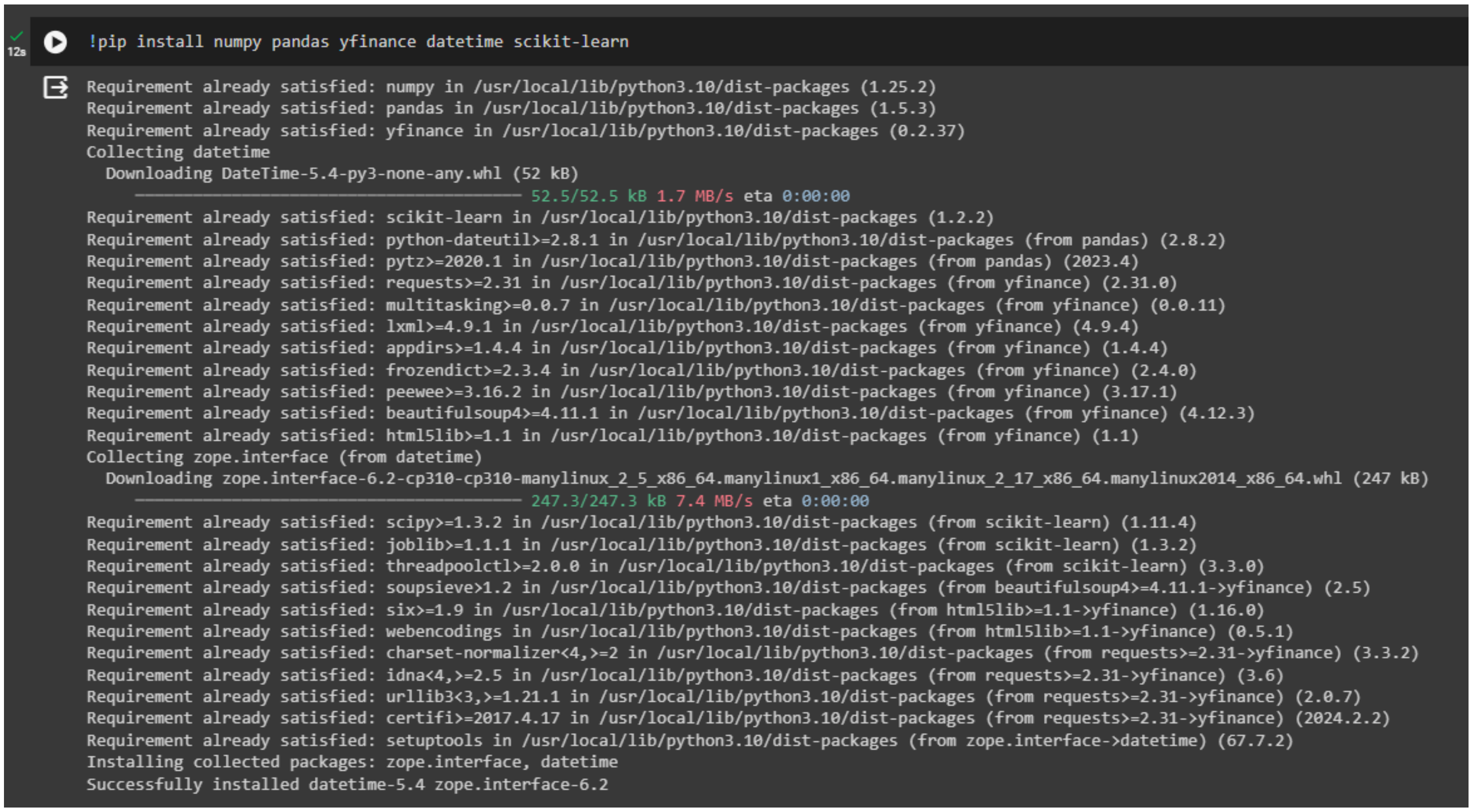
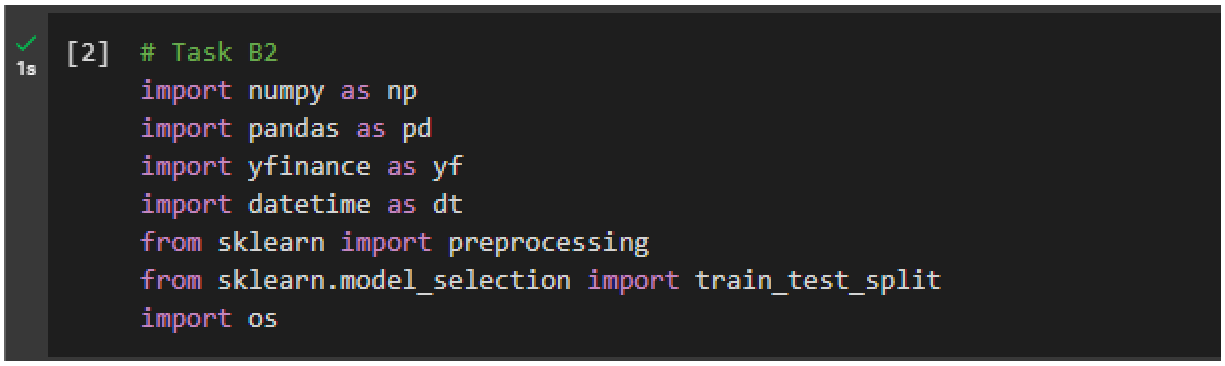


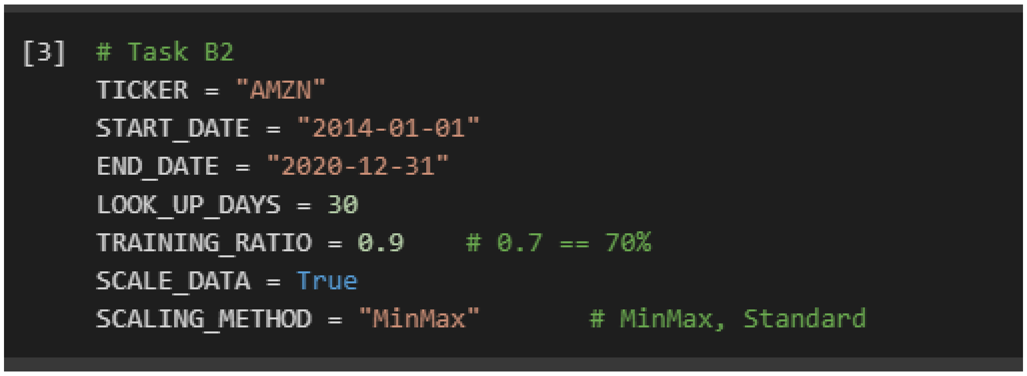
```
!pip install numpy pandas yfinance datetime scikit-learn

Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.37)
Collecting datetime
  Downloading DateTime-5.4-py3-none-any.whl (52 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.5/52.5 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.9.4)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.4.0)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Collecting zope.interface (from datetime)
  Downloading zope.interface-6.2-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (247 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.3/247.3 kB 7.4 MB/s eta 0:00:00
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.3.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2024.2.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from zope.interface->datetime) (67.7.2)
Installing collected packages: zope.interface, datetime
Successfully installed datetime-5.4 zope.interface-6.2
```



```
[2] # Task B2
    import numpy as np
    import pandas as pd
    import yfinance as yf
    import datetime as dt
    from sklearn import preprocessing
    from sklearn.model_selection import train_test_split
    import os
```

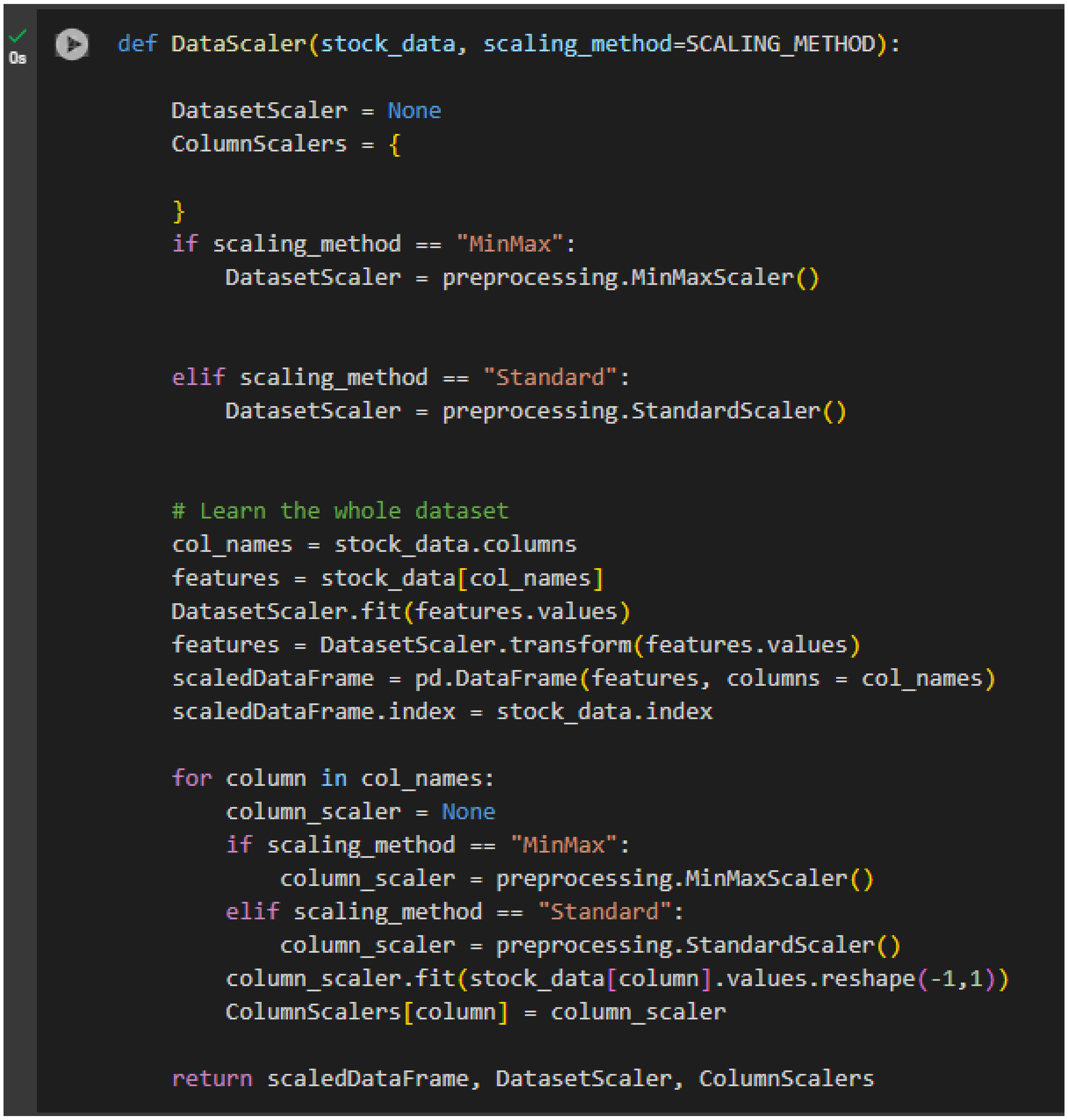
We import the necessary modules and libraries for data processing and machine learning.

Import yfinance as yf: This line imports the yfinance module, which is a Python library that provides access to financial data from Yahoo Finance. It allows us to download the stock data for a given ticker and time period.



```
[3] # Task B2
    TICKER = "AMZN"
    START_DATE = "2014-01-01"
    END_DATE = "2020-12-31"
    LOOK_UP_DAYS = 30
    TRAINING_RATIO = 0.9    # 0.7 == 70%
    SCALE_DATA = True
    SCALING_METHOD = "MinMax"       # MinMax, Standard
```

We define all necessary constants and parameters for the data loading and splitting, such as the ticker symbol, the start and end dates, the look up days, the training ratio, the scale data flag, and the scaling method.



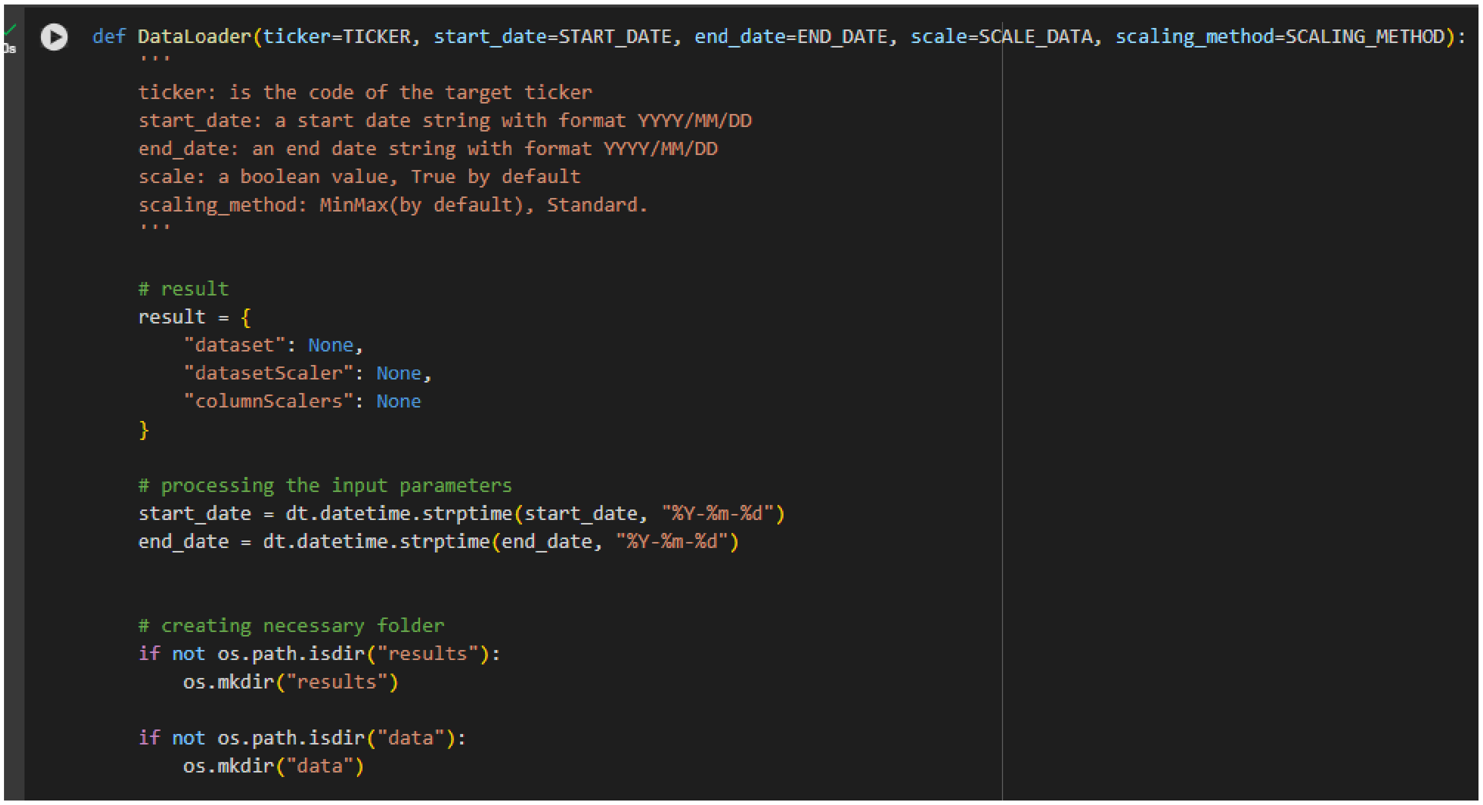
```python
def DataScaler(stock_data, scaling_method=SCALING_METHOD):

    DatasetScaler = None
    ColumnScalers = {

    }
    if scaling_method == "MinMax":
        DatasetScaler = preprocessing.MinMaxScaler()


    elif scaling_method == "Standard":
        DatasetScaler = preprocessing.StandardScaler()


    # Learn the whole dataset
    col_names = stock_data.columns
    features = stock_data[col_names]
    DatasetScaler.fit(features.values)
    features = DatasetScaler.transform(features.values)
    scaledDataFrame = pd.DataFrame(features, columns = col_names)
    scaledDataFrame.index = stock_data.index

    for column in col_names:
        column_scaler = None
        if scaling_method == "MinMax":
            column_scaler = preprocessing.MinMaxScaler()
        elif scaling_method == "Standard":
            column_scaler = preprocessing.StandardScaler()
        column_scaler.fit(stock_data[column].values.reshape(-1,1))
        ColumnScalers[column] = column_scaler

    return scaledDataFrame, DatasetScaler, ColumnScalers
```

Next, we define a function called DataScaler that takes a stock data DataFrame and a scaling method as inputs, and returns a scaled DataFrame, a dataset scaler, and a dictionary of column scalers. The scaling method can be either MinMax or Standard, and the function applies the scaling to each column of the stock data.

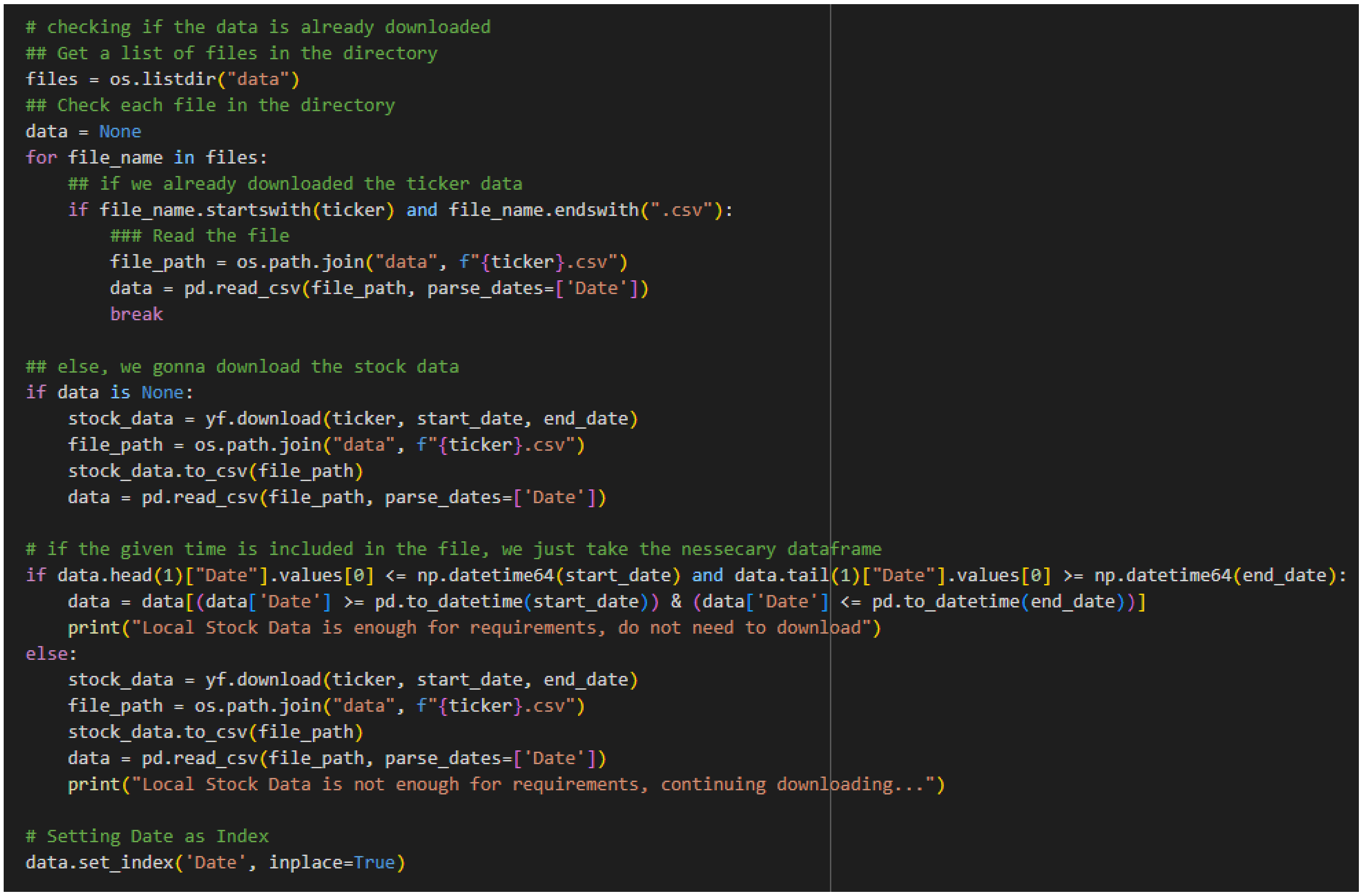
DatasetScaler = preprocessing.MinMaxScaler(): This line creates an instance of the MinMaxScaler class, which scales the data to be between 0 and 1. This is supposedly done to make the training process easier and faster, as the netwrok can learn from smaller and normalized values.

DatasetScaler = preprocessing.StandardScaler(): This line creates an instance of the StandardScaler class, which standardizes the features in a dataset. Standardization means transforming the features to have zero mean and unit variance and can improve the performance of many machine learning algorithms. The StandardScaler class can compute the mean and standard deviation of the features on a training set, and then apply the same transformation on the testing set.

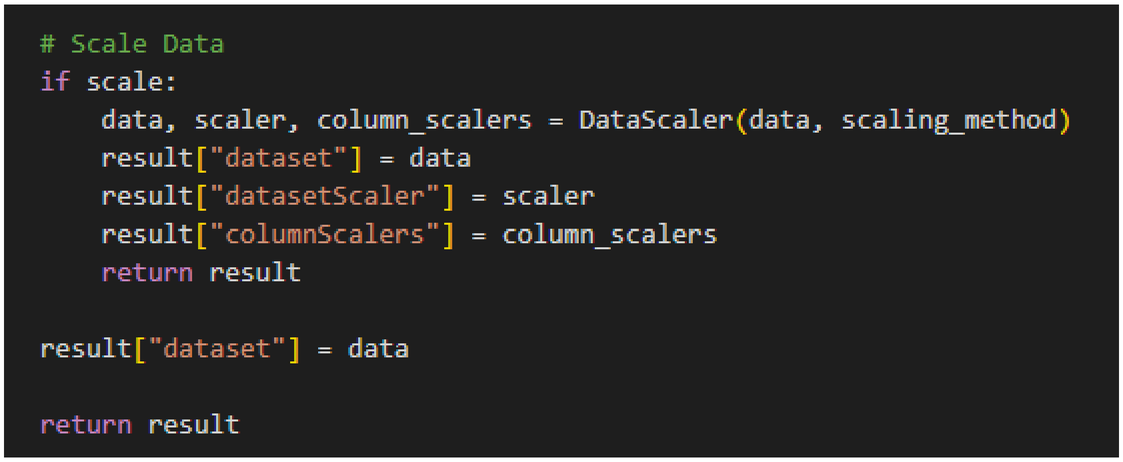
column_scaler.fit(stock_data[column].values.reshape(-1,1)): This line fits the column_scaler to the values of a specific column in the stock data. The reshape method changes the shape of the array to have one column and as many rows as needed. This is required because the scaler expects a two-dimensional input.



```
def DataLoader(ticker=TICKER, start_date=START_DATE, end_date=END_DATE, scale=SCALE_DATA, scaling_method=SCALING_METHOD):
    '''
    ticker: is the code of the target ticker
    start_date: a start date string with format YYYY/MM/DD
    end_date: an end date string with format YYYY/MM/DD
    scale: a boolean value, True by default
    scaling_method: MinMax(by default), Standard.
    '''

    # result
    result = {
        "dataset": None,
        "datasetScaler": None,
        "columnScalers": None
    }

    # processing the input parameters
    start_date = dt.datetime.strptime(start_date, "%Y-%m-%d")
    end_date = dt.datetime.strptime(end_date, "%Y-%m-%d")


    # creating necessary folder
    if not os.path.isdir("results"):
        os.mkdir("results")

    if not os.path.isdir("data"):
        os.mkdir("data")
```



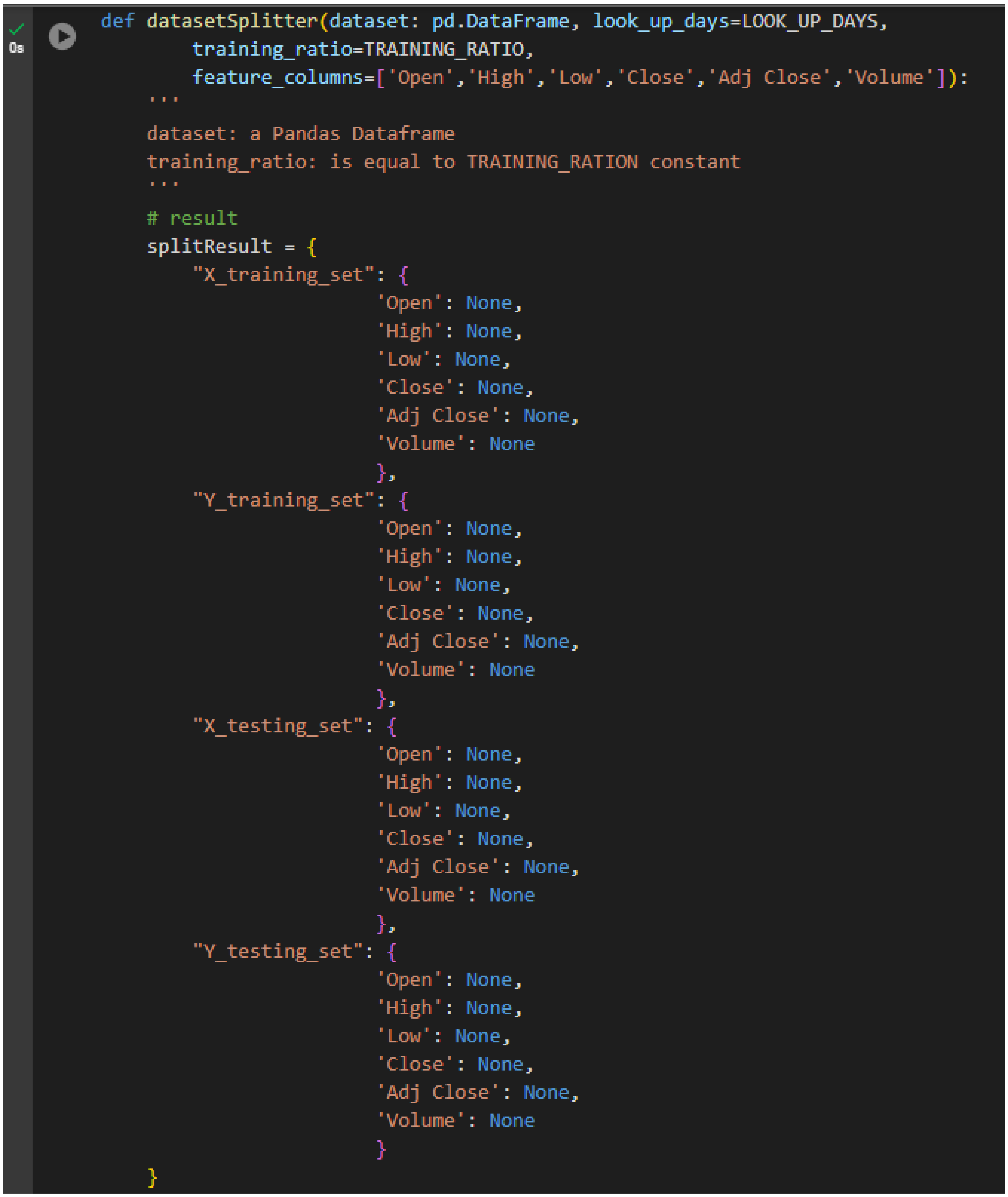
```python
# checking if the data is already downloaded
## Get a list of files in the directory
files = os.listdir("data")
## Check each file in the directory
data = None
for file_name in files:
    ## if we already downloaded the ticker data
    if file_name.startswith(ticker) and file_name.endswith(".csv"):
        ### Read the file
        file_path = os.path.join("data", f"{ticker}.csv")
        data = pd.read_csv(file_path, parse_dates=['Date'])
        break

## else, we gonna download the stock data
if data is None:
    stock_data = yf.download(ticker, start_date, end_date)
    file_path = os.path.join("data", f"{ticker}.csv")
    stock_data.to_csv(file_path)
    data = pd.read_csv(file_path, parse_dates=['Date'])

# if the given time is included in the file, we just take the nessecary dataframe
if data.head(1)["Date"].values[0] <= np.datetime64(start_date) and data.tail(1)["Date"].values[0] >= np.datetime64(end_date):
    data = data[(data['Date'] >= pd.to_datetime(start_date)) & (data['Date'] <= pd.to_datetime(end_date))]
    print("Local Stock Data is enough for requirements, do not need to download")
else:
    stock_data = yf.download(ticker, start_date, end_date)
    file_path = os.path.join("data", f"{ticker}.csv")
    stock_data.to_csv(file_path)
    data = pd.read_csv(file_path, parse_dates=['Date'])
    print("Local Stock Data is not enough for requirements, continuing downloading...")

# Setting Date as Index
data.set_index('Date', inplace=True)
```



```python
# Scale Data
if scale:
    data, scaler, column_scalers = DataScaler(data, scaling_method)
    result["dataset"] = data
    result["datasetScaler"] = scaler
    result["columnScalers"] = column_scalers
    return result

result["dataset"] = data

return result
```

We define a function called DataLoader that takes a ticker, a start date, an end date, a scale flag, and a scaling method as inputs, and returns a result dictionary that contains the dataset, the dataset scaler, and the column scalers. The function checks if the data is already downloaded in the data folder, and if not, it downloads the data from yfinance. It also filters the data according to the given dates, and sets the date as the index. If the scale flag is True, it calls the DataScaler function defined above to scale the data.

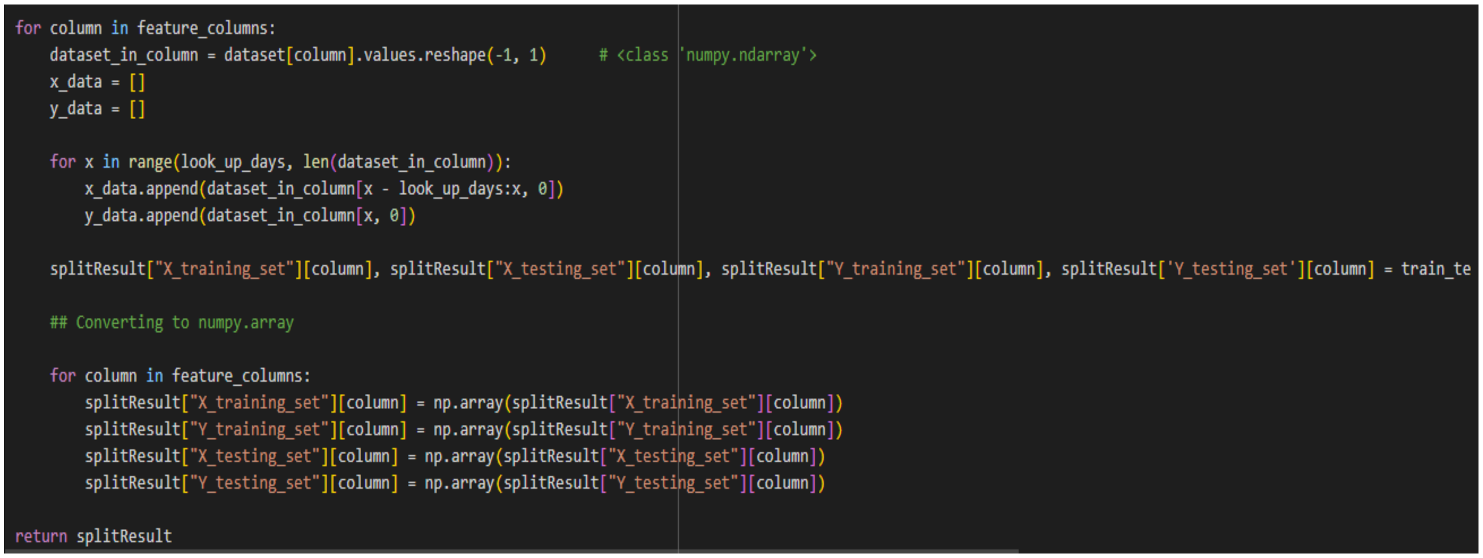
stock_data = yf.download(ticker, start_date, end_date): This line downloads the stock data for the given ticker, start date, and end date using the yfinance module. It returns a pandas DataFrame with the date, open, high, low, , close, adjclose, and volume columns.



```
def datasetSplitter(dataset: pd.DataFrame, look_up_days=LOOK_UP_DAYS,
        training_ratio=TRAINING_RATIO,
        feature_columns=['Open','High','Low','Close','Adj Close','Volume']):
    '''
    dataset: a Pandas Dataframe
    training_ratio: is equal to TRAINING_RATION constant
    '''
    # result
    splitResult = {
        "X_training_set": {
                        'Open': None,
                        'High': None,
                        'Low': None,
                        'Close': None,
                        'Adj Close': None,
                        'Volume': None
                        },
        "Y_training_set": {
                        'Open': None,
                        'High': None,
                        'Low': None,
                        'Close': None,
                        'Adj Close': None,
                        'Volume': None
                        },
        "X_testing_set": {
                        'Open': None,
                        'High': None,
                        'Low': None,
                        'Close': None,
                        'Adj Close': None,
                        'Volume': None
                        },
        "Y_testing_set": {
                        'Open': None,
                        'High': None,
                        'Low': None,
                        'Close': None,
                        'Adj Close': None,
                        'Volume': None
                        }
    }
```

We then define a function called datasetSplitter that takes a dataset DataFrame, a look up days parameter, a training ratio parameter, and a list of feature columns as inputs, and returns a splitResult dictionary that contains the X and Y training and testing sets for each feature column. The function splits the data into X and Y arrays based on the look up days, and then splits them into training and testing sets based on the training ratio. It also converts the arrays to numpy arrays.

At the beginning of this function definition, we precisely specify the splittedResult dictionary to later correctly transfer the splitted data into their assigned label.

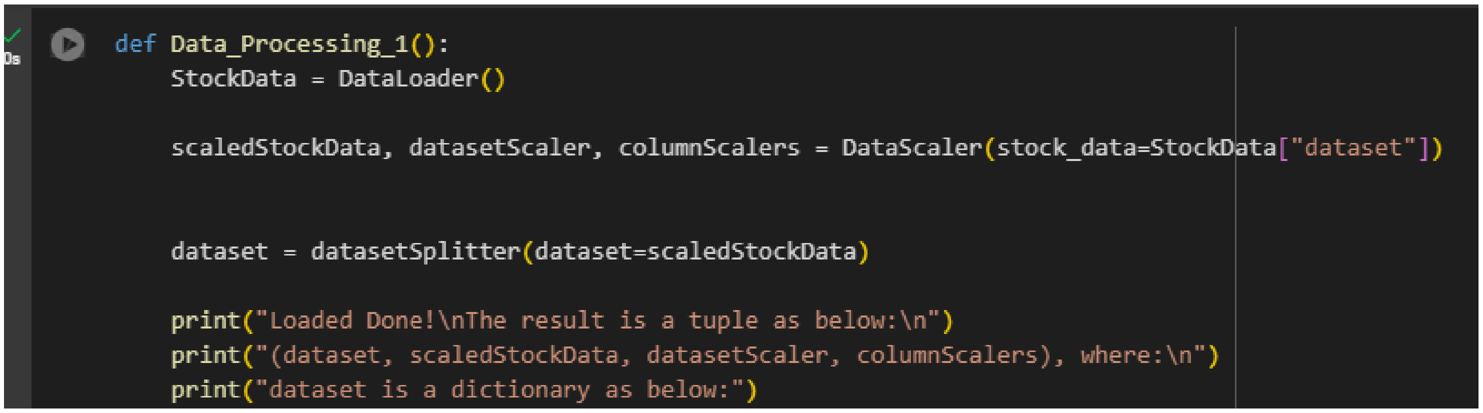


```
for column in feature_columns:
    dataset_in_column = dataset[column].values.reshape(-1, 1)      # <class 'numpy.ndarray'>
    x_data = []
    y_data = []

    for x in range(look_up_days, len(dataset_in_column)):
        x_data.append(dataset_in_column[x - look_up_days:x, 0])
        y_data.append(dataset_in_column[x, 0])

    splitResult["X_training_set"][column], splitResult["X_testing_set"][column], splitResult["Y_training_set"][column], splitResult['Y_testing_set'][column] = train_te

    ## Converting to numpy.array

    for column in feature_columns:
        splitResult["X_training_set"][column] = np.array(splitResult["X_training_set"][column])
        splitResult["Y_training_set"][column] = np.array(splitResult["Y_training_set"][column])
        splitResult["X_testing_set"][column] = np.array(splitResult["X_testing_set"][column])
        splitResult["Y_testing_set"][column] = np.array(splitResult["Y_testing_set"][column])

return splitResult
```

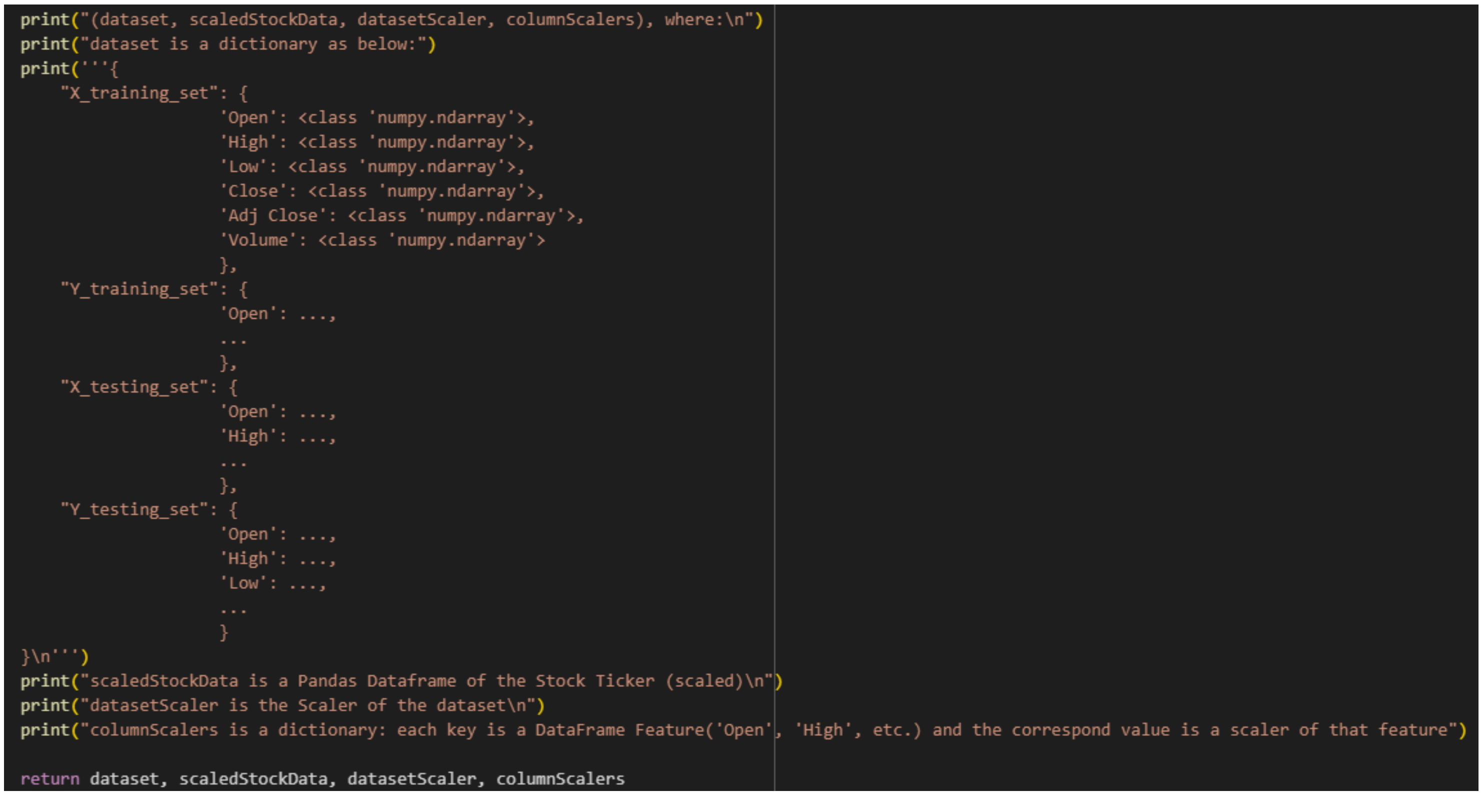
The loop here extracts the values as a numpy array and reshapes it to have one column and as many rows as needed. This is done to make the array compatible with the scaler and the neural network model.

x_data.append(dataset_in_column[x - look_up_days:x, 0]): This line appends a slice of the dataset_in_column input array to the x_data list. The slice contains the values from the x-look_up_days row to the x-1 row, and from the 0th column.

y_data.append(dataset_in_column[x, 0]): Thí line appends the value of the dataset_in_column output array to the y_data list. The value is at the xth row and the 0th column. This means that for each time t, the y_data will contain the value of the column at time t. For example, if t is 31, then y_data will contain the value at row 31, which is the value at day 31.

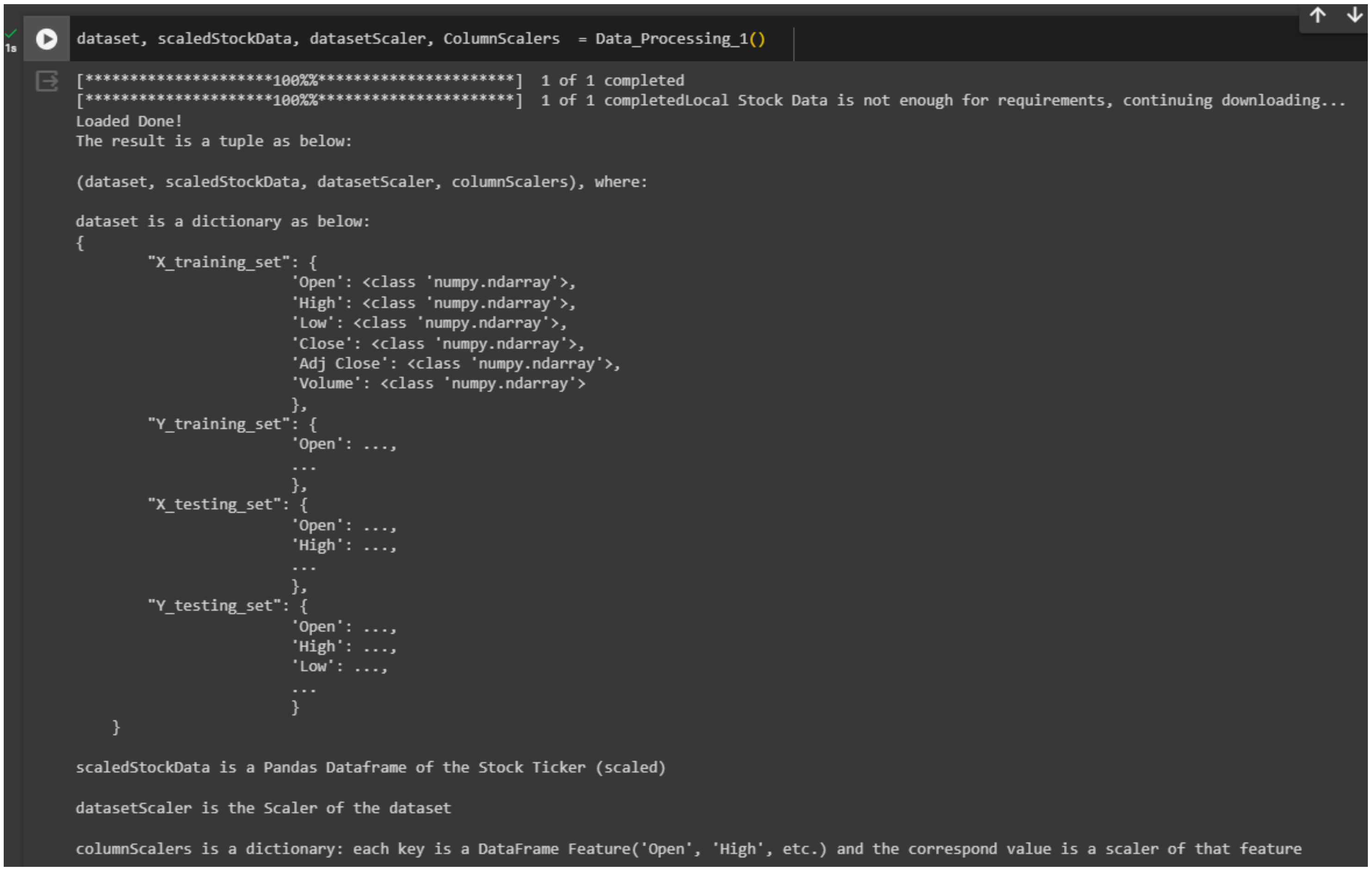


```
def Data_Processing_1():
    StockData = DataLoader()

    scaledStockData, datasetScaler, columnScalers = DataScaler(stock_data=StockData["dataset"])


    dataset = datasetSplitter(dataset=scaledStockData)

    print("Loaded Done!\nThe result is a tuple as below:\n")
    print("(dataset, scaledStockData, datasetScaler, columnScalers), where:\n")
    print("dataset is a dictionary as below:")
```



```
print("(dataset, scaledStockData, datasetScaler, columnScalers), where:\n")
print("dataset is a dictionary as below:")
print('''{
    "X_training_set": {
                    'Open': <class 'numpy.ndarray'>,
                    'High': <class 'numpy.ndarray'>,
                    'Low': <class 'numpy.ndarray'>,
                    'Close': <class 'numpy.ndarray'>,
                    'Adj Close': <class 'numpy.ndarray'>,
                    'Volume': <class 'numpy.ndarray'>
                    },
    "Y_training_set": {
                    'Open': ...,
                    ...
                    },
    "X_testing_set": {
                    'Open': ...,
                    'High': ...,
                    ...
                    },
    "Y_testing_set": {
                    'Open': ...,
                    'High': ...,
                    'Low': ...,
                    ...
                    }
}\n''')
print("scaledStockData is a Pandas Dataframe of the Stock Ticker (scaled)\n")
print("datasetScaler is the Scaler of the dataset\n")
print("columnScalers is a dictionary: each key is a DataFrame Feature('Open', 'High', etc.) and the correspond value is a scaler of that feature")

return dataset, scaledStockData, datasetScaler, columnScalers
```

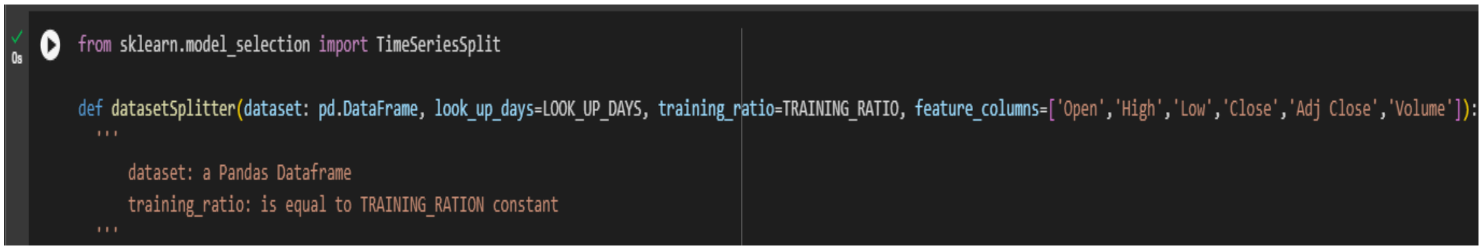
We defines a function called Data_Processing_1 that calls the DataLoader function to get the stock data, the DataScaler function to scale the data, and the datasetSplitter function to split the data. It also prints some fundamental information about the result tuple, which contains the dataset, the scaledStockData, the datasetScaler, and the columnScalers.



```
dataset, scaledStockData, datasetScaler, ColumnScalers  = Data_Processing_1()

[********************100%%**********************]  1 of 1 completed
[********************100%%**********************]  1 of 1 completedLocal Stock Data is not enough for requirements, continuing downloading...
Loaded Done!
The result is a tuple as below:

(dataset, scaledStockData, datasetScaler, columnScalers), where:

dataset is a dictionary as below:
{
        "X_training_set": {
                        'Open': <class 'numpy.ndarray'>,
                        'High': <class 'numpy.ndarray'>,
                        'Low': <class 'numpy.ndarray'>,
                        'Close': <class 'numpy.ndarray'>,
                        'Adj Close': <class 'numpy.ndarray'>,
                        'Volume': <class 'numpy.ndarray'>
                        },
        "Y_training_set": {
                        'Open': ...,
                        ...
                        },
        "X_testing_set": {
                        'Open': ...,
                        'High': ...,
                        ...
                        },
        "Y_testing_set": {
                        'Open': ...,
                        'High': ...,
                        'Low': ...,
                        ...
                        }
    }

scaledStockData is a Pandas Dataframe of the Stock Ticker (scaled)

datasetScaler is the Scaler of the dataset

columnScalers is a dictionary: each key is a DataFrame Feature('Open', 'High', etc.) and the correspond value is a scaler of that feature
```

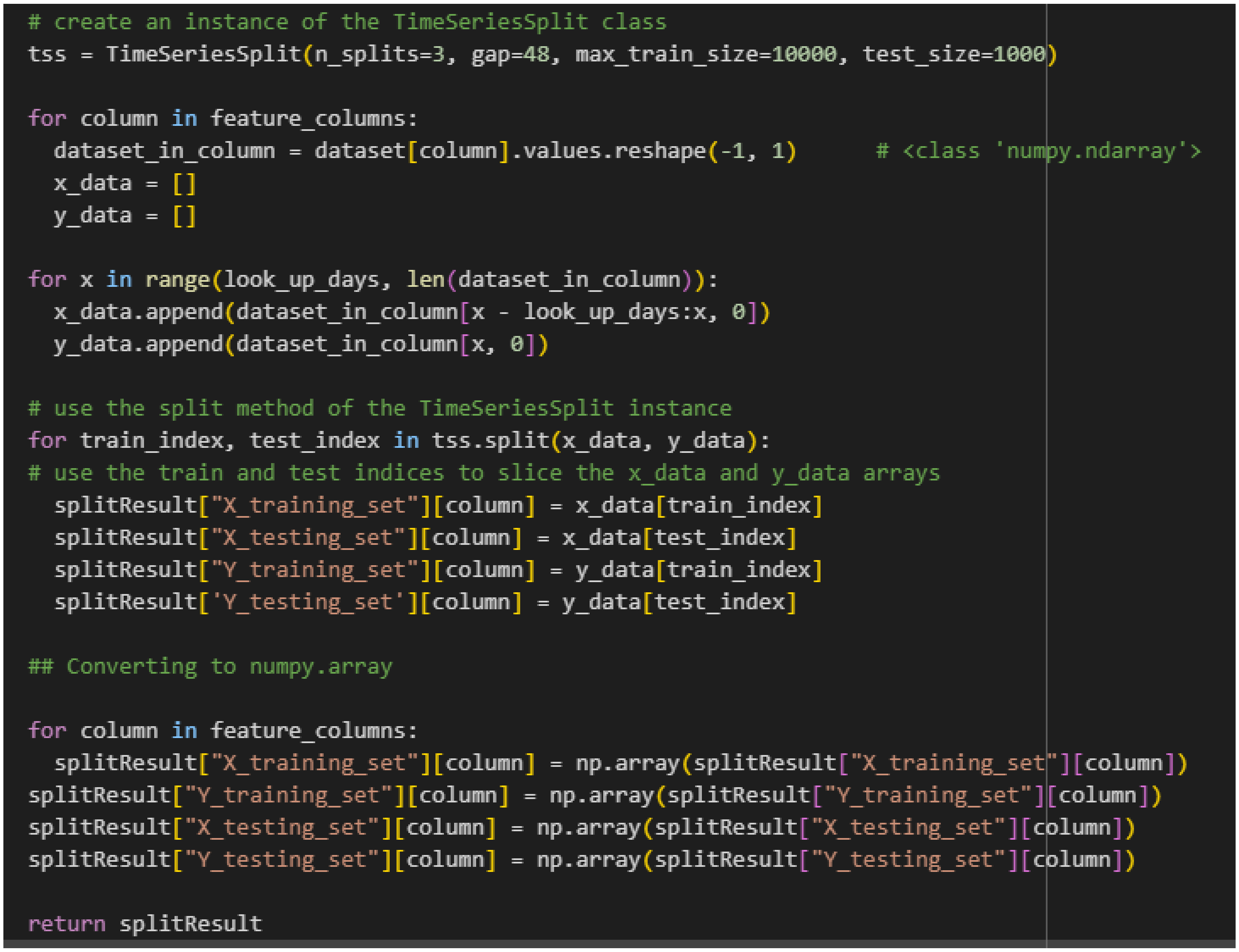
And here is the results from the Data_Processing_1() function, with all required output described in the Task specification.

We also consider to improve our curent data preparation model by implementing the sklearn.model_selection.TimeSeriesSplit method to split our data into training and testing sets in a time-series manner since this will preserve the temporal order and structure of our data, and avoid any data leakage or contamination. Besides, we can also use the sklearn.metrics module to evaluate our model performance on the testing set, such as the mean absolute error, the mean squared error, the root mean squared error, the R-squared score, etc.



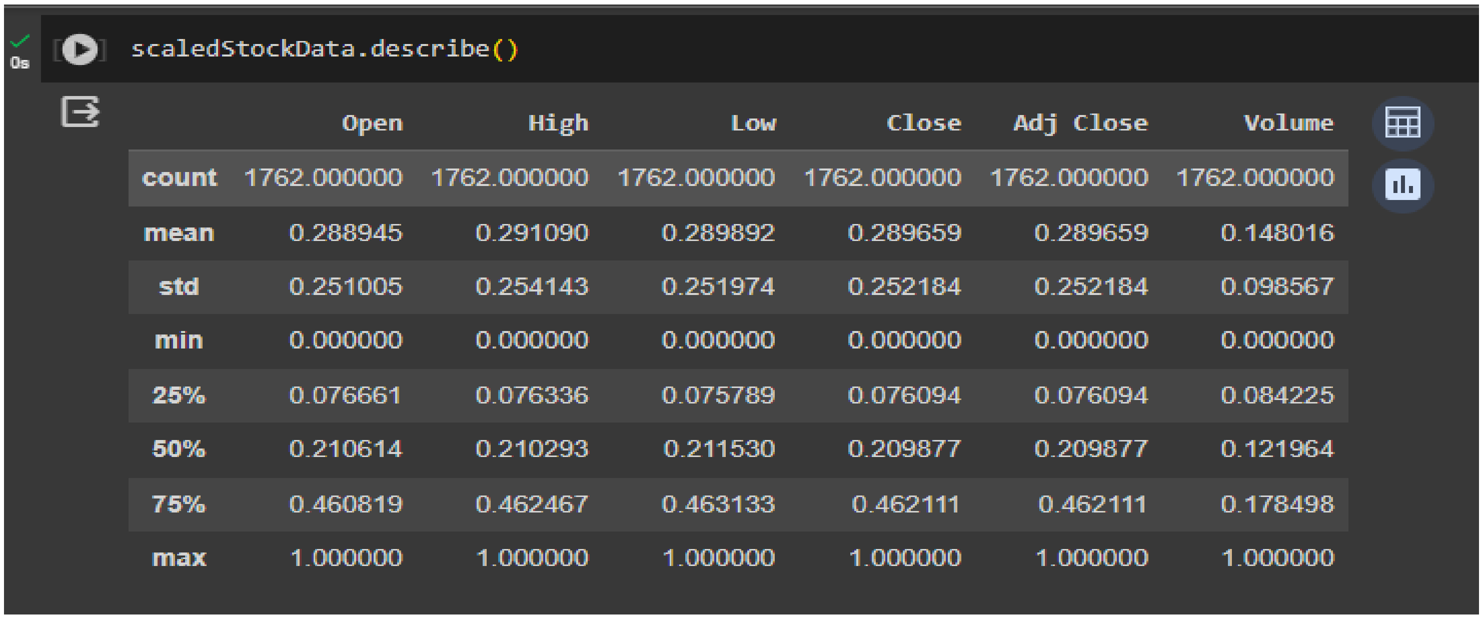
```
from sklearn.model_selection import TimeSeriesSplit

def datasetSplitter(dataset: pd.DataFrame, look_up_days=LOOK_UP_DAYS, training_ratio=TRAINING_RATIO, feature_columns=['Open','High','Low','Close','Adj Close','Volume']):
  '''
      dataset: a Pandas Dataframe
      training_ratio: is equal to TRAINING_RATION constant
  '''
```

```
#result:
splitResult = {
    "X_training_set": {
        "Open": None,
        "High": None,
        "Low": None,
        "Close": None,
        "Adj Close": None,
        "Volume": None
    },
    "Y_training_set": {
        "Open": None,
        "High": None,
        "Low": None,
        "Close": None,
        "Adj Close": None,
        "Volume": None
    },
    "X_testing_set": {
        "Open": None,
        "High": None,
        "Low": None,
        "Close": None,
        "Adj Close": None,
        "Volume": None
    },
    "Y_testing_set": {
        "Open": None,
        "High": None,
        "Low": None,
        "Close": None,
        "Adj Close": None,
        "Volume": None
    }
}
```



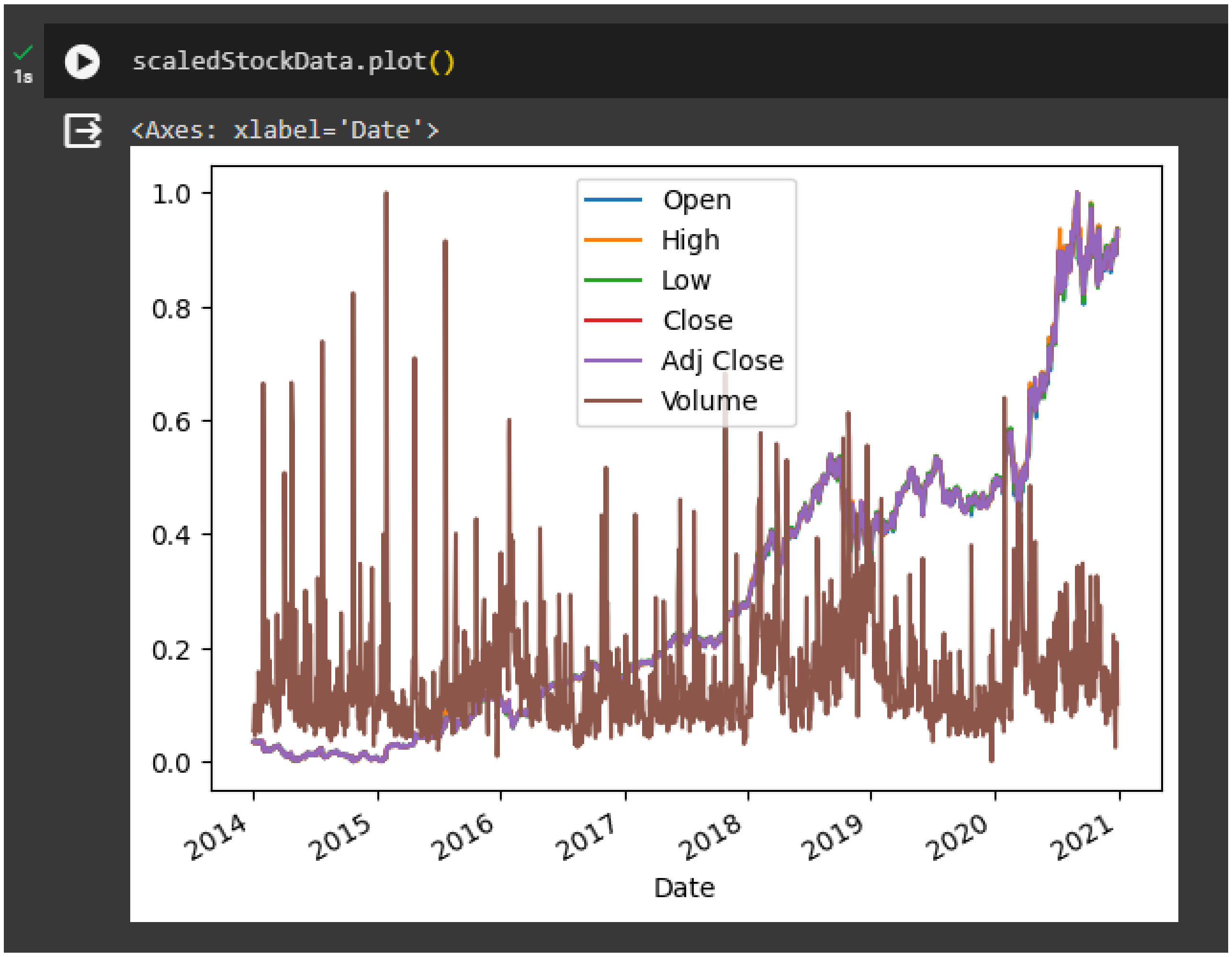
```
# create an instance of the TimeSeriesSplit class
tss = TimeSeriesSplit(n_splits=3, gap=48, max_train_size=10000, test_size=1000)

for column in feature_columns:
  dataset_in_column = dataset[column].values.reshape(-1, 1)      # <class 'numpy.ndarray'>
  x_data = []
  y_data = []

for x in range(look_up_days, len(dataset_in_column)):
  x_data.append(dataset_in_column[x - look_up_days:x, 0])
  y_data.append(dataset_in_column[x, 0])

# use the split method of the TimeSeriesSplit instance
for train_index, test_index in tss.split(x_data, y_data):
# use the train and test indices to slice the x_data and y_data arrays
  splitResult["X_training_set"][column] = x_data[train_index]
  splitResult["X_testing_set"][column] = x_data[test_index]
  splitResult["Y_training_set"][column] = y_data[train_index]
  splitResult['Y_testing_set'][column] = y_data[test_index]

## Converting to numpy.array

for column in feature_columns:
  splitResult["X_training_set"][column] = np.array(splitResult["X_training_set"][column])
splitResult["Y_training_set"][column] = np.array(splitResult["Y_training_set"][column])
splitResult["X_testing_set"][column] = np.array(splitResult["X_testing_set"][column])
splitResult["Y_testing_set"][column] = np.array(splitResult["Y_testing_set"][column])

return splitResult
```

Some other ways to further explore and analyze the current preprocessed inputs as a result of this whole data preprocessing step:



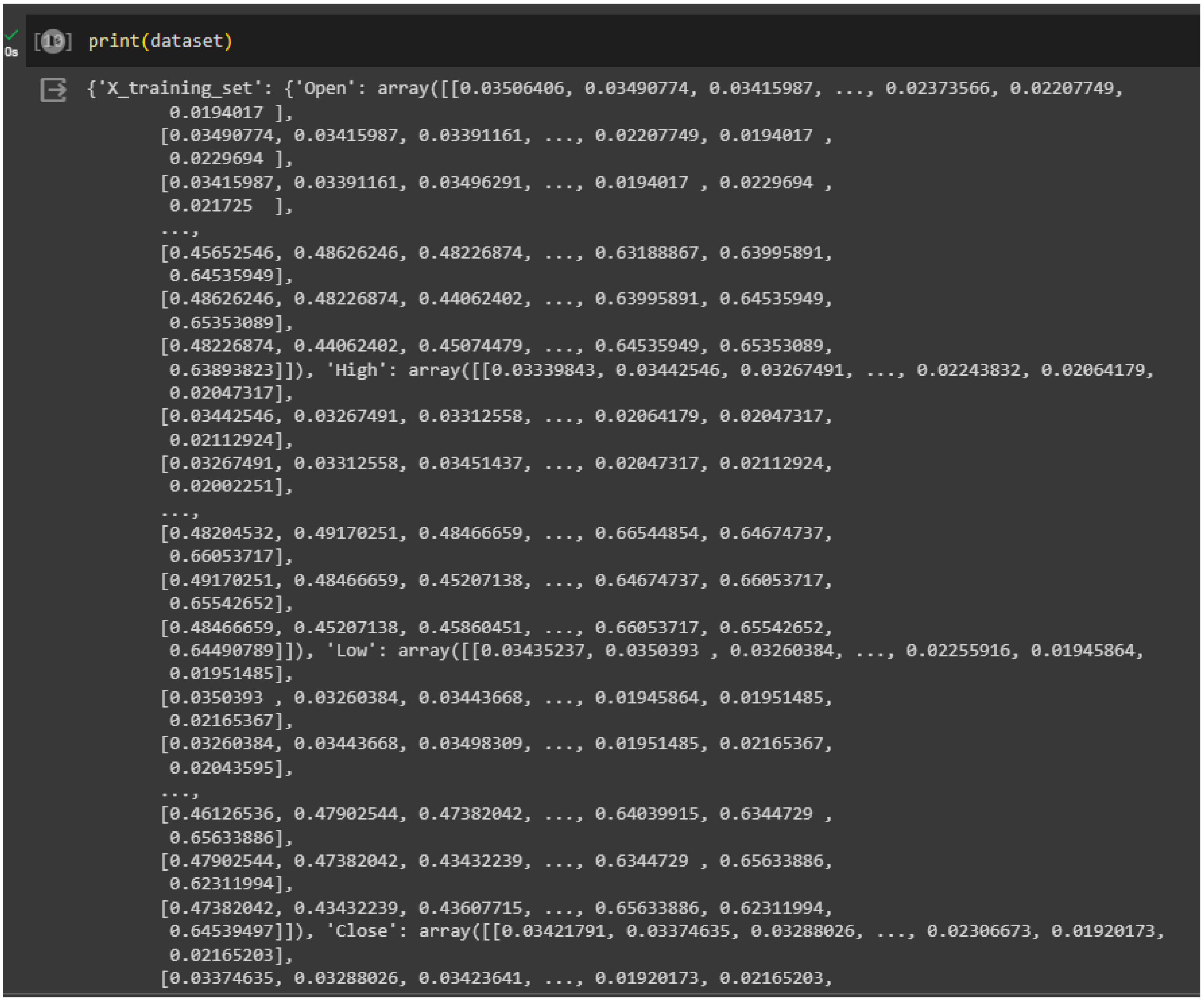
```
scaledStockData.describe()
```

```
              Open         High          Low        Close    Adj Close       Volume
count  1762.000000  1762.000000  1762.000000  1762.000000  1762.000000  1762.000000
mean      0.288945     0.291090     0.289892     0.289659     0.289659     0.148016
std       0.251005     0.254143     0.251974     0.252184     0.252184     0.098567
min       0.000000     0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.076661     0.076336     0.075789     0.076094     0.076094     0.084225
50%       0.210614     0.210293     0.211530     0.209877     0.209877     0.121964
75%       0.460819     0.462467     0.463133     0.462111     0.462111     0.178498
max       1.000000     1.000000     1.000000     1.000000     1.000000     1.000000
```



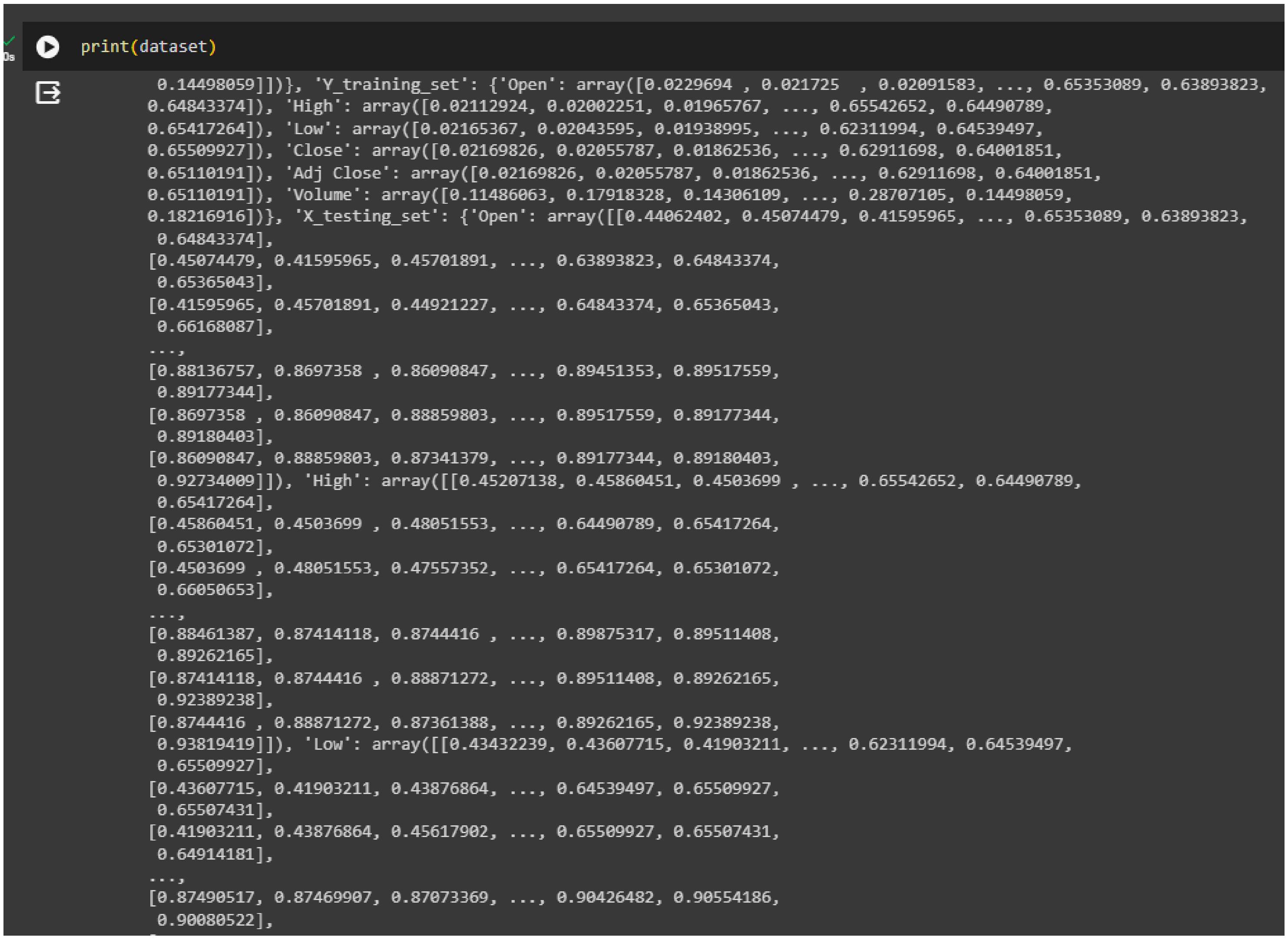
```
scaledStockData.plot()
```

```
<Axes: xlabel='Date'>
```



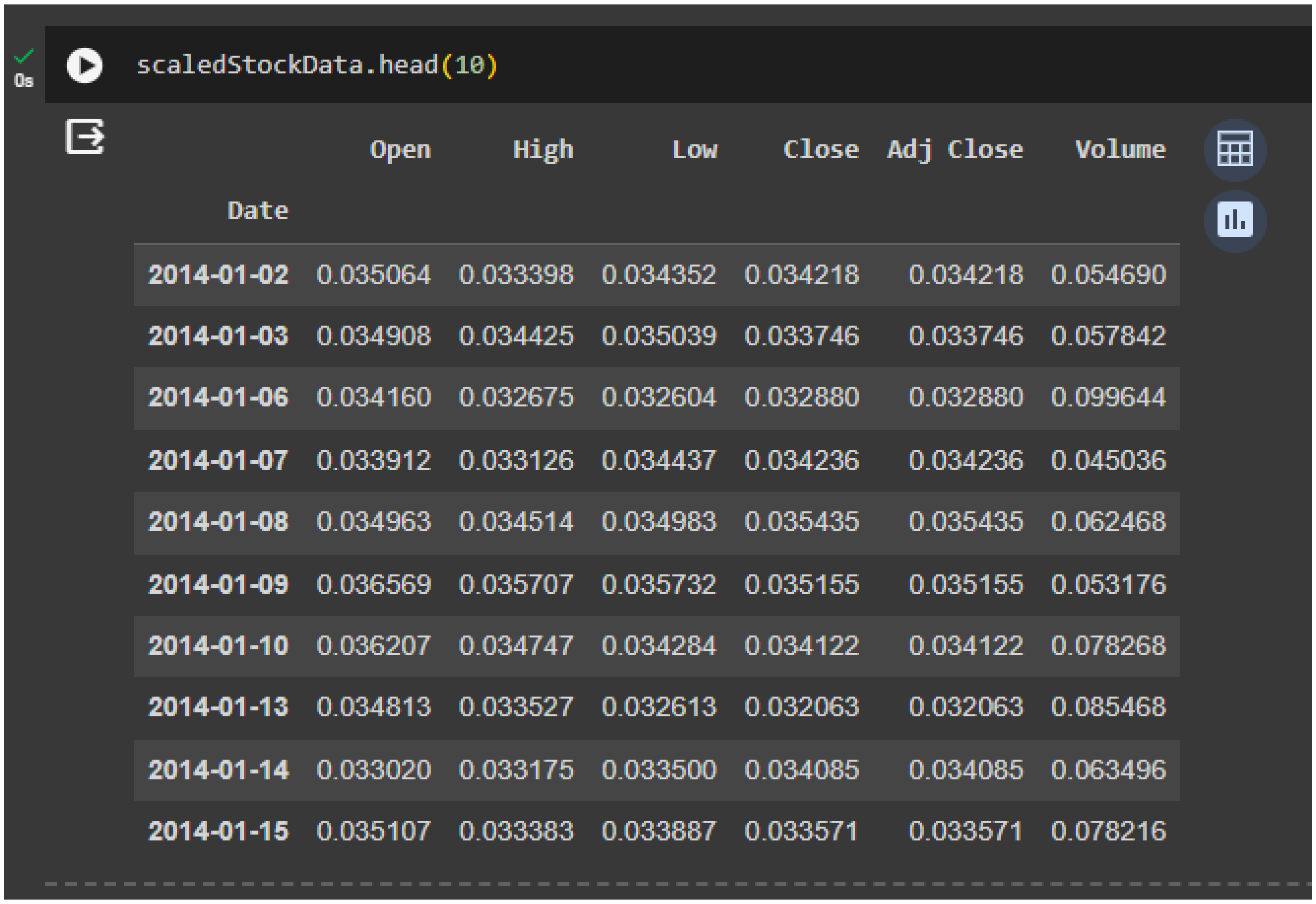
```
[13] print(dataset)
```

```
{'X_training_set': {'Open': array([[0.03506406, 0.03490774, 0.03415987, ..., 0.02373566, 0.02207749,
         0.0194017 ],
        [0.03490774, 0.03415987, 0.03391161, ..., 0.02207749, 0.0194017 ,
         0.0229694 ],
        [0.03415987, 0.03391161, 0.03496291, ..., 0.0194017 , 0.0229694 ,
         0.021725  ],
        ...,
        [0.45652546, 0.48626246, 0.48226874, ..., 0.63188867, 0.63995891,
         0.64535949],
        [0.48626246, 0.48226874, 0.44062402, ..., 0.63995891, 0.64535949,
         0.65353089],
        [0.48226874, 0.44062402, 0.45074479, ..., 0.64535949, 0.65353089,
         0.63893823]]), 'High': array([[0.03339843, 0.03442546, 0.03267491, ..., 0.02243832, 0.02064179,
         0.02047317],
        [0.03442546, 0.03267491, 0.03312558, ..., 0.02064179, 0.02047317,
         0.02112924],
        [0.03267491, 0.03312558, 0.03451437, ..., 0.02047317, 0.02112924,
         0.02002251],
        ...,
        [0.48204532, 0.49170251, 0.48466659, ..., 0.66544854, 0.64674737,
         0.66053717],
        [0.49170251, 0.48466659, 0.45207138, ..., 0.64674737, 0.66053717,
         0.65542652],
        [0.48466659, 0.45207138, 0.45860451, ..., 0.66053717, 0.65542652,
         0.64490789]]), 'Low': array([[0.03435237, 0.0350393 , 0.03260384, ..., 0.02255916, 0.01945864,
         0.01951485],
        [0.0350393 , 0.03260384, 0.03443668, ..., 0.01945864, 0.01951485,
         0.02165367],
        [0.03260384, 0.03443668, 0.03498309, ..., 0.01951485, 0.02165367,
         0.02043595],
        ...,
        [0.46126536, 0.47902544, 0.47382042, ..., 0.64039915, 0.6344729 ,
         0.65633886],
        [0.47902544, 0.47382042, 0.43432239, ..., 0.6344729 , 0.65633886,
         0.62311994],
        [0.47382042, 0.43432239, 0.43607715, ..., 0.65633886, 0.62311994,
         0.64539497]]), 'Close': array([[0.03421791, 0.03374635, 0.03288026, ..., 0.02306673, 0.01920173,
         0.02165203],
        [0.03374635, 0.03288026, 0.03423641, ..., 0.01920173, 0.02165203,
```



```python
print(dataset)
```

```
 0.14498059]])}, 'Y_training_set': {'Open': array([0.0229694 , 0.021725  , 0.02091583, ..., 0.65353089, 0.63893823,
0.64843374]), 'High': array([0.02112924, 0.02002251, 0.01965767, ..., 0.65542652, 0.64490789,
0.65417264]), 'Low': array([0.02165367, 0.02043595, 0.01938995, ..., 0.62311994, 0.64539497,
0.65509927]), 'Close': array([0.02169826, 0.02055787, 0.01862536, ..., 0.62911698, 0.64001851,
0.65110191]), 'Adj Close': array([0.02169826, 0.02055787, 0.01862536, ..., 0.62911698, 0.64001851,
0.65110191]), 'Volume': array([0.11486063, 0.17918328, 0.14306109, ..., 0.28707105, 0.14498059,
0.18216916])}, 'X_testing_set': {'Open': array([[0.44062402, 0.45074479, 0.41595965, ..., 0.65353089, 0.63893823,
 0.64843374],
[0.45074479, 0.41595965, 0.45701891, ..., 0.63893823, 0.64843374,
 0.65365043],
[0.41595965, 0.45701891, 0.44921227, ..., 0.64843374, 0.65365043,
 0.66168087],
...,
[0.88136757, 0.8697358 , 0.86090847, ..., 0.89451353, 0.89517559,
 0.89177344],
[0.8697358 , 0.86090847, 0.88859803, ..., 0.89517559, 0.89177344,
 0.89180403],
[0.86090847, 0.88859803, 0.87341379, ..., 0.89177344, 0.89180403,
 0.92734009]]), 'High': array([[0.45207138, 0.45860451, 0.4503699 , ..., 0.65542652, 0.64490789,
 0.65417264],
[0.45860451, 0.4503699 , 0.48051553, ..., 0.64490789, 0.65417264,
 0.65301072],
[0.4503699 , 0.48051553, 0.47557352, ..., 0.65417264, 0.65301072,
 0.66050653],
...,
[0.88461387, 0.87414118, 0.8744416 , ..., 0.89875317, 0.89511408,
 0.89262165],
[0.87414118, 0.8744416 , 0.88871272, ..., 0.89511408, 0.89262165,
 0.92389238],
[0.8744416 , 0.88871272, 0.87361388, ..., 0.89262165, 0.92389238,
 0.93819419]]), 'Low': array([[0.43432239, 0.43607715, 0.41903211, ..., 0.62311994, 0.64539497,
 0.65509927],
[0.43607715, 0.41903211, 0.43876864, ..., 0.64539497, 0.65509927,
 0.65507431],
[0.41903211, 0.43876864, 0.45617902, ..., 0.65509927, 0.65507431,
 0.64914181],
...,
[0.87490517, 0.87469907, 0.87073369, ..., 0.90426482, 0.90554186,
 0.90080522],
```



```
scaledStockData.head(10)
```

```
                Open      High       Low     Close  Adj Close    Volume
Date
2014-01-02  0.035064  0.033398  0.034352  0.034218   0.034218  0.054690
2014-01-03  0.034908  0.034425  0.035039  0.033746   0.033746  0.057842
2014-01-06  0.034160  0.032675  0.032604  0.032880   0.032880  0.099644
2014-01-07  0.033912  0.033126  0.034437  0.034236   0.034236  0.045036
2014-01-08  0.034963  0.034514  0.034983  0.035435   0.035435  0.062468
2014-01-09  0.036569  0.035707  0.035732  0.035155   0.035155  0.053176
2014-01-10  0.036207  0.034747  0.034284  0.034122   0.034122  0.078268
2014-01-13  0.034813  0.033527  0.032613  0.032063   0.032063  0.085468
2014-01-14  0.033020  0.033175  0.033500  0.034085   0.034085  0.063496
2014-01-15  0.035107  0.033383  0.033887  0.033571   0.033571  0.078216
```