COS30018 – Intelligent System

Option B: Stock Price Prediction

Report v0.2

Name Phong Tran
Student Id:104334842

Class: 1-5
Tutor: Tuan Dung Lai

Table of Contents

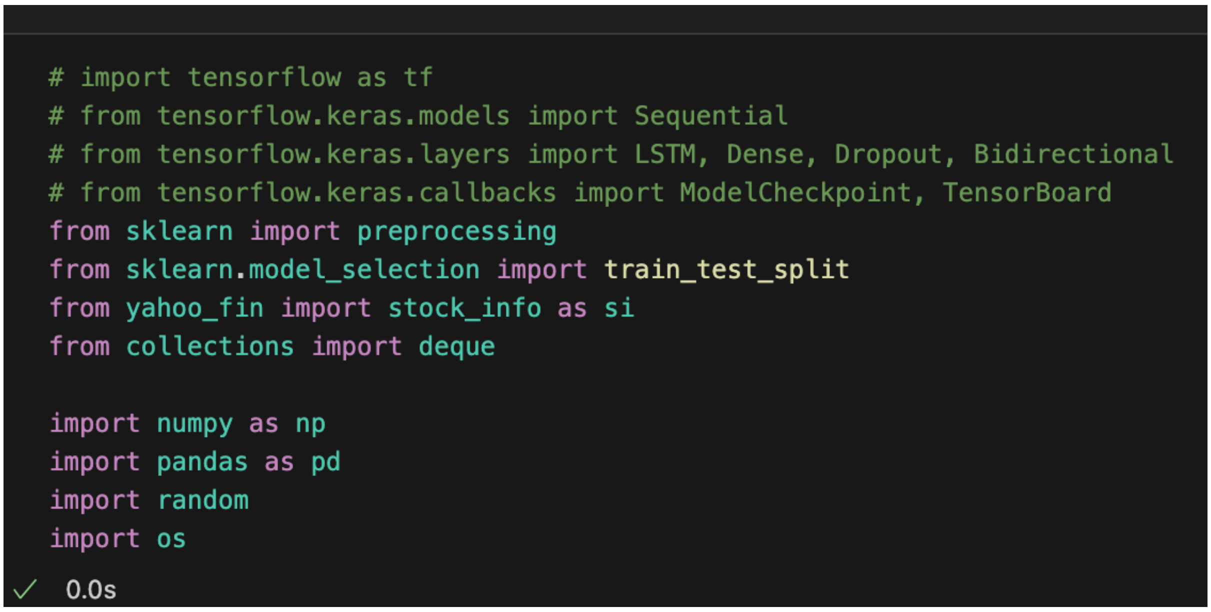
Introduction

This report is about the function in terms of processing the data from yfinance – Yahoo finance and download the data stock market price from AMZN ticker, which also called Amazon in abbreviation before saving locally to save time for future uses. This function will range specific dates, from start to end at dataset. Then, it will helps to deal with NaN issue with two approaches, before splitting the data into train/test data to specify it by date or randomly. Finally, this function contains an option to scale feature columns and store the scalers in data structure to allow future access to these scalers.

Implementation

1. Import libraries

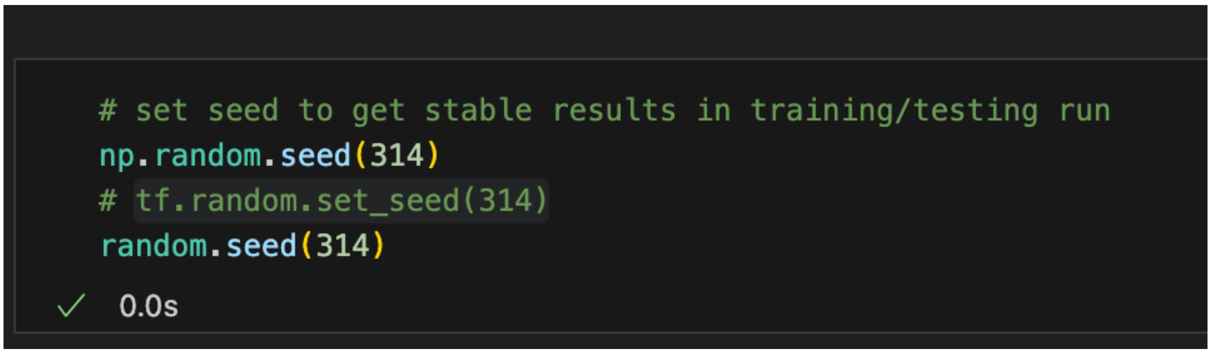
Comment TensorFlow as it unused for this Processing Data



```
# import tensorflow as tf
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
# from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from yahoo_fin import stock_info as si
from collections import deque

import numpy as np
import pandas as pd
import random
import os
```

✓ 0.0s

2. Set seed



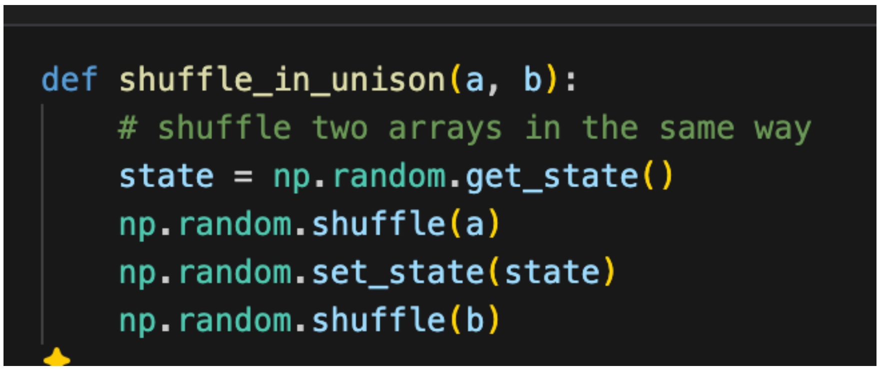
```
# set seed to get stable results in training/testing run
np.random.seed(314)
# tf.random.set_seed(314)
random.seed(314)
```

✓ 0.0s

Remove only tensorflow seed as it unused

3. Write a function as requirements

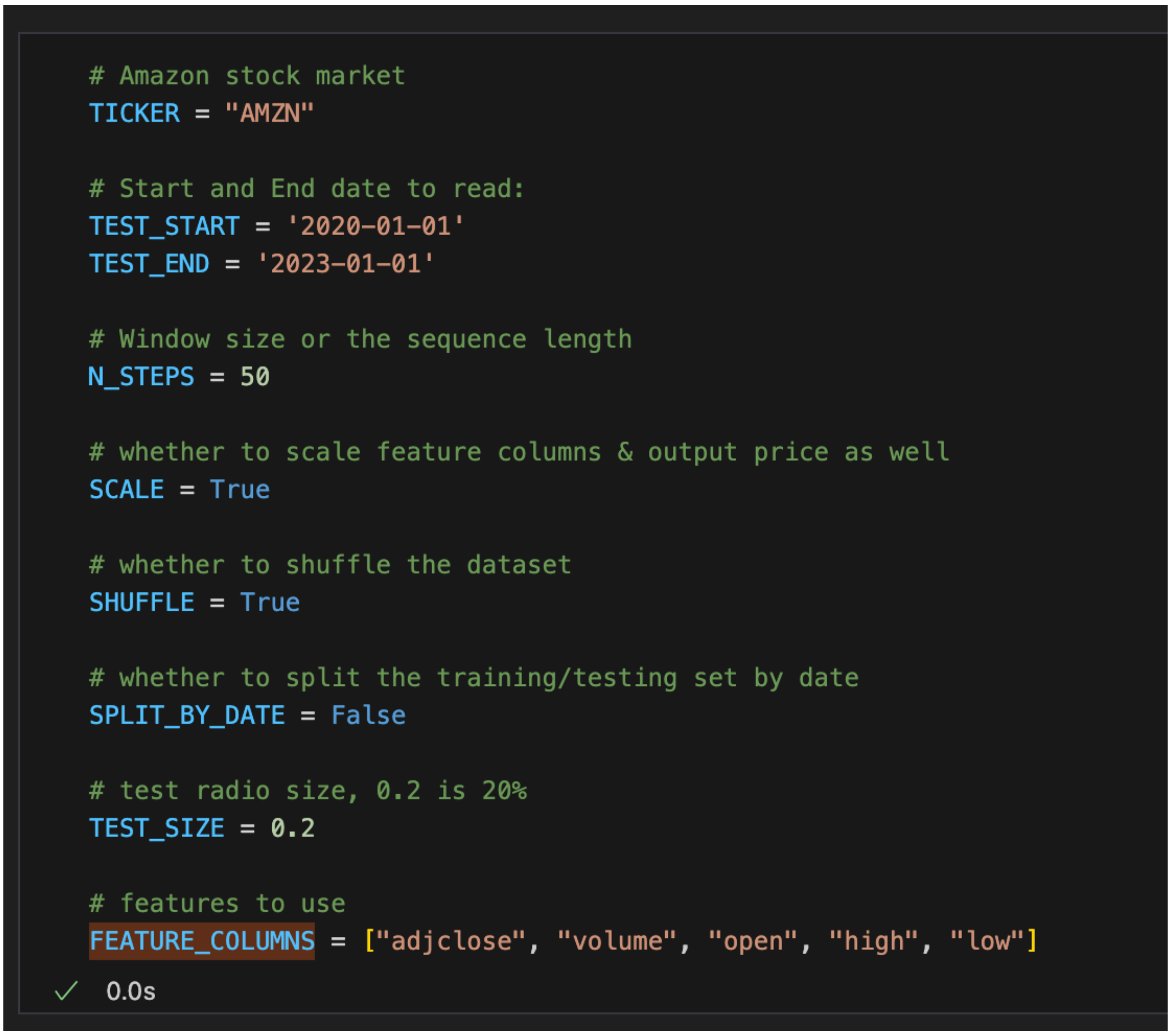
3.1. Keep shuffle_in_unison as it uses for shuffle randomly between X and Y in both train and test.



```
def shuffle_in_unison(a, b):
    # shuffle two arrays in the same way
    state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(state)
    np.random.shuffle(b)
```

3.2. `load_data()` with different parameters provided, such as start and end date, windows step, scale data between 0 and 1, shuffling dataset in both training and testing, allow to shuffle by date or not, test size split in percent, feature columns with OLHC and adj close.

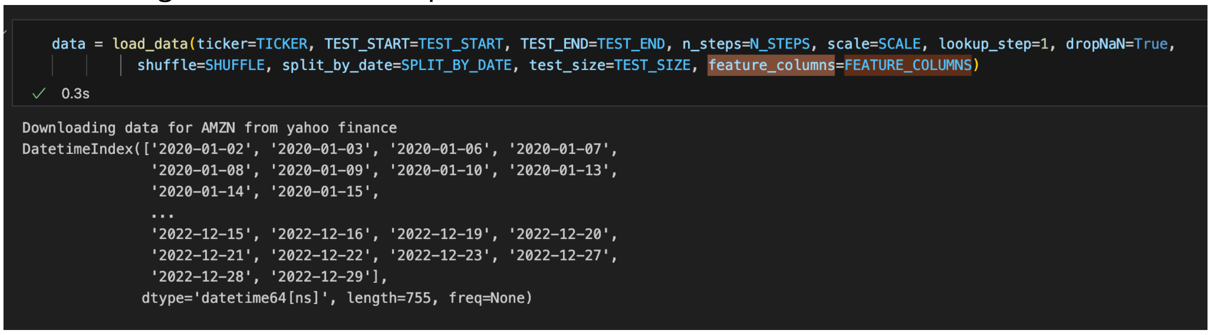
4. Variable (all in uppercase and separate with method parameters when assigned into a method)



```python
# Amazon stock market
TICKER = "AMZN"

# Start and End date to read:
TEST_START = '2020-01-01'
TEST_END = '2023-01-01'

# Window size or the sequence length
N_STEPS = 50

# whether to scale feature columns & output price as well
SCALE = True

# whether to shuffle the dataset
SHUFFLE = True

# whether to split the training/testing set by date
SPLIT_BY_DATE = False

# test radio size, 0.2 is 20%
TEST_SIZE = 0.2

# features to use
FEATURE_COLUMNS = ["adjclose", "volume", "open", "high", "low"]
```

✓ 0.0s

5. Assign the method with parameters fulfilled to variable

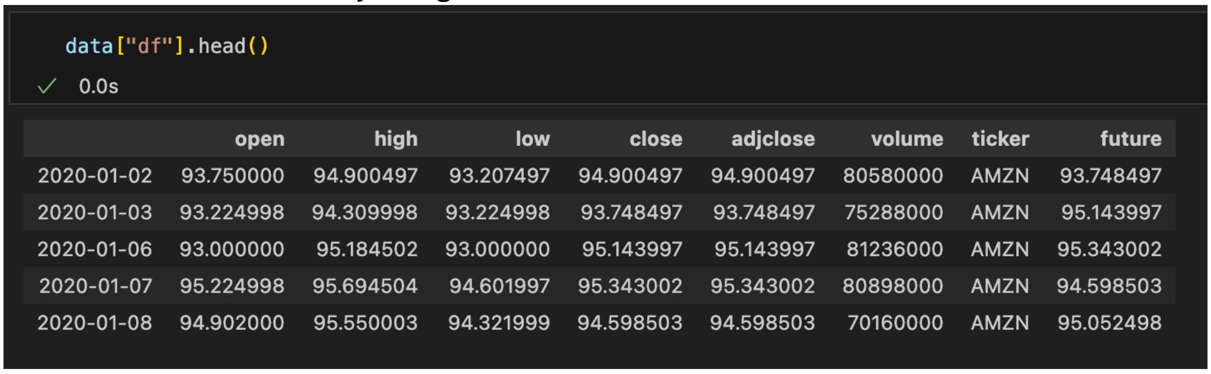


```python
data = load_data(ticker=TICKER, TEST_START=TEST_START, TEST_END=TEST_END, n_steps=N_STEPS, scale=SCALE, lookup_step=1, dropNaN=True,
        shuffle=SHUFFLE, split_by_date=SPLIT_BY_DATE, test_size=TEST_SIZE, feature_columns=FEATURE_COLUMNS)
```

✓ 0.3s

```
Downloading data for AMZN from yahoo finance
DatetimeIndex(['2020-01-02', '2020-01-03', '2020-01-06', '2020-01-07',
               '2020-01-08', '2020-01-09', '2020-01-10', '2020-01-13',
               '2020-01-14', '2020-01-15',
               ...
               '2022-12-15', '2022-12-16', '2022-12-19', '2022-12-20',
               '2022-12-21', '2022-12-22', '2022-12-23', '2022-12-27',
               '2022-12-28', '2022-12-29'],
              dtype='datetime64[ns]', length=755, freq=None)
```

6. Show first 5 rows by using head method



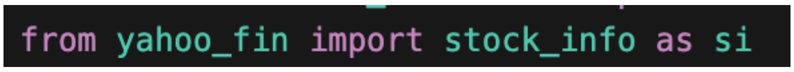
```python
data["df"].head()
```

✓ 0.0s

	open	high	low	close	adjclose	volume	ticker	future
2020-01-02	93.750000	94.900497	93.207497	94.900497	94.900497	80580000	AMZN	93.748497
2020-01-03	93.224998	94.309998	93.224998	93.748497	93.748497	75288000	AMZN	95.143997
2020-01-06	93.000000	95.184502	93.000000	95.143997	95.143997	81236000	AMZN	95.343002
2020-01-07	95.224998	95.694504	94.601997	95.343002	95.343002	80898000	AMZN	94.598503
2020-01-08	94.902000	95.550003	94.321999	94.598503	94.598503	70160000	AMZN	95.052498

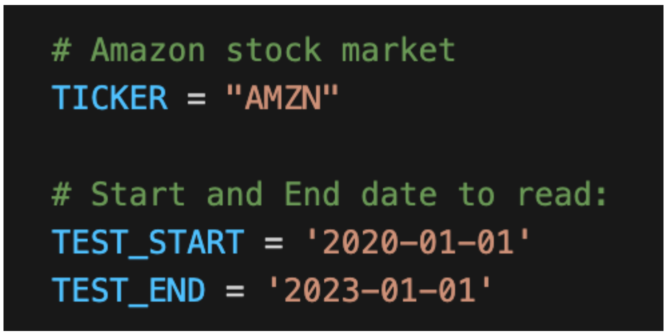
Function Explain

a)Allow to specify the start date and end date for the whole dataset as inputs



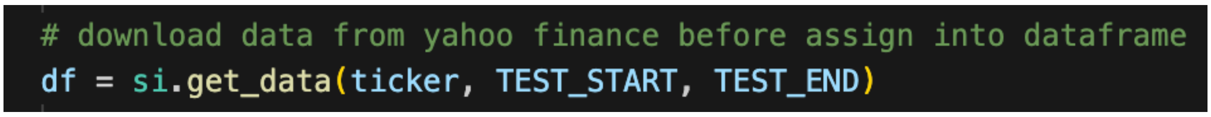
```python
from yahoo_fin import stock_info as si
```

Import module stock_info from yahoo finance, `as` si means we can input si instead of stock_info



```python
# Amazon stock market
TICKER = "AMZN"

# Start and End date to read:
TEST_START = '2020-01-01'
TEST_END = '2023-01-01'
```

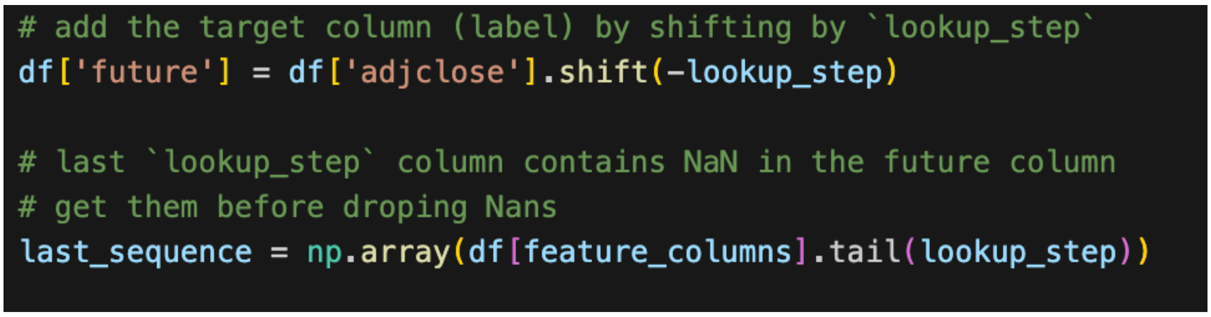
Specify tickers AMZN, also called Amazon in abbreviation. Start date and end date variable are specified in the image with format YYYY-MM-DD



```python
# download data from yahoo finance before assign into dataframe
df = si.get_data(ticker, TEST_START, TEST_END)
```

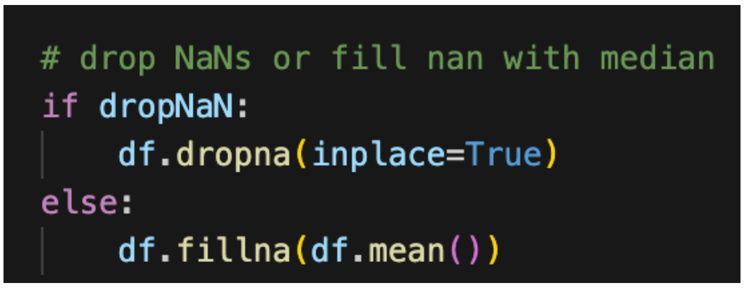
get_data method means download the data from stock info at yahoo finance, with specify ticker chosen from start date to end date

b)Deal with NaN issue in the data



```python
# add the target column (label) by shifting by `lookup_step`
df['future'] = df['adjclose'].shift(-lookup_step)

# last `lookup_step` column contains NaN in the future column
# get them before droping Nans
last_sequence = np.array(df[feature_columns].tail(lookup_step))
```

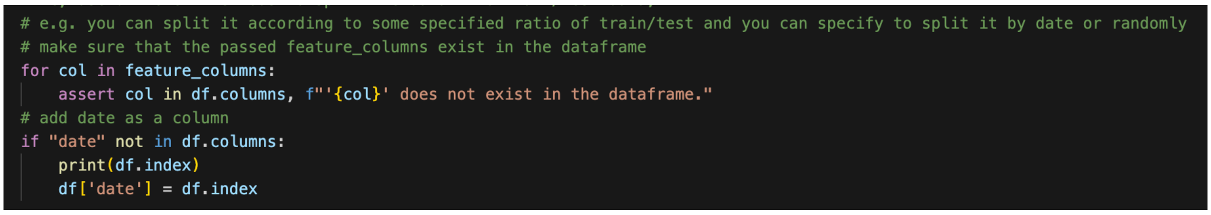
Assign target column `future`, by shifting to `adjclose` column



```python
# drop NaNs or fill nan with median
if dropNaN:
    df.dropna(inplace=True)
else:
    df.fillna(df.mean())
```

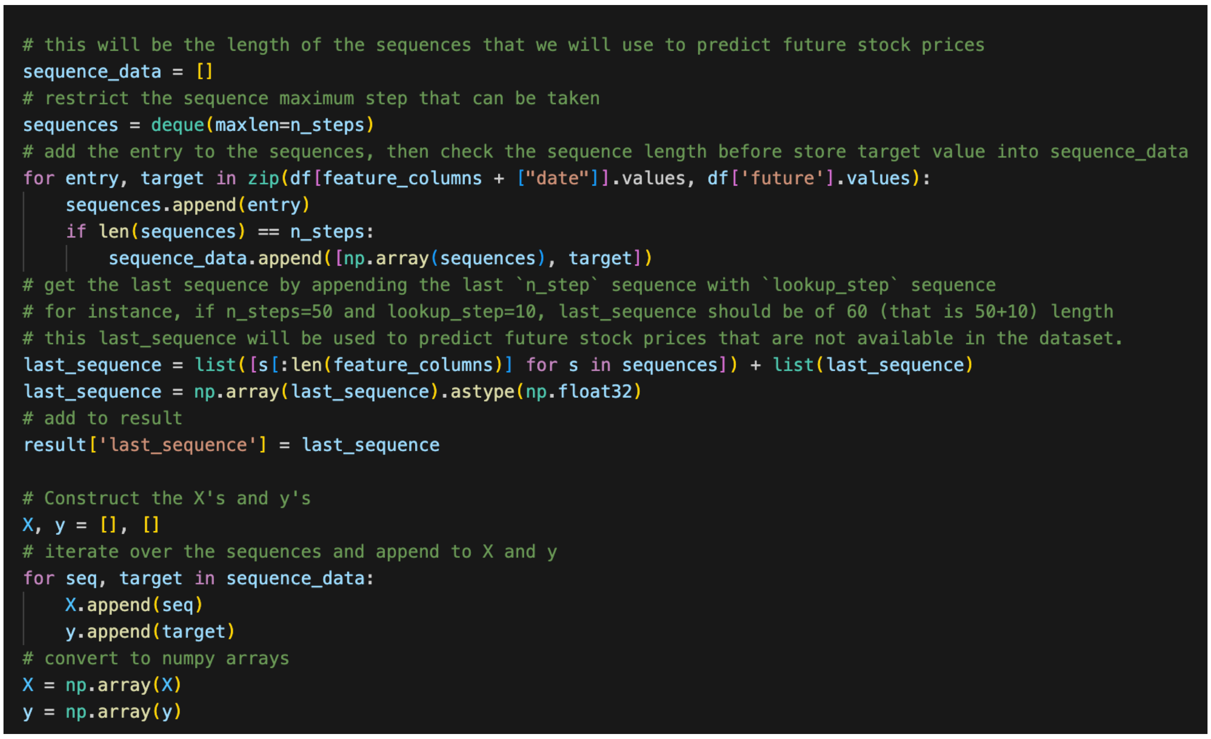
Condition if dropNaN true, will drop Nan, otherwise fill NaN with median value from dataframe.

c)Use different methods to split the data into train/test data (e.g you can split it according to some specified ratio of train/test and you can specify to split it by date or randomly



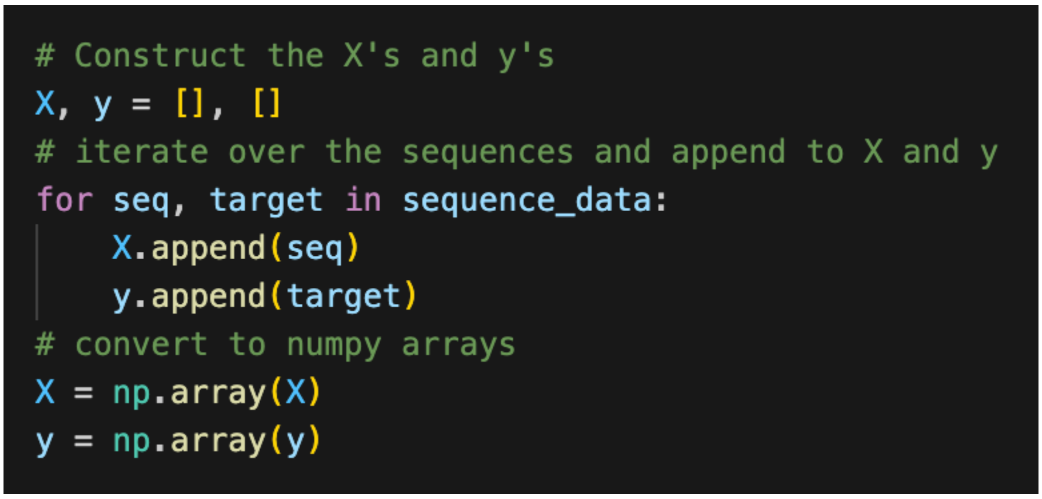
```python
# e.g. you can split it according to some specified ratio of train/test and you can specify to split it by date or randomly
# make sure that the passed feature_columns exist in the dataframe
for col in feature_columns:
    assert col in df.columns, f"'{col}' does not exist in the dataframe."
# add date as a column
if "date" not in df.columns:
    print(df.index)
    df['date'] = df.index
```

Check if feature column is exist, and check if date is not column. If ` date` column is not the table, we need to assign column date as index column.



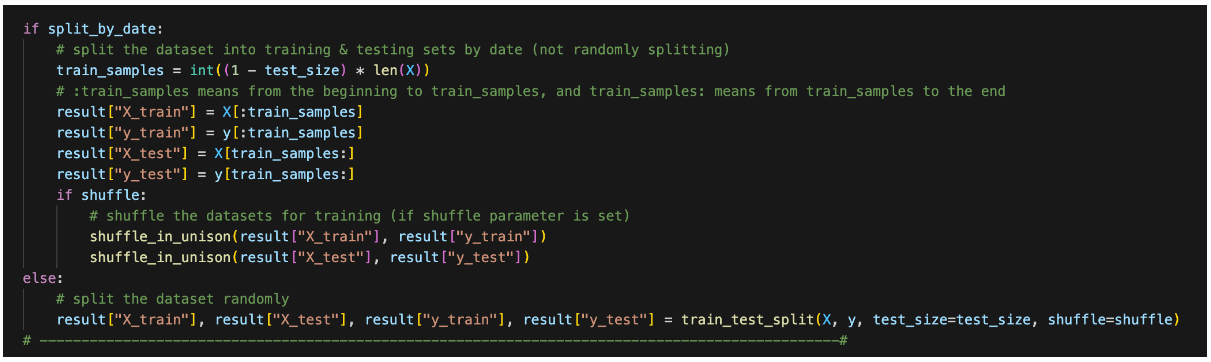
```python
# this will be the length of the sequences that we will use to predict future stock prices
sequence_data = []
# restrict the sequence maximum step that can be taken
sequences = deque(maxlen=n_steps)
# add the entry to the sequences, then check the sequence length before store target value into sequence_data
for entry, target in zip(df[feature_columns + ["date"]].values, df['future'].values):
    sequences.append(entry)
    if len(sequences) == n_steps:
        sequence_data.append([np.array(sequences), target])
# get the last sequence by appending the last `n_step` sequence with `lookup_step` sequence
# for instance, if n_steps=50 and lookup_step=10, last_sequence should be of 60 (that is 50+10) length
# this last_sequence will be used to predict future stock prices that are not available in the dataset.
last_sequence = list([s[:len(feature_columns)] for s in sequences]) + list(last_sequence)
last_sequence = np.array(last_sequence).astype(np.float32)
# add to result
result['last_sequence'] = last_sequence

# Construct the X's and y's
X, y = [], []
# iterate over the sequences and append to X and y
for seq, target in sequence_data:
    X.append(seq)
    y.append(target)
# convert to numpy arrays
X = np.array(X)
y = np.array(y)
```

Use sequence data include the entry and target value with date and future values. Use last sequence the result for predict the future stock price that are not available in the dataset.



```python
# Construct the X's and y's
X, y = [], []
# iterate over the sequences and append to X and y
for seq, target in sequence_data:
    X.append(seq)
    y.append(target)
# convert to numpy arrays
X = np.array(X)
y = np.array(y)
```

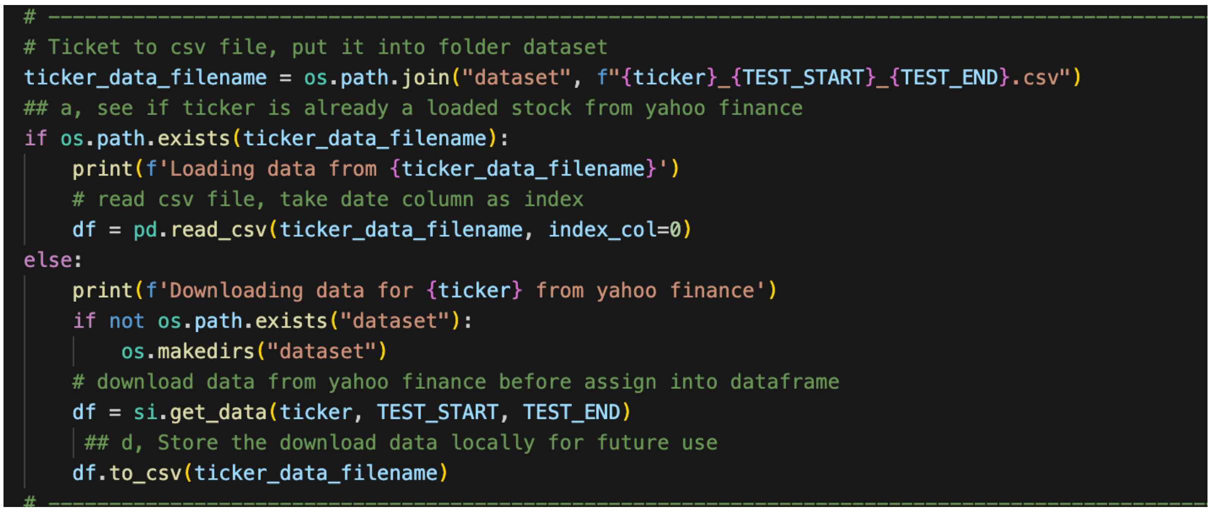
Use the sequence data to assign with X and y in numpy arrays.



```python
if split_by_date:
    # split the dataset into training & testing sets by date (not randomly splitting)
    train_samples = int((1 - test_size) * len(X))
    # :train_samples means from the beginning to train_samples, and train_samples: means from train_samples to the end
    result["X_train"] = X[:train_samples]
    result["y_train"] = y[:train_samples]
    result["X_test"] = X[train_samples:]
    result["y_test"] = y[train_samples:]
    if shuffle:
        # shuffle the datasets for training (if shuffle parameter is set)
        shuffle_in_unison(result["X_train"], result["y_train"])
        shuffle_in_unison(result["X_test"], result["y_test"])
else:
    # split the dataset randomly
    result["X_train"], result["X_test"], result["y_train"], result["y_test"] = train_test_split(X, y, test_size=test_size, shuffle=shuffle)
# -----------------------------------------------------------------------------------------------#
```

If we want to split by date, X_train and y_train will be used train_samples from the beginning, then X_test and Y_test will be used from train_samples till the end. If we need to shuffle the data, this will shuffle together with the result of X and Y train into one method, another test from X and Y into method to learn the general patterns from the data and avoid overfitting. If we put ` split_by_date` into false, X, y train and test will put into train_test_split module with test size of 0.2 ~ 80% training set 20% test set before shuffling randomly.

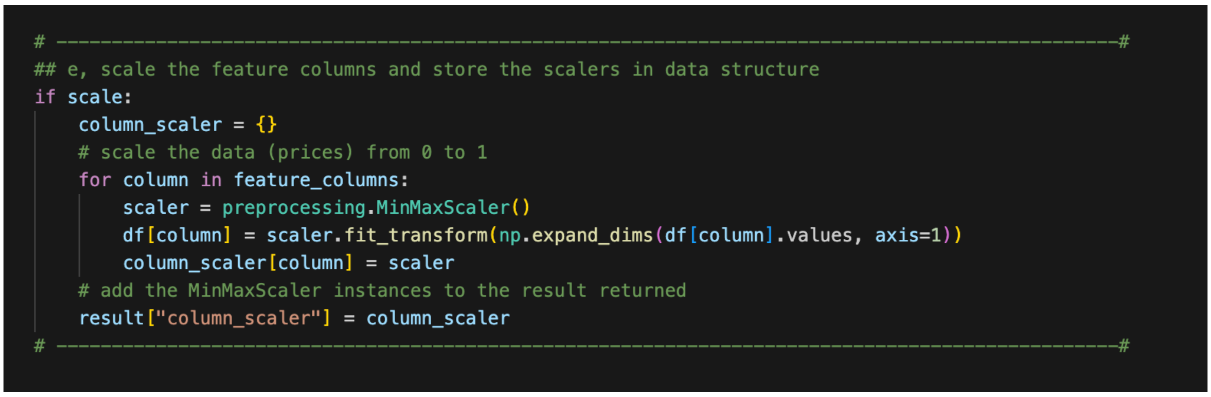
d)Store the download data and end date for the whole dataset as inputs



```
# ----------------------------------------------------------------------------------------------
# Ticket to csv file, put it into folder dataset
ticker_data_filename = os.path.join("dataset", f"{ticker}_{TEST_START}_{TEST_END}.csv")
## a, see if ticker is already a loaded stock from yahoo finance
if os.path.exists(ticker_data_filename):
    print(f'Loading data from {ticker_data_filename}')
    # read csv file, take date column as index
    df = pd.read_csv(ticker_data_filename, index_col=0)
else:
    print(f'Downloading data for {ticker} from yahoo finance')
    if not os.path.exists("dataset"):
        os.makedirs("dataset")
    # download data from yahoo finance before assign into dataframe
    df = si.get_data(ticker, TEST_START, TEST_END)
     ## d, Store the download data locally for future use
    df.to_csv(ticker_data_filename)
# ----------------------------------------------------------------------------------------------
```

If the path did not exist the dataset, it must be created. Then if the data is saved locally by downloading previously, we need to make sure that the file csv can be able to seen with structure format as it seen from the variable ‘ticker_data_filename’ . Otherwise, we need to download the dataset before saving csv file at dataset folder.

e)Option to scale feature columns and store the scalers in a data structure to allow future access to these scalers.



```
# ------------------------------------------------------------------------------------------------#
## e, scale the feature columns and store the scalers in data structure
if scale:
    column_scaler = {}
    # scale the data (prices) from 0 to 1
    for column in feature_columns:
        scaler = preprocessing.MinMaxScaler()
        df[column] = scaler.fit_transform(np.expand_dims(df[column].values, axis=1))
        column_scaler[column] = scaler
    # add the MinMaxScaler instances to the result returned
    result["column_scaler"] = column_scaler
# ------------------------------------------------------------------------------------------------#
```

Use column scaler in set variable type. After that, we need to scale the data in feature column. Every column mention in the feature_column array will be fit_transform in the way that the value from dataframe column is able to expand dimension before return to the column_scaler dictionary. Finally, assign into result dictionary.