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

```
from sklearn.model_selection import train_test_split
from yahoo_fin import stock_info as si
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import os
```

There are 3 libraries that have to access the function, and four are used for importing document.

- Function with libraries
    - `train_test_split`: used for split randomly in train and test data

    - `stock_info`: get information about stock price
    - `MinMaxScaler`: Transform the features by scaling each feature to a specific range.
- Libraries
    - `pandas`: for reading csv file
    - `os`: access the specific directory

Remove only tensorflow seed as it unused

2. Write a function as requirements

2.1. `load_data()` with different parameters provided, such as start and end date, shuffling dataset in both training and testing, step on how to deal with NaN, chosen whether split by ratio or by date or randomly, feature columns scale data between 0 and 1.

```
def load_data(ticker, TEST_START='2020-01-01', TEST_END='2023-01-01', shuffle=True, dropNaN=True, split_by_ratio=True, split_by_date=False,
 split_date=None, test_size=0.2, feature_columns=["adjclose", "volume", "open", "high", "low"], scale=True):
    """
    Loads data from yahoo finance, then scaling, shuffle, and normalization.
    :param ticker: (str/pd.DataFrame), the ticker you want to load, like META, AAPL,...
    :param TEST_START: str, the start date of the test data (format: "YYYY-MM-DD")
    :param TEST_END: str, the end date of the test data (format: "YYYY-MM-DD")
    :param shuffle: bool, whether to shuffle the dataset (both training and testing), default is True
    :param dropNaN: bool, whether to drop NaN values, default is True
    :param split_by_ratio: bool, whether to split the data into training and testing by ratio of percent data, default is True
    :param split_by_date: bool, whether to split the data into training and testing by date, default is False
    :param split_date: str, the date to split the data into training and testing by date, default is None
    :param test_size: ratio of test data to train data, default is 0.2 (80% for training and 20% for testing)
    :param feature_columns: the list of features to feed into the model, default is everything grabbed from yahoo_fin
    :param scale: whether to scale prices between 0 and 1, default is True
    :return:
    """
```

3. 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"]
```

4. Assign the method with parameters fulfilled to variable

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

5. Show first 5 rows by using head method

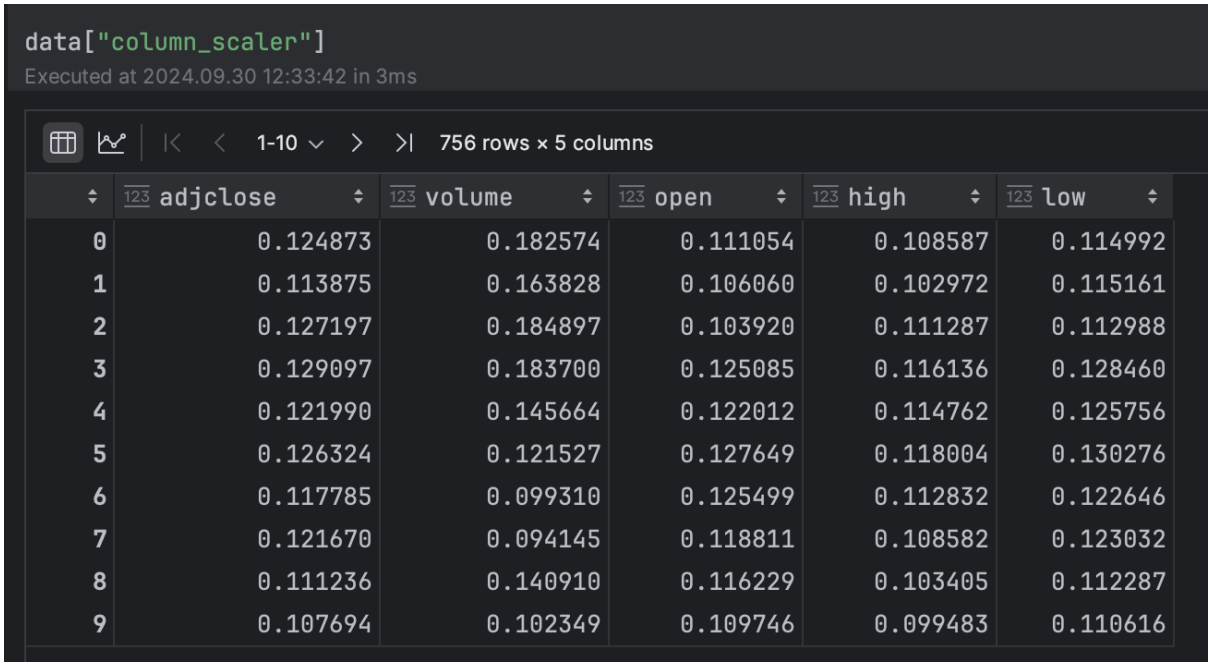
```python
data["df"].head()
```

Executed at 2024.09.30 12:33:42 in 3ms

5 rows × 7 columns

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

6. Show Feature scaling



```
data["column_scaler"]
```

Executed at 2024.09.30 12:33:42 in 3ms

1-10 756 rows × 5 columns

	adjclose	volume	open	high	low
0	0.124873	0.182574	0.111054	0.108587	0.114992
1	0.113875	0.163828	0.106060	0.102972	0.115161
2	0.127197	0.184897	0.103920	0.111287	0.112988
3	0.129097	0.183700	0.125085	0.116136	0.128460
4	0.121990	0.145664	0.122012	0.114762	0.125756
5	0.126324	0.121527	0.127649	0.118004	0.130276
6	0.117785	0.099310	0.125499	0.112832	0.122646
7	0.121670	0.094145	0.118811	0.108582	0.123032
8	0.111236	0.140910	0.116229	0.103405	0.112287
9	0.107694	0.102349	0.109746	0.099483	0.110616

Function Explain

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

```
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

```
# 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

```
# 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

```
# ------------------------------------------------------------
## b, Allow you to deal with the NaN issue in the data

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

If parameter decide dropNan is true, it will drop rows that contains NaN value. Otherwise it will fill NaN with mean value

```
# 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

```
# ------------------------------------------------------------------------------------------#
## 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
if split_by_ratio:
    # split the dataset into training & testing sets by ratio (not randomly splitting)
    train_samples = int((1 - test_size) * len(df))

    # :train_samples means from the beginning to train_samples, and train_samples: means from train_samples to the end
    result['df_train'] = df[:train_samples]
    result['df_test'] = df[train_samples:]

elif split_by_date and split_date is not None:
    # Convert split_date to a datetime object
    split_date = pd.to_datetime(split_date)

    # Split by date on the index
    result['df_train'] = df[df.index <= split_date]  # Take data from the start to split_date
    result['df_test'] = df[df.index > split_date]  # Take data after split_date to the end

else:
    # If neither split by ratio nor split by date, randomly split the dataset
    result['df_train'], result['df_test'] = train_test_split(df, test_size=test_size, shuffle=shuffle)
# ------------------------------------------------------------------------------------------#
```

Three way to split, split by ratio, split by date with date included, and split randomly

- Split by ratio: The different between split randomly and split by ratio is that the former one uses algorithm to get specific numeric samples for splitting the data, before split the train, test data based on the specific ratio numeric that has been calculated, the latter used the library method train_test_split for split randomly the data which indicate by the test size and we need to shuffle to make sure that there will be no bias in the data.
  For example for split by ratio: At dataframe, there are 756 rows, (1 – 0.2) means calculate the train_size, then 756 * 0.8 = 604.8, however, we only use integer, which means that the result is 604 rows.
- Split by date and mention by specific date: Split the data between train and test, in the train data, we take data from the start till the split date mentioned, and the test data is taken after split date to the end.

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:
    result['column_scaler'] = df[feature_columns]
    scale_min_max = MinMaxScaler(feature_range=(0, 1))
    result['column_scaler'] = scale_min_max.fit_transform(result['column_scaler'])
    result['column_scaler'] = pd.DataFrame(result['column_scaler'], columns = feature_columns)
# ------------------------------------------------------------------------------------------------------#
```

By scaling data using MinMaxScaler with range of 0 and 1. We need to transform the data frame into the range of MinMaxScaler that we mentioned before. After that, put the data that has been transformed into the dataframe with feature columns that we specified at parameters.