TASK 2 REPORT

COS30018

INTELLIGENT SYSTEMS

Halim Vlahos (104015943)

FUNCTION CODE IMPLEMENTATION SUMMARY

A: This function will allow you to specify the start date and the end date for the whole dataset as inputs.

This was implemented here, giving a parameter for both start and end date for the full dataset:

```
def load_process_dataset(company = COMPANY, start_date = '2020-01-01', end_date = '2024-07-02', split = '2023-08-01', save_path = './data/'):
```

B: This function will allow you to deal with the NaN issue in the data.

This was implemented here, using dropna (from pandas.DataFrame.dropna) which excludes empty values in the dataset. Upon research, it seems there were other methods for dealing with NaNs, such as mean (replace missing value with mean) and fills.

```
#This line is used to deal with NaNs; drops all missing values from dataset
full_data.dropna(inplace=True)
```

C: This function will also allow you to 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.

One of the function parameters I added was called ‘split’; and depending on the format in which the split string parameter is written, the function will use a different method to split the train / test data. If the split parameter passes the YYYY-MM-DD format, it will use the Date method and split at the point of the split date located in the data.

If split is in a decimal (0.##) format, it will use the ratio method, and split the full dataset by a percentage ratio into the train and test data sets.

```
#Check split type
regexDate = r"^\d{4}-\d{2}-\d{2}$"
regexRatio = r"^0\.\d+$"
if (re.search(regexDate, split)):    #Check split type = Date YYYY-MM-DD
    split_index = full_data.index.get_loc(pd.to_datetime(split))  #set index of split to the location of the split date.
    train_data = full_data[:split_index] #values preceeding split are for training and those after are for testing
    test_data = full_data[split_index:]
elif(re.search(regexRatio, split)):  #Check split type = Ratio 0.#
    split_index = int(len(full_data) * split)  #set index of split to a ratio of the data set
    train_data = full_data[:split_index]#values preceeding split are for training and those after are for testing
    test_data = full_data[split_index:]
```

D: This function will have the option to allow you to store the downloaded data on your local machine for future uses and to load the data locally to save time.

One of the function parameters I added was save_path. The function saves and loads in the following way:

First, it checks if the save directory exists at the save_path. If not, it creates the directory.

The file name is formatted as company_startdate_enddate.csv.

If such a file exists, it will load the data from that local csv file.

If not, it will download the data from yahoo finance, and then create the csv file.

```
#Ensure save directory exists, if not make one
if not os.path.exists(save_path):
    os.makedirs(save_path)

#file format for full_data = savepath/company_startdate_enddate.csv
file_path = os.path.join(save_path, f"{company}_{start_date}_{end_date}.csv")

# If a file for the current parameter data exists, load from the local file
if os.path.exists(file_path):
    print(f"Loading data from {file_path}")
    full_data = pd.read_csv(file_path, index_col="Date", parse_dates=True)
else:
    #if no local file exists, download from yf, and create local file for future reference
    print(f"Downloading data for {company} from {start_date} to {end_date}")

    # Get the data for the stock AAPL
    full_data = yf.download(company,start_date,end_date)

    full_data.to_csv(file_path)
    print(f"Data saved to {file_path}")
```

E: This function will also allow you to have an option to scale your feature columns and store the scalers in a data structure to allow future access to these scalers.

Last, I added a scale Boolean as a parameter; if it is set to True, it will use a MinMaxScaler and fit_transform to compress the data to values between 0 and 1.

```
def load_process_dataset(company = COMPANY, start_date = '2020-01-01', end_date = '2024-07-02', split = '2023-08-01', save_path = './data/', scale = True):
    #-------------------------------------------------------------------------
    # Prepare Data
    ## To do:
    # 1) Check if data has been prepared before.
    # If so, load the saved data
    # If not, save the data into a directory
    # 2) Use a different price value eg. mid-point of Open & Close
    # 3) Change the Prediction days
    #-------------------------------------------------------------------------
    if (scale):
        scaler = MinMaxScaler(feature_range=(0, 1))
    # Note that, by default, feature_range=(0, 1). Thus, if you want a different
    # feature_range (min,max) then you'll need to specify it here
        scaled_data = scaler.fit_transform(full_data[PRICE_VALUE].values.reshape(-1, 1))
    # Flatten and normalise the data
    # First, we reshape a 1D array(n) to 2D array(n,1)
    # We have to do that because sklearn.preprocessing.fit_transform()
    # requires a 2D array
    # Here n == len(scaled_data)
    # Then, we scale the whole array to the range (0,1)
    # The parameter -1 allows (np.)reshape to figure out the array size n automatically
    # values.reshape(-1, 1)
    # https://stackoverflow.com/questions/18691084/what-does-1-mean-in-numpy-reshape'
    # When reshaping an array, the new shape must contain the same number of elements
    # as the old shape, meaning the products of the two shapes' dimensions must be equal.
    # When using a -1, the dimension corresponding to the -1 will be the product of
    # the dimensions of the original array divided by the product of the dimensions
    # given to reshape so as to maintain the same number of elements.
```