Task 2 Report

NACA HITCHMAN - 103072170

Function Setup

I began by creating the declaration for my function, as well as providing descriptions for each parameter input.

```
#task 2 - function to load and process a dataset with multiple features
def processData(
    ticker,
    start_date,
    end_date,
    save_file,
    prediction_column,
    prediction_days,
    feature_columns=[],
    split_method='date',
    split_ratio=0.8,
    split_date=None,
    fillna_method='drop',
    scale_features=False,
    scale_min=0,
    scale_max=1,
    save_scalers=False):
    """
    Load and process a dataset with multiple features.

    :param ticker: str, company ticker symbol
    :param start_date: str, start date for the dataset in the format 'YYYY-MM-DD'
    :param end_date: str, end date for the dataset in the format 'YYYY-MM-DD'
    :param save_file: bool, whether to save the dataset to a file
    :param prediction_days: int, number of days to predict into the future
    :param feature_columns: list, list of feature columns to use in the model
    :param split_method: str, method to split the data into train/test data ('date' or 'random')
    :param split_ratio: float, ratio of train/test data if split_method is 'random'
    :param split_date: str, date to split the data if split_method is 'date'
    :param fillna_method: str, method to drop or fill NaN values in the data ('drop', 'ffill', 'bfill', or 'mean')
    :param scale_features: bool, whether to scale the feature columns
    :param scale_min: int, minimum value to scale the feature columns
    :param scale_max: int, maximum value to scale the feature columns
    :param save_scalers: bool, whether to save the scalers to a file

    :return: tuple of pandas.DataFrame, train and test data
    """
```

Then, I set up the system to check if the dataset file already exists, or if the ticker passed was an existing data frame. If the file doesn’t exist, the data is downloaded from yahoo finance and saved to a file using the .to_csv() method. If the file does exist, it is read to a data frame. A result dictionary is also created, which will store all the relevant data to be returned.

```
#create data folder in working directory if it doesnt already exist
data_dir = os.path.join(os.getcwd(), 'data')
if not os.path.exists(data_dir):
    os.makedirs(data_dir)
data = None
#if ticker is a string, load it from yfinance library
if isinstance(ticker, str):
    # Check if data file exists based on ticker, start_date and end_date
    file_path = os.path.join(data_dir, f"{ticker}_{start_date}_{end_date}.csv")
    if os.path.exists(file_path):
        # Load data from file
        data = pd.read_csv(file_path)
    else:
        # Download data using yfinance
        data = yf.download(ticker, start=start_date, end=end_date, progress=False)
        # Save data to file if boolean save_file is True
        if save_file:
            data.to_csv(file_path)
#if passed in ticker is a dataframe, use it directly
elif isinstance(ticker, pd.DataFrame):
    # already loaded, use it directly
    data = ticker
else:
    # raise error if ticker is neither a string nor a dataframe
    raise TypeError("ticker can be either a str or a `pd.DataFrame` instances")

# this will contain all the elements we want to return from this function
result = {}
# we will also return the original dataframe itself
result['df'] = data.copy()
```

Next, we check if the feature columns passed are in the data, then either drop or fill the nans based on the method passed as a parameter. Dropna is the default option, removing nan rows altogether from the dataset. The various fills use different methods to replace the nan values instead.

```
# make sure that the passed feature_columns exist in the dataframe
if len(feature_columns) > 0:
    for col in feature_columns:
        assert col in data.columns, f"'{col}' does not exist in the dataframe."
else:
    # if no feature_columns are passed, use all columns except the prediction_column
    feature_columns = list(filter(lambda column: column != 'Date', data.columns))

# add feature columns to result
result['feature_columns'] = feature_columns
# Deal with potential NaN values in the data
# Drop NaN values
if fillna_method == 'drop':
   data.dropna(inplace=True)
#use forward fill method, fill NaN values with the previous value
elif fillna_method == 'ffill':
    data.fillna(method='ffill', inplace=True)
#use backward fill method, fill NaN values with the next value
elif fillna_method == 'bfill':
    data.fillna(method='bfill', inplace=True)
#use mean method, fill NaN values with the mean of the column
elif fillna_method == 'mean':
    data.fillna(data.mean(), inplace=True)
```

The dataset is then split into train and test, either by date if the split method is date, or randomly with a ratio. As data is a dataframe, the .loc() method can be used to compare the value of the 'Date' column, and compare it to the passed split date.

Both split date and split ratio are passed as parameters. Next, the datasets indexes and sorting are reset to ensure they are in the correct order.

```
# Split data into train and test sets based on date
if split_method == 'date':
    train_data = data.loc[data['Date'] < split_date]
    test_data = data.loc[data['Date'] >= split_date]
# Split data into train and test sets randomly with provided ratio
elif split_method == 'random':
    train_data, test_data = train_test_split(data, train_size=split_ratio, random_state=42)

# Reset index of both dataframes
train_data = train_data.reset_index()
test_data = test_data.reset_index()
# Sort dataframes by date
train_data = train_data.sort_values(by='Date')
test_data = test_data.sort_values(by='Date')
```

Next, if the specified, the data will be scaled. The min and max scale properties can be passed to the function. I've looped through each feature column passed, where a new min max scaler is created with the passed min and max values. Scaler.fit_transform is used for the training data, to fit the scaler, and standard transform is used for the test data, to ensure the test data is scaled based on the scaling on the train data. Further, the values passed to the scaler for each column and reshaped to ensure it will be compatible.

The scalers are then saved to a text file if specified using pickle, then the scaled data is converted back into data frames and saved to the result dictionary.

```python
# Scale features
if scale_features:
    # Create scaler dictionary to store all scalers for each feature column
    scaler_dict = {}
    # Dictionaries to store scaled train and test data
    scaled_train_data = {}
    scaled_test_data = {}
    #loop through each feature column
    for col in feature_columns:
        # Create scaler for each feature column using Min Max, passing in the scale_min and scale_max
        scaler = MinMaxScaler(feature_range=(scale_min, scale_max))
        # Fit and transform scaler on train data
        scaled_train_data[col] = scaler.fit_transform(train_data[col].values.reshape(-1, 1)).ravel()
        # Transform test data using scaler
        scaled_test_data[col] = scaler.transform(test_data[col].values.reshape(-1,1)).ravel()
        # Add scaler to scaler dictionary, using the feature column name as key
        scaler_dict[col] = scaler
    # Add scaler dictionary to result
    result["column_scaler"] = scaler_dict

     # Save scalers to file
    if save_scalers:
        # Create scalers directory if it doesn't exist
        scalers_dir = os.path.join(os.getcwd(), 'scalers')
        if not os.path.exists(scalers_dir):
            os.makedirs(scalers_dir)
        # Create scaler file name
        scaler_file_name = f"{ticker}_{start_date}_{end_date}_scalers.txt"
        scaler_file_path = os.path.join(scalers_dir, scaler_file_name)
        with open(scaler_file_path, 'wb') as f:
            pickle.dump(scaler_dict, f)

    # Convert scaled data to dataframes
    train_data = pd.DataFrame(scaled_train_data)
    test_data = pd.DataFrame(scaled_test_data)

# Add train and test data to result
result["scaled_train"] = train_data
result["scaled_test"] = test_data
```

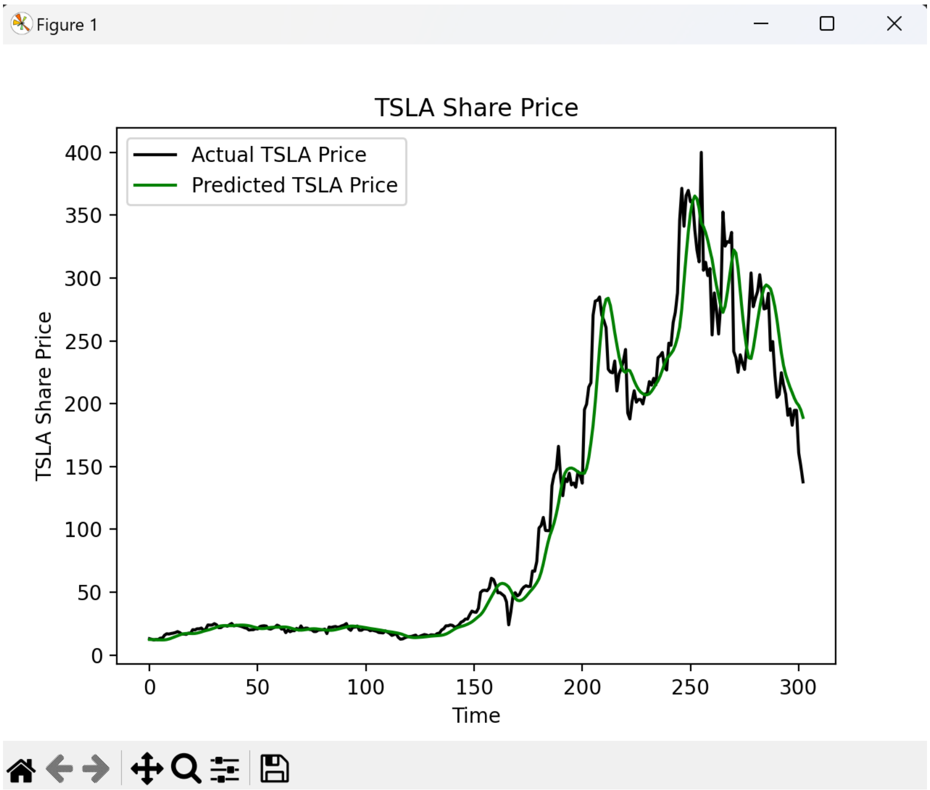
Lastly, the scaled data is split into their respective X and y arrays. This is also based on the prediction days as the original base is, and then saved into the results dictionary, before the result is returned.

```
# Construct the X's and y's for the training data
X_train, y_train = [], []
# Loop through the training data from prediction_days to the end
for x in range(prediction_days, len(train_data)):
    # Append the values of the passed prediction column to X_train and y_train
    X_train.append(train_data[prediction_column].iloc[x-prediction_days:x])
    y_train.append(train_data[prediction_column].iloc[x])

# convert to numpy arrays
result["X_train"] = np.array(X_train)
result["y_train"] = np.array(y_train)
# reshape X_train for proper fitting into LSTM model
result["X_train"] = np.reshape(result["X_train"], (result["X_train"].shape[0], result['X_train'].shape[1], -1));
# construct the X's and y's for the test data
X_test, y_test = [], []
# Loop through the test data from prediction_days to the end
for x in range(prediction_days, len(test_data)):
    # Append the values of the passed prediction column to X_test and y_test
    X_test.append(test_data[prediction_column].iloc[x - prediction_days:x])
    y_test.append(test_data[prediction_column].iloc[x])

# convert to numpy arrays
X_test = np.array(X_test)
y_test = np.array(y_test)
#assign y_test to result
result["y_test"] = y_test
#assign X_test to result and reshape X_test for prediction compatibility
result["X_test"] = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1));

return result
```

Within the main method now, the parameters to be passed are defined, and sent to the function. The results are saved into 'data' and used to train the model and predict the prices as the original base did.

```
# define function parameters to use
DATA_SOURCE = "yahoo"
COMPANY = "TSLA"
DATA_START_DATE = '2015-01-01'
DATA_END_DATE = '2022-12-31'
SAVE_FILE = True
PREDICTION_DAYS = 100
SPLIT_METHOD = 'random'
SPLIT_RATIO = 0.8
SPLIT_DATE = '2020-01-02'
NAN_METHOD = 'drop'
FEATURE_COLUMNS = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
SCALE_FEATURES = True
SCALE_MIN = 0
SCALE_MAX = 1
SAVE_SCALERS = True
prediction_column = "Close"

# Call processData function passing in parameters
data = processData(
    ticker=COMPANY,
    start_date=DATA_START_DATE,
    end_date=DATA_END_DATE,
    save_file=SAVE_FILE,
    prediction_column=prediction_column,
    prediction_days=PREDICTION_DAYS,
    split_method=SPLIT_METHOD,
    split_ratio=SPLIT_RATIO,
    split_date=SPLIT_DATE,
    fillna_method=NAN_METHOD,
    feature_columns=FEATURE_COLUMNS,
    scale_features=SCALE_FEATURES,
    scale_min=SCALE_MIN,
    scale_max=SCALE_MAX,
    save_scalers=SAVE_SCALERS
    )
```

We can see this provides valid prediction results, with the next day being predicted.




```
Epoch 25/25
48/48 [==============================] - 13s 262ms/step - loss: 0.0016
10/10 [==============================] - 3s 53ms/step
1/1 [==============================] - 0s 97ms/step
Prediction:  [156.92934]
```