Task 2 Report

Most of the code is straightforward there are only a few which are less straightforward in my opinion, those I will explain here better based on my understandings.

```
# Add date column if it does not exist, for some reason there is already a date column, but the format is weird.
if "Date" not in data.columns:
    data["Date"] = data.index
if scale:
    column_scaler = {}
    # scale the data (prices) from 0 to 1
    for column in feature_columns:
        data[column] = scaler.fit_transform(np.expand_dims(data[column].values, axis=1))
        column_scaler[column] = scaler
```

In this section of the code there is a line:

```
data[column] = scaler.fit_transform(np.expand_dims(data[column].values, axis=1))
```

This section transforms or scales the data of a column and iterates through all the columns specified in “feature_columns” which is ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']. Using a scaler which was specified in an earlier section of the code “MinMaxScaler”, to scale the data into a float range in between 0 and 1 so that the data can be used for comparison by the model further down the road.

```
# adds a future column in the dataframe, the .shift(-lookup_step)
# shifts the values of the columns up. specified by -lookup_step
data['Future'] = data['Adj Close'].shift(-lookup_step)

# copies the last lookup step rows and saves it in last_sequence.
last_sequence = np.array(data[feature_columns].tail(lookup_step))

# drops rows which has NaN in their row.
data.dropna(inplace=True)
```

This is another section of code, this section basically as the comments suggests, adds a column named “Future” to the data frame. The data here will be taken from the “Adj Close” column and shifted upwards by the integer value stored in “lookup_step”, now the value is set to 1 so the data is copied over to “Future” and shifted up by one. Because the value is shifted up, there will be 1 row at the very bottom in the Future column that has a NaN value. This row is then deleted by the last line in the screenshot above.

```
# initialize sequence_data list
sequence_data = []
# initialize sequences double ended list which has maximum length of n_steps
# if maximum length is reached, the oldest data will be overwritten by new data
sequences = deque(maxlen=n_steps)
# entry is the combination of feature_columnns and the dates
# target is the future values.
for entry, target in zip(data[feature_columns + ["Date"]].values, data['Future'].values):
    sequences.append(entry)
    # if length of sequences has reached n_steps it will append array of sequences with target to sequence_data.
    # this is so that the sequence_data can show the data in which, X amount of sequences will result in the
    # target value to be achieved. To be used to train the model later on. In the current case it will use
    # the past 50 days of data to determine the future value.
    if len(sequences) == n_steps:
        sequence_data.append([np.array(sequences), target])

# list object
# combines the feature columns sequences with last_sequence and creates a list
last_sequence = list([s[:len(feature_columns)] for s in sequences]) + list(last_sequence)
# the list is then converted to a numpy array with the type float32
last_sequence = np.array(last_sequence).astype(np.float32)
# save the array into the result dictionary as 'last_sequence'
result['last_sequence'] = last_sequence
```

To summarize this section of code, two variables are initialized one a list and another a double ended queue respectively. A for loop is then entered iterating through an object created by the zip keyword. The object has two sections the first half being the “entry” data in the “feature_columns” plus the date and the second half being the “target” data which is the “Future” column. What the for loop does essentially gets 50 (set by an integer value stored in n_steps) days of historical data and then connects that with the data stored in the “Future” column on the 50th day and storing it in “sequence_data”. Using this “sequence_data” the model can learn and try and find a pattern to help predict the future stock price movements.

Then an additional part, “last_sequence” which is the final row which was removed from the data frame after the creation of the Future column is updated to include the 50 days prior to the “last_sequence” which hosts the Future column which stores the NaN data.

The rest of the code is pretty straight forward, and I have written comments as well.