Configuration Manual

MSc Research Project
MSc in Data Analytics

Arul Selvam Perumal
Student ID:x21128847

School of Computing
National College of Ireland

Supervisor: Prof. Athanasios Staikopoulos

Configuration Manual
Project Submission Sheet
School of Computing

Student Name:	Arul Selvam Perumal
Student ID:	x21128847
Programme:	MSc in Data Analytics
Year:	2022-2023
Module:	Research Project
Supervisor:	Prof. Athanasios Staikopoulos
Submission Due Date:	14th August 2023
Project Title:	Walmart Sales forecasting using Equilibrium optimized Deep LSTM
Word Count:	304
Page Count:	6

Signature:	Arul Selvam Perumal
Date:	14 th August 2023

Configuration Manual

Arul Selvam Perumal
x21128847

Appendix:

Introduction This setup manual provides a detailed outline to replicate the planned study and achieve the desired outcomes. It includes step-by-step procedures and code snippets for implementing the approaches, along with instructions and code snippets for testing and evaluating the model.

1 System configuration:

The minimum requirements for software and hardware tools to conduct this research are stated ad follows:

(a) Hardware requirements:

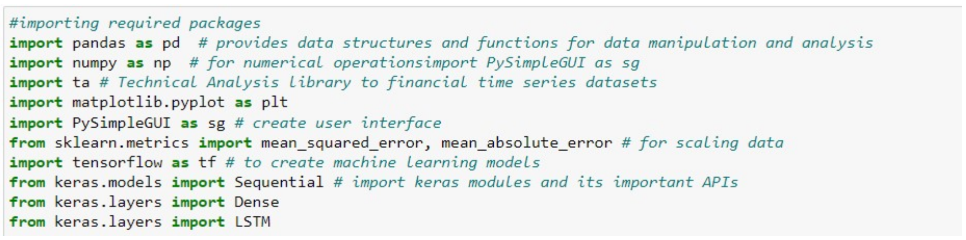
- Processor: Apple M1 chip
- RAM: 8 GB Unified Memory
- ROM: 256 GB SSD Storage

(b) Software Requirements:

- Operating System: macOS Ventura
- Technology: Python
- IDE: Jupyter Notebook

The libraries used are stated as follows:

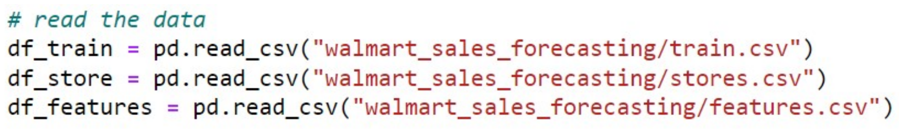
- Numpy
- Pandas
- Keras
- Math
- Matplotlib
- tensorflow
- PySimpleGUI



```
#importing required packages
import pandas as pd  # provides data structures and functions for data manipulation and analysis
import numpy as np  # for numerical operationsimport PySimpleGUI as sg
import ta # Technical Analysis library to financial time series datasets
import matplotlib.pyplot as plt
import PySimpleGUI as sg # create user interface
from sklearn.metrics import mean_squared_error, mean_absolute_error # for scaling data
import tensorflow as tf # to create machine learning models
from keras.models import Sequential # import keras modules and its important APIs
from keras.layers import Dense
from keras.layers import LSTM
```

Figure 1: Packages

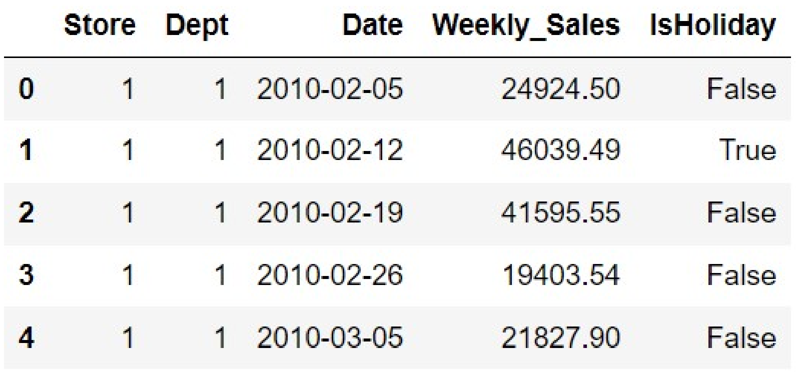
Figure 1 depicts the packages imported. Here, the data structures and functions for data manipulation and analysis are provided by the pandas. The NumPy is used for numerical operations. The ta is utilized for the technical analysis library to financial time series datasets. Then, PySimple is used to create the user interface. The matplotlib is used for the visualization and the standard scalar is used for scaling the data. TensorFlow is utilized to create the machine learning models. Keras. models used for importing the Keras modules and their important APIs.



```
# read the data
df_train = pd.read_csv("walmart_sales_forecasting/train.csv")
df_store = pd.read_csv("walmart_sales_forecasting/stores.csv")
df_features = pd.read_csv("walmart_sales_forecasting/features.csv")
```

Figure 2: Read the CSV file

Figure 2 demonstrates reading the CSV file regarding three data files, such as train, stores, and features, from the dataset.



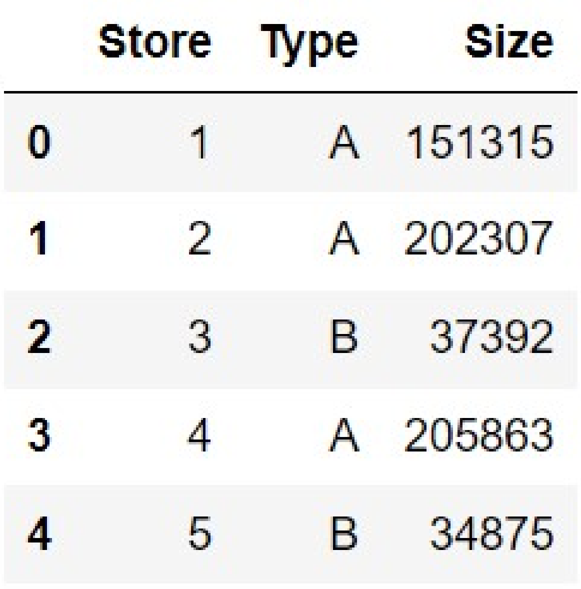
	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

Figure 3: Read the “train” data

Figure 3 demonstrates reading the data “train”. Here, the features, like date, weekly sales, and IsHoliday regarding the train data are considered.

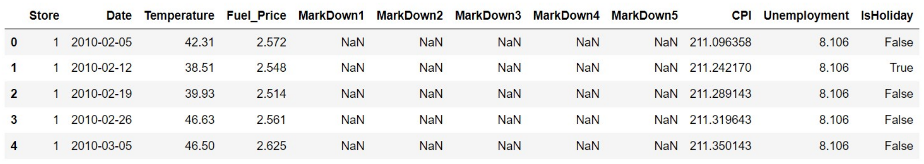
Figure 4 illustrates reading the “store” data that considers store type and store size as its features.

Figure 5 demonstrates reading the feature data having store, data temperature, CPI, unemployment list, and IsHoliday as its features.



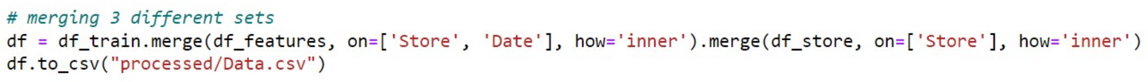
	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

Figure 4: Read the “store” data



	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

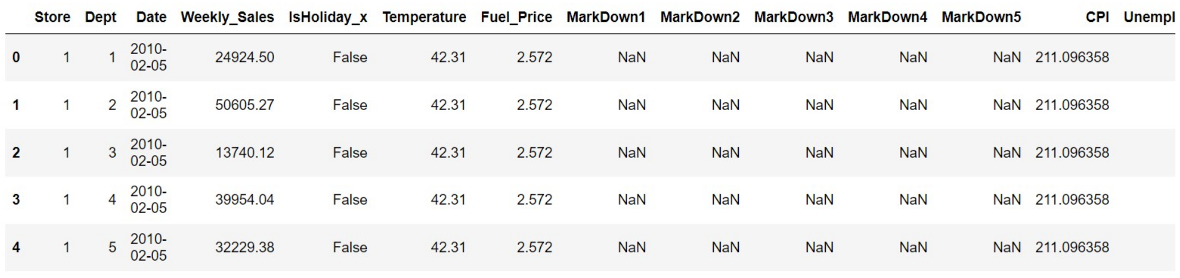
Figure 5: Read the “feature” data



```
# merging 3 different sets
df = df_train.merge(df_features, on=['Store', 'Date'], how='inner').merge(df_store, on=['Store'], how='inner')
df.to_csv("processed/Data.csv")
```

Figure 6: Merging 3 different sets

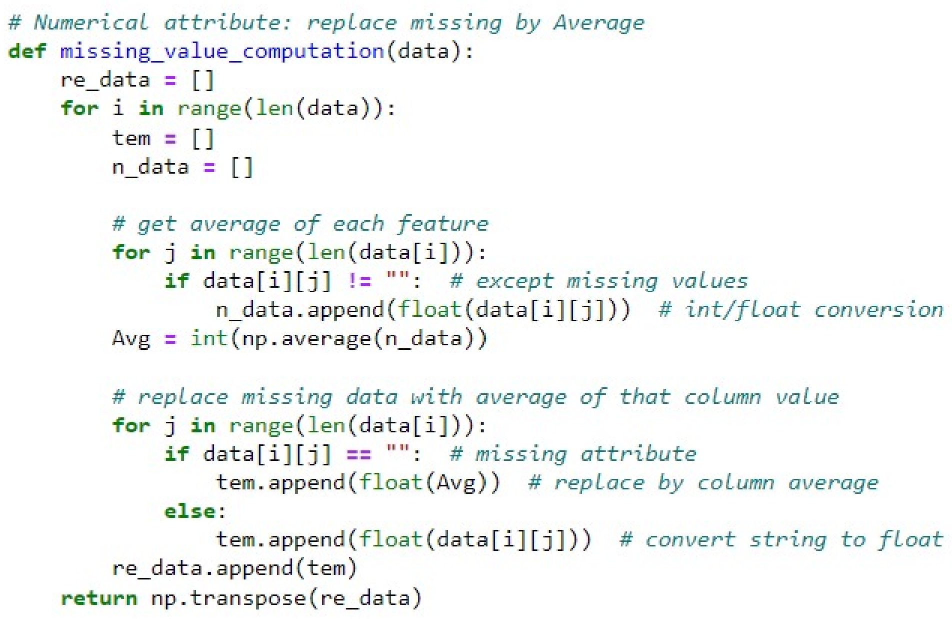
Figure 6 demonstrates the coding for merging three different sets, store, feature, and train data.



	Store	Dept	Date	Weekly_Sales	IsHoliday_x	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unempl
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	
1	1	2	2010-02-05	50605.27	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	
2	1	3	2010-02-05	13740.12	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	
3	1	4	2010-02-05	39954.04	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	
4	1	5	2010-02-05	32229.38	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	

Figure 7: Merged data

Figure 7 demonstrates the merged data from "train" "store" and "feature" with the features as store, dept, date, IsHoliday, type, and size.



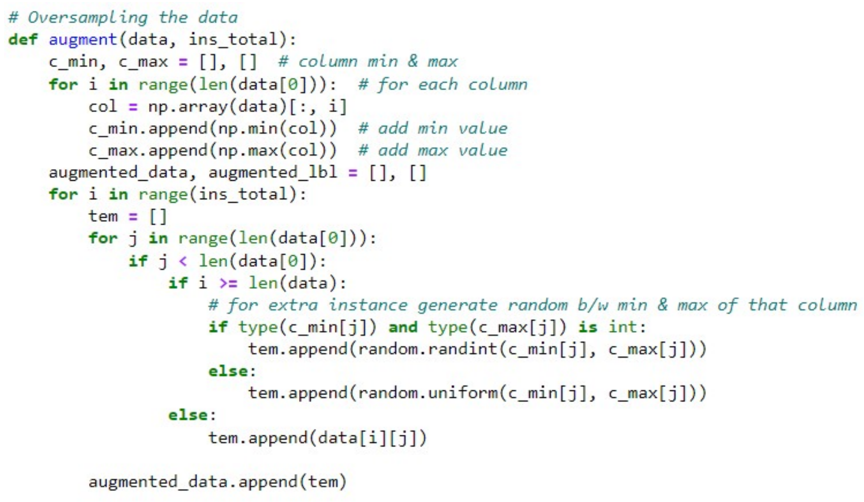
```
# Numerical attribute: replace missing by Average
def missing_value_computation(data):
    re_data = []
    for i in range(len(data)):
        tem = []
        n_data = []

        # get average of each feature
        for j in range(len(data[i])):
            if data[i][j] != "":  # except missing values
                n_data.append(float(data[i][j]))  # int/float conversion
        Avg = int(np.average(n_data))

        # replace missing data with average of that column value
        for j in range(len(data[i])):
            if data[i][j] == "":  # missing attribute
                tem.append(float(Avg))  # replace by column average
            else:
                tem.append(float(data[i][j]))  # convert string to float
        re_data.append(tem)
    return np.transpose(re_data)
```

Figure 8: Preprocessing using missing data imputation

Figure 8 depicts the preprocessing using missing data imputation. The preprocessing involves handling missing data through an approach called missing data imputation. Also, missing data are imputed by replacing them with the average value of the respective column.

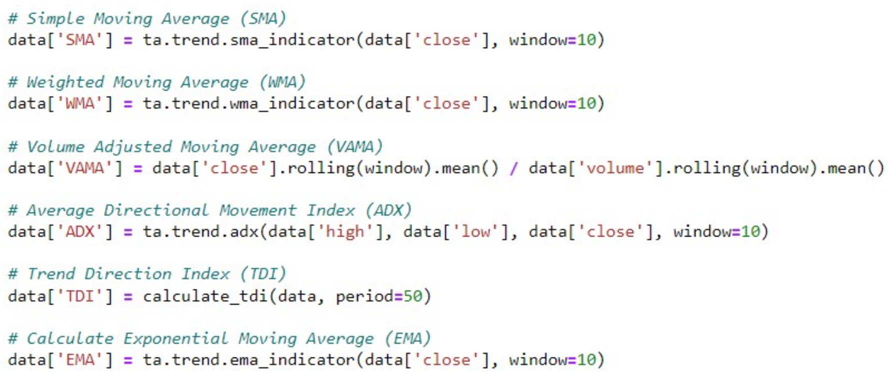
Figure 9 illustrates the data oversampling by the data augmentation. Initially, the size of the data is 50,000. The minimum and maximum values for each column in the dataset are determined. Then, the minimum and maximum values are added as additional instances to the dataset. By appending the added minimum and maximum values, the size of the dataset is increased.

Figure 10 demonstrates the computation of technical indicators. Here, the indicators, like SMA, WMA, VAMA, ADX, TDI, and EMA model are calculated.



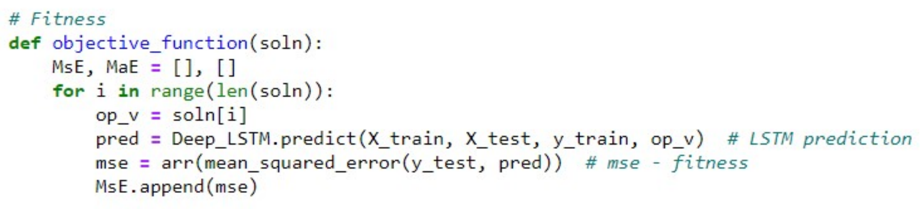
```python
# Oversampling the data
def augment(data, ins_total):
    c_min, c_max = [], []  # column min & max
    for i in range(len(data[0])):  # for each column
        col = np.array(data)[:, i]
        c_min.append(np.min(col))  # add min value
        c_max.append(np.max(col))  # add max value
    augmented_data, augmented_lbl = [], []
    for i in range(ins_total):
        tem = []
        for j in range(len(data[0])):
            if j < len(data[0]):
                if i >= len(data):
                    # for extra instance generate random b/w min & max of that column
                    if type(c_min[j]) and type(c_max[j]) is int:
                        tem.append(random.randint(c_min[j], c_max[j]))
                    else:
                        tem.append(random.uniform(c_min[j], c_max[j]))
                else:
                    tem.append(data[i][j])

        augmented_data.append(tem)
```

Figure 9: Data oversampling



```python
# Simple Moving Average (SMA)
data['SMA'] = ta.trend.sma_indicator(data['close'], window=10)

# Weighted Moving Average (WMA)
data['WMA'] = ta.trend.wma_indicator(data['close'], window=10)

# Volume Adjusted Moving Average (VAMA)
data['VAMA'] = data['close'].rolling(window).mean() / data['volume'].rolling(window).mean()

# Average Directional Movement Index (ADX)
data['ADX'] = ta.trend.adx(data['high'], data['low'], data['close'], window=10)

# Trend Direction Index (TDI)
data['TDI'] = calculate_tdi(data, period=50)

# Calculate Exponential Moving Average (EMA)
data['EMA'] = ta.trend.ema_indicator(data['close'], window=10)
```

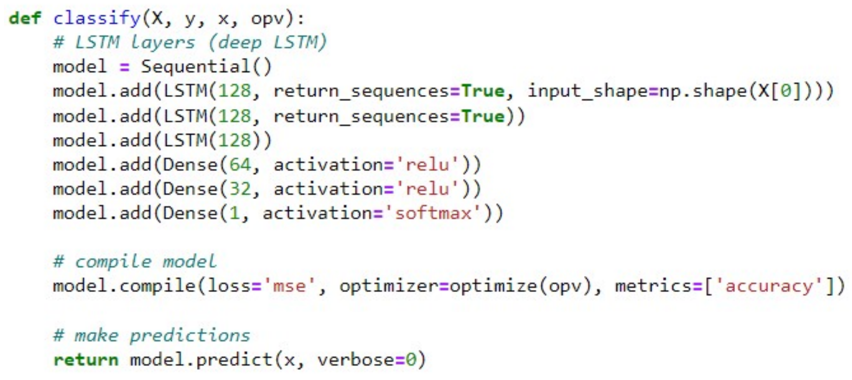
Figure 10: Technical Indicator computation



```python
# Fitness
def objective_function(soln):
    MsE, MaE = [], []
    for i in range(len(soln)):
        op_v = soln[i]
        pred = Deep_LSTM.predict(X_train, X_test, y_train, op_v)  # LSTM prediction
        mse = arr(mean_squared_error(y_test, pred))  # mse - fitness
        MsE.append(mse)
```

Figure 11: Optimization objective function

Figure 11 illustrates the coding for optimization objective function. The main aim of this function is to train the deep LSTM model.



```
def classify(X, y, x, opv):
    # LSTM layers (deep LSTM)
    model = Sequential()
    model.add(LSTM(128, return_sequences=True, input_shape=np.shape(X[0])))
    model.add(LSTM(128, return_sequences=True))
    model.add(LSTM(128))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='softmax'))

    # compile model
    model.compile(loss='mse', optimizer=optimize(opv), metrics=['accuracy'])

    # make predictions
    return model.predict(x, verbose=0)
```

Figure 12: Deep LSTM

Figure 12 demonstrated the Deep LSTM model. In the deep LSTM model, the commonly used activation functions are ReLU (Rectified Linear Unit) and softmax.c