National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Lithin Dominic Joseph
Student ID: 20187963

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Lithin Dominic Joseph
Student ID:	20187963
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	799
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Lithin Dominic Joseph
Date:	30th January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Lithin Dominic Joseph
20187963

1 Introduction

This is the configuration manual describing guidelines to implement the research project 'Time Series Approaches to Predict Soccer Outcomes'. The hardware and software requirements are also specified in this report.

2 Hardware Configuration

The hardware requirements for the project is given below.

- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.81 GHz
- RAM : 8 GB
- Storage : 128 SSD, 1TB HDD
- OS : Windows 10 Home Single Language version 20H2

3 Software Configuration

For the implementation of the project following software are specifically used.

- Anaconda
- Jupyter Notebook v6.4.6
- Weka v3.8.5
- Microsoft Excel

4 Environment Setup

4.1 Jupter Notebook

This section will describe the steps to setup Jupyter Notebook. Anaconda is downloaded and installed in the system. A new enviroment is created and latest packages are installed. Jupyter Notebook and CMD prompt for Anaconda is also installed. Installed tools can be visible in Figure 1. Install Python version 3.9.5 on the environment created.

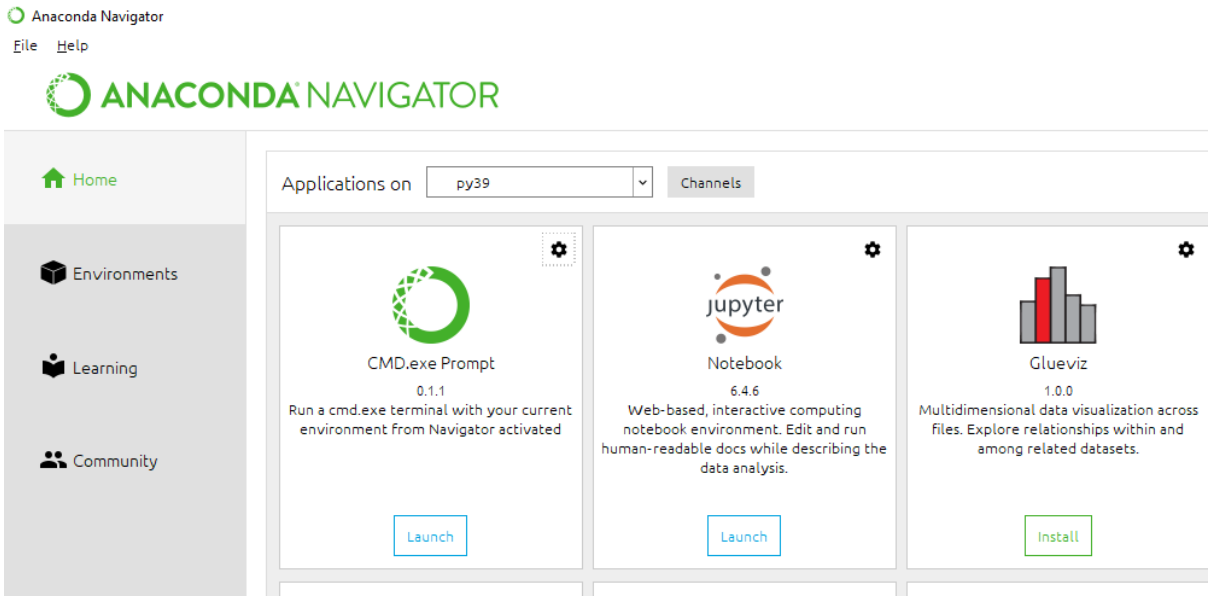

Figure 1: Anaconda Tool

4.2 Weka 3.8.5

Weka is a open source machine learning tool available online https://www.cs.waikato.ac.nz/ml/weka/ The installation file can be downloaded from the URL. Screenshot is given in Figure **??**

5 Data Selection

The dataset for the project is available on open website http://www.football-data.co.uk. Here all season-wise EPL games data are available to download. It is a csv file containing 360 entries. Website given in Figure 3

6 Implementation

This section will deal with implementation of the project work.

6.1 Weka Implementation

All the three seasons dataset is merged using Microsoft Excel and removed the unwanted fields. Then these csv data uploaded to the tool using Open File option in the interface. There is option to select and deselect data features in the system. All the required fields are selected.

In the Classify tab, there are options to choose algorithm, test options and can choose the prediction variable. This screen is visible in Figure 4. The output of the algorithm will be printed on the right side of the screen. Bayesian algorithms are available under the section Weka - classifiers - Bayes.

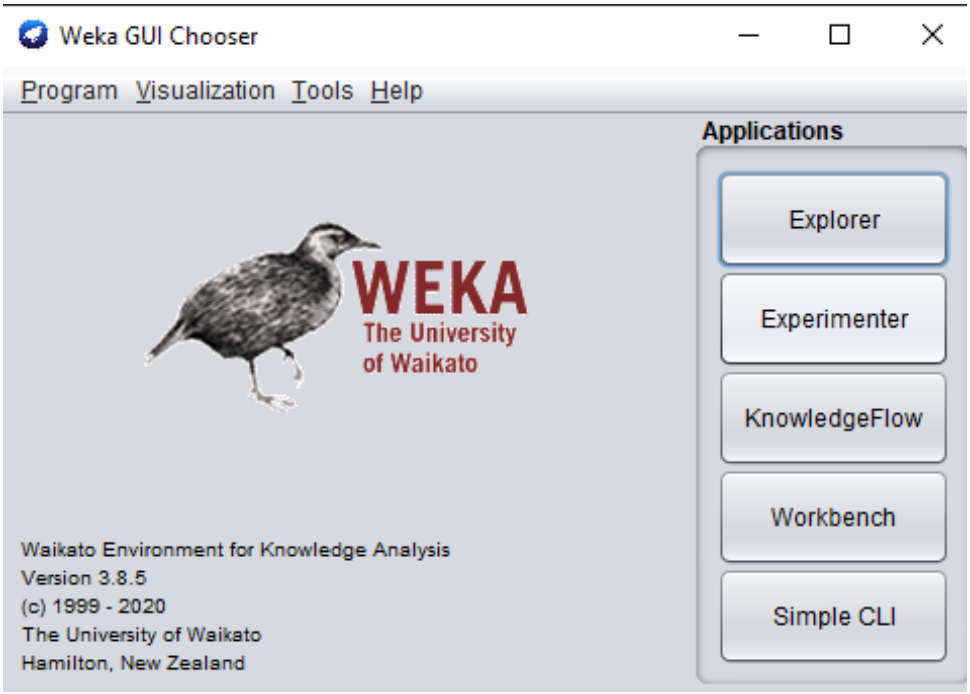

Figure 2: WEKA

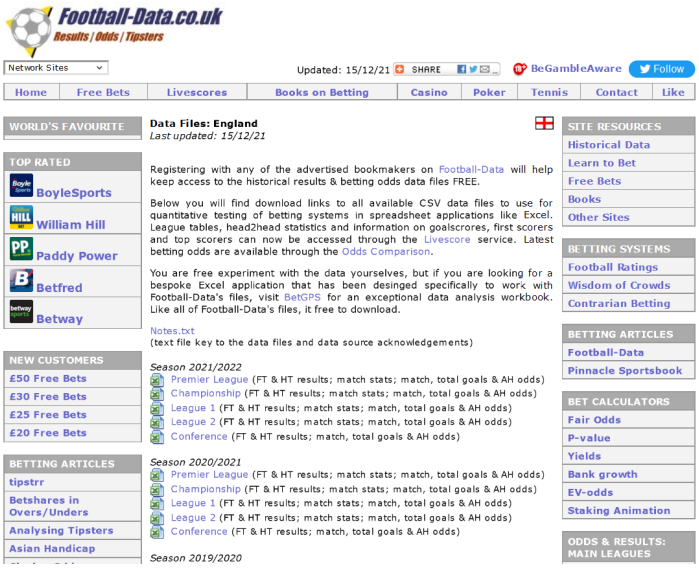

Figure 3: Data downloaded website

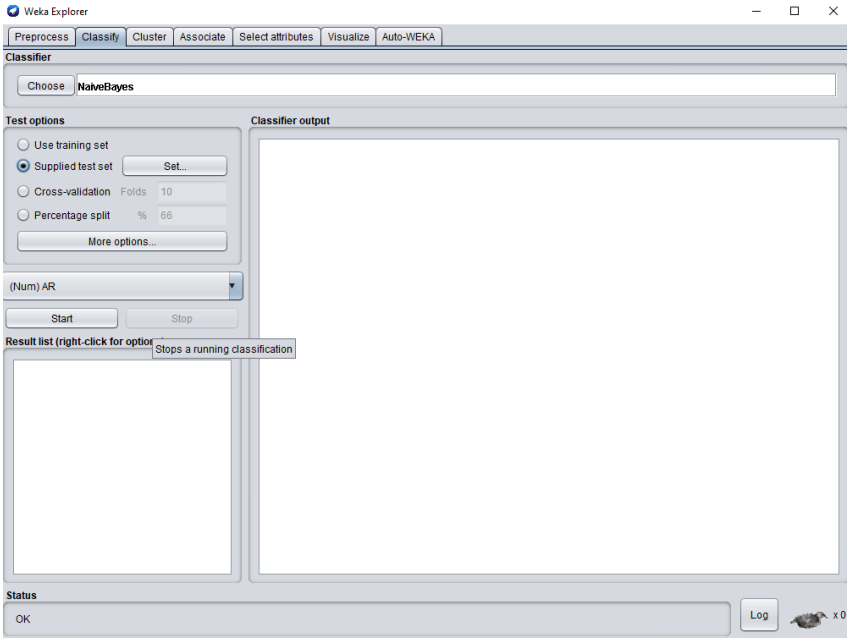

Figure 4: Weka Classification

6.2 Jupter Python Implementation

For the Python implementation part of the project following libraries are needed. figure-name 5

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import glob
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from statsmodels.tsa.api import SimpleExpSmoothing
from sklearn import linear_model, tree, ensemble
from sklearn.naive_bayes import BernoulliNB, GaussianNB,MultinomialNB
import plotly.express as px
import itertools
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
from keras.preprocessing.sequence import TimeseriesGenerator
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

Figure 5: Libarires Loaded

6.3 Bayesian Networks using Python

The three years dataset is saved in a location. The below Python snippet Figure 6 will all the CSV file and convert into a dataframe. Then the dataframe is cleaned using the below section Figure 7. First 23 fields are selected and the remaining fields discarded. fields like DIV and data are also removed.

Then, the predicting variable 'FTR' is removed from the training set. On this cleaned dataset, three bayesian networks are applied. The function defining code snippet is attached Figure 8.

```
# get data file names
def getData(path):
    filenames = glob.glob(path + "/*.csv")
    dfs = []
    for filename in filenames:
        dfs.append(pd.read_csv(filename))
    # Concatenate all data into one DataFrame
    EPL_data = pd.concat(dfs, ignore_index=True)
    return EPL_data
```

Figure 6: Function to load data

```
EPL_cleared =EPL_data.drop(EPL_data.iloc[:, 23:127], axis = 1)
EPL_tData = EPL_cleared
EPL_tData = EPL_tData.drop(['Div'], axis=1)
EPL_tData = EPL_tData.drop(['Date'], axis=1)
EPL_tData
```

Figure 7: Cleaning of the dataframe

```
def Bmodels(X, Y):
    res = dict()
    xtrain = pd.get_dummies(X)

    bnb = BernoulliNB()
    cross_val_score(bnb, xtrain, Y, cv=10)
    res['bernoulliNB_cvs'] = cross_val_score(bnb, xtrain, Y, cv=10).mean()

    gnb = GaussianNB()
    cross_val_score(gnb, xtrain, Y, cv=10)
    res['GaussianNB_cvs'] = cross_val_score(gnb, xtrain, Y, cv=10).mean()

    mnb = MultinomialNB()
    cross_val_score(mnb, xtrain, Y, cv=10)
    res['MultinomialNB_cvs'] = cross_val_score(mnb, xtrain, Y, cv=10).mean()
```

Figure 8: Cleaning of the dataframe

6.4

6.5 Time Series Based Models

Here the dataset contains all the past result of EPL. There are 27 CSV files in the directory. The same load function used in section 6.3 is used to load the all content into datframe. But additional parameters passed while read from CSV file to filter out the unwanted fields.

Then created list of important clubs in the series that played continuously in EPL and found out the possible pairs between them using Python combination function from library 'itertools'.

```
def makeTimeSeries(df, homeTeam = False):
    df['Won'] = -1
    df['Won'][df['FTR']== 'H'] = 1
    df['Won'][df['FTR']=='D'] = 0
    df['Won'] = df['Won'].astype(float)
    df['Datetime'] = pd.to_datetime(df['Date'])
    df = df.sort_values(by="Datetime")
    ts_series = df.set_index(pd.DatetimeIndex(df['Datetime']))
    ts_df = ts_series[['Won']]
    return ts_df
```

Figure 9: Creating Time Series

Then, the cleaned dataset is used to create the time series using the function 'makeTimeSeries'. The code is given below Figure 9. Then the moving average values are calculated using 'ewm' method. It is given in Figure 10.

```
def tsCalculate(df, homeTeam, awayTeam, span):
    ht_df = df.loc[(df['HomeTeam'] == homeTeam ) & (df['AwayTeam'] == awayTeam)]
    ht_ts = makeTimeSeries(ht_df)
    ht_ts['smooth'] = ht_ts['Won'].ewm(span = span).mean()
    pred =  round(ht_ts.smooth.iat[-1],4)
    return pred
```

executed in 6ms, finished 13:00:31 2021-12-14

Figure 10: Calculation of moving Average

6.6 Simple Sequential Model

The simple sequential model is created using TimeseriesGenerator method from Keras. The function 'fnTraining' define and predict the values for each pair of teams. Its depicted in Figure 11.

6.7 LSTM Model

The LSTM model is defined using Keras and Tenserflow libraries. A function is created to define the model and it is given in Figure 12.

6.8 Evaluation

A function 'getResultLSTM' is used to structure the output prediction from the loop to a dataframe Figure 13. This structured dataframe is used to calculate the absolute mean error and accuracy of the model. The predicted result is then rounded in to 1, 0 and -1. The code for this conversion is gven in Figure 14. Using the already created functions

```
def fnTraining(ts):
    features = ts['Won'].tolist()
    ts_gen = TimeseriesGenerator(features, features, batch_size=1, length=5 )
    # define model
    model = Sequential()
    model.add(Dense(100, activation='relu', input_dim=5))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')

    model.fit_generator(ts_gen, steps_per_epoch=1, epochs=200, verbose=0)
    x_input = array([features[-5],features[-4],features[-3],features[-2],features[-1]]).reshape((1,5))
    yhat = model.predict(x_input, verbose=0)
    return yhat
```

Figure 11: Function to define and predict simple sequential model

```
def fnTrainingLSTM(ts):
    features = ts['Won'].tolist()
    ts_gen = TimeseriesGenerator(features, features, batch_size=1, length=5 )
    #LSTMmodel
    model = Sequential()
    model.add(LSTM(100, activation='relu', input_shape=(5, 1)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    model.fit(ts_gen, steps_per_epoch=1, epochs=200, verbose=0)
    x_input = array([features[-5],features[-4],features[-3],features[-2],features[-1]]).reshape((1,5,1))
    yhat = model.predict(x_input, verbose=0)
    return yhat
```

Figure 12: Function to define LSTM model

Figure 15 , the accuracy of the prediction can be calculated. The 'mean_absolute_error' method from sklearn is used to calculate the mean absolute error.

Finally, The results of each rounding value is re-arranged and plotted on a graph. Also its value displayed. Results of LSTM model given in Figure 16

7 Other Tools

The online latex tool overleaf [1] is used to prepare report for the project. Figure **??**

[1] URL:https://www.overleaf.com

```
def getResultLSTM(teamCombList):
    HAlist = []
    reslt = []
    result = []
    hmTeam =[]
    awTeam = []
    for comb in teamCombList:
        HAlist.append(tsCalculateLSTM(epl_df, comb[0], comb[1], 4))
        reslt.append(checkresult(data2020, comb[0], comb[1]))
        hmTeam.append(comb[0])
        awTeam.append(comb[1])
        HAlist.append(tsCalculateLSTM(epl_df, comb[1], comb[0], 4))
        reslt.append(checkresult(data2020, comb[1], comb[0]))
        hmTeam.append(comb[1])
        awTeam.append(comb[0])
    # HAlist
    data = pd.DataFrame({'home': hmTeam, 'away':awTeam, 'pred': HAlist, 'result': reslt}, columns=['home', 'away','pred', 're
    return data
```

Figure 13: Function to arrange the results

```
####Rounding

LSTMmodel['rPred.5'] = 0
LSTMmodel.loc[LSTMmodel['pred'] <= -0.5, 'rPred.5'] = -1
LSTMmodel.loc[LSTMmodel['pred'] >=  0.5, 'rPred.5'] =  1
rounded5 = LSTMmodel[['home','away','result', 'rPred.5']]
# rounded5.plot(figsize=(25,10))

LSTMmodel['rPred.33'] = 0
LSTMmodel.loc[LSTMmodel['pred'] <= -0.33, 'rPred.33'] = -1
LSTMmodel.loc[LSTMmodel['pred'] >=  0.33, 'rPred.33'] =  1

rounded33 = LSTMmodel[['home','away','result', 'rPred.33']]
# rounded33.plot(figsize=(25,10))

LSTMmodel['rPred.2'] = 0
LSTMmodel.loc[LSTMmodel['pred'] <= -0.2, 'rPred.2'] = -1
LSTMmodel.loc[LSTMmodel['pred'] >=  0.2, 'rPred.2'] =  1

rounded20 = LSTMmodel[['home','away','result', 'rPred.2']]
# rounded20.plot(figsize=(25,10))


LSTMmodel['rPred.28'] = 0
LSTMmodel.loc[LSTMmodel['pred'] <= -0.28, 'rPred.28'] = -1
LSTMmodel.loc[LSTMmodel['pred'] >=  0.28, 'rPred.28'] =  1

rounded28 = LSTMmodel[['home','away','result', 'rPred.28']]
# rounded28.plot(figsize=(25,10))
```

Figure 14: Rounding Methods

```python
def fnPredictionPercentage(data, result, prediction):
    data['success'] = np.where(data[result] == data[prediction], True, False)
    success = data[['success']].eq(True).sum()
    total = len(data[['success']])
    return (success['success']/total)*100

def fnPredAccDraw(data, result, prediction):
    dfDraw = data.loc[(data[result] == 0 )]
    success = dfDraw[prediction].eq(0).sum()
    total = len(dfDraw[result])
    return (success/total)*100
# fnPredAccWin(data, 'result', 'rPred2')

def fnPredAccWin(data, result, prediction):
    dfDraw = data.loc[(data[result] == 1 )]
    success = dfDraw[prediction].eq(1).sum()
    total = len(dfDraw[result])
    return (success/total)*100

def fnPredAccLoss(data, result, prediction):
    dfDraw = data.loc[(data[result] == -1 )]
    success = dfDraw[prediction].eq(-1).sum()
    total = len(dfDraw[result])
    return (success/total)*100
```

Figure 15: Function to calculate accuracy

```python
LSTMpredAcc = dict()
predColumns = ['rPred.5', 'rPred.33', 'rPred.2','rPred.28']

for predColumn in predColumns:
    LSTMpredAcc[predColumn] = dict()
    LSTMpredAcc[predColumn]['Accuracy'] = fnPredictionPercentage(LSTMmodel, "result", predColumn)
#     LSTMpredAcc[predColumn]['winAccuracy'] = fnPredAccWin(LSTMmodel, "result", predColumn)
#     LSTMpredAcc[predColumn]['drawAccuracy'] = fnPredAccDraw(LSTMmodel, "result", predColumn)
#     LSTMpredAcc[predColumn]['LossAccuracy'] = fnPredAccLoss(LSTMmodel, "result", predColumn)

dDF = pd.DataFrame.from_dict(LSTMpredAcc)
dDF.T.plot()
dDF
```

executed in 138ms, finished 16:50:00 2021-12-14

	rPred.5	rPred.33	rPred.2	rPred.28
Accuracy	43.589744	40.384615	38.461538	41.025641

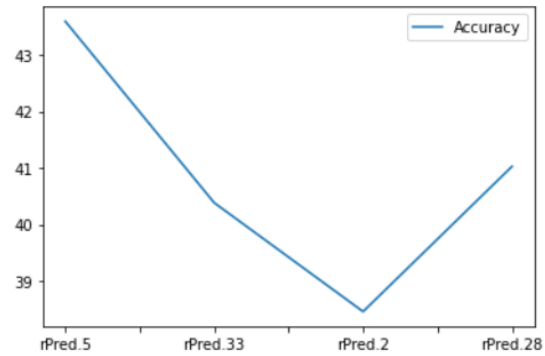

Figure 16: Snippent to get Final Result

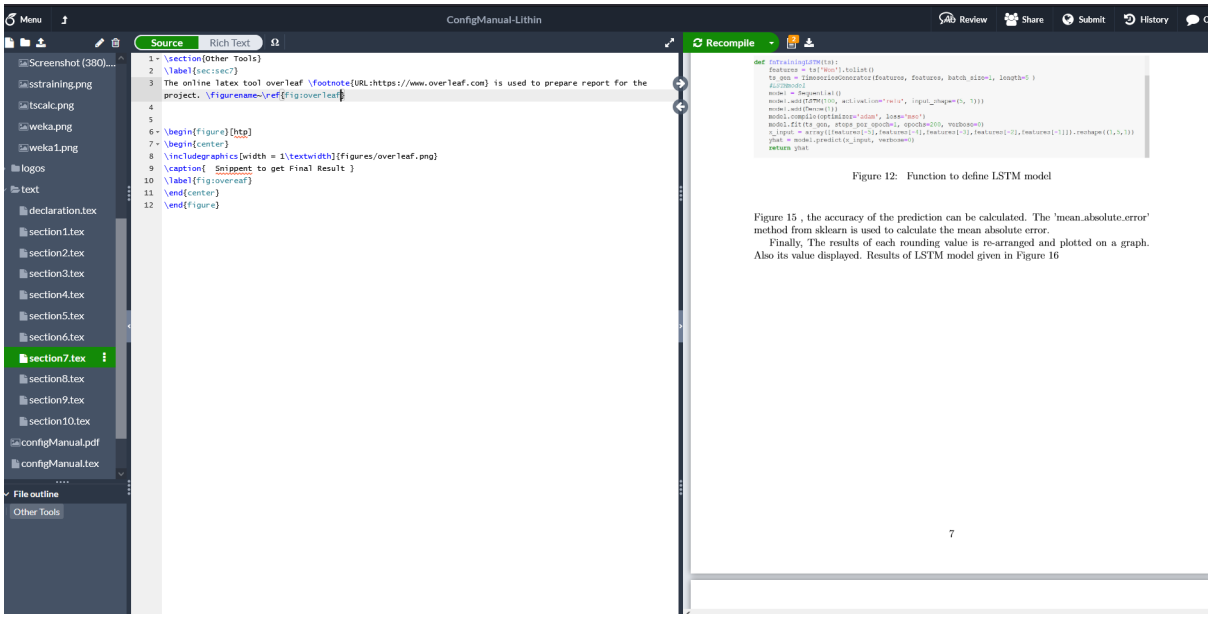

Figure 17: Snippent to get Final Result