Configuration Manual

MSc Research Project
MSc in Data Analytics

Raghav Krishna Kumar
Student ID: X18181848

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes
Dr. Pramod Pathak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Raghav krishna Kumar
Student ID:	18181848
Programme:	Msc in Data Analytics **Year:** 2019
Module:	Msc Research project
Supervisor:	Dr. Paul Stynes & Dr. Pramod Pathak
Submission Due Date:	17/08/2020
Project Title:	Short term forecasting of Agro-products pricing using multivariate time series analysis
Word Count:	1290 **Page Count:** 12

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: Raghav krishna Kumar

Date: 17/08/2020

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

Raghav Krishna Kumar
Student ID: x18181848

1 Introduction

This configuration manual is used to describe the requirements for the research project on short term price forecasting of Agro-products using multivariate time series analysis. This document explains the step by step process to replicate the model with the exact results produced. This manual also consists of the software and hardware specifications required for the project to be replicated.

2 System Specification

2.1 Hardware requirement

The below specified specifications are required to run the model smoothly with no performance issues.

Processor	: Intel(R) Core(TM) i5-8265U CPU @1.60GHz 1.80 GHz
RAM	: 8 GB
Storage	: 256 SSD + 1TB HDD
Operating system	: 64-bit operating system, Windows 10 Home

2.2 Software Requirement

The software which are required for replicating the project is explained in this section while the installation procedures are explained in section 3.

Anaconda which is an open source free distribution of python is used in this project. The Anaconda can be downloaded from the official online website. The Jupyter notebook IDE is used from the anaconda launcher wherein the model is executed using the python language.

3 Installations

This section illustrates the steps for downloading the required softwares and the procedure for the installation.

The Anaconda comes with python pre-loaded with its setup and this is no specific requirement for installing the python separately. The anaconda software is downloaded from the official website as shown in figure 1. The 64-Bit Graphical Installer (466 MB) option is selected from the Windows list.

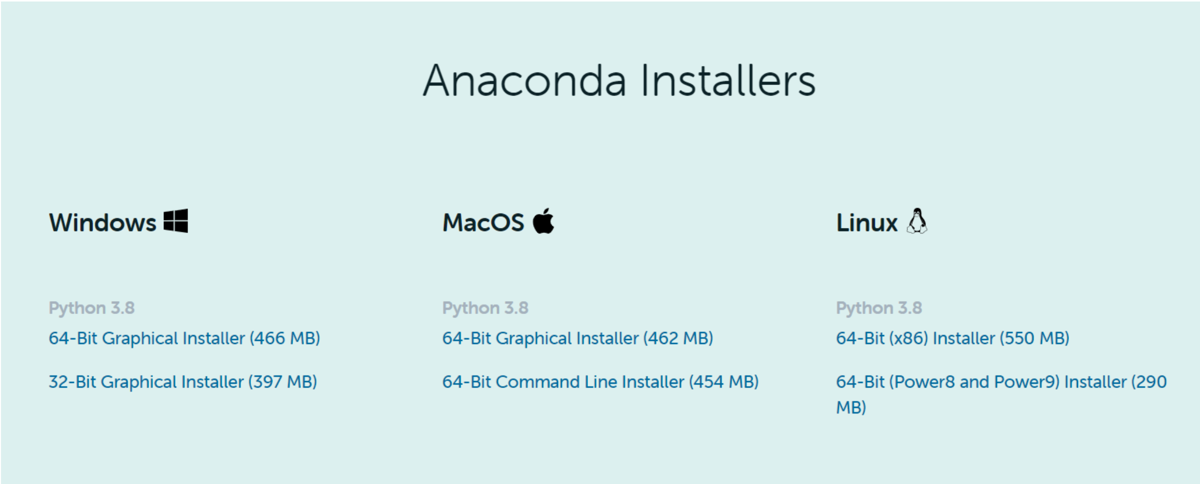


Figure 1: installing Anaconda software

After the successful installation of the anaconda software, the jupyter notebook is launched from the homepage of the software as shown in figure 2

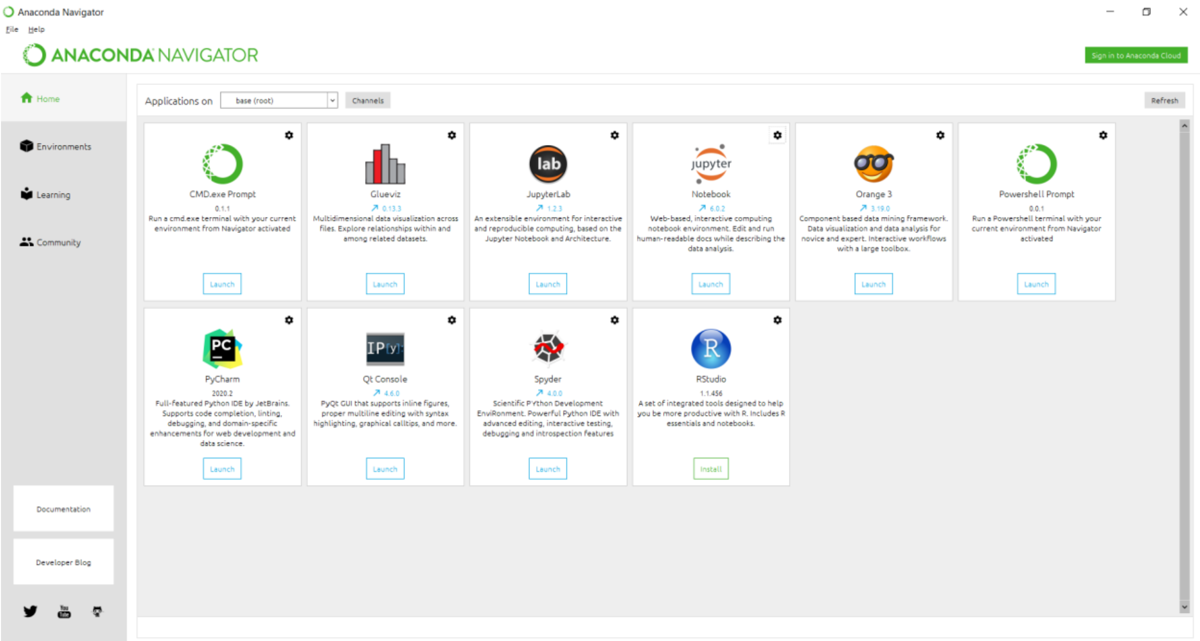


Figure 2: Anaconda Homepage

4 Data Source

The agricultural commodity pricing dataset is obtained from the open source repository managed by the government of India[1]. The prices of onion, tomato, banana and cauliflower are obtained from 2005 to 2016 time period on a daily basis. The climatic factors and weather statistics are obtained from the Indian meteorological site[2] for the same time period on a daily basis.

[1]https://agmarknet.gov.in/
[2] http://dsp.imdpune.gov.in/

5 Project Environment Setup

The Jupyter notebook is launched from the anaconda navigator window and is opened in the online browser and in this case, it opens in google chrome as shown in figure 3. From the Jupyter homepage the new option in top right corner is selected and python 3 is selected as the model is run using python programming language.

Figure 3: Jupyter notebook homepage

When the python 3 option is selected a new python kernel is opened where the coding is performed. The jupyter notebook kernel looks like the figure 4 where the coding needs to be written in the tabular column and the RUN button toolbar is used to execute the written code. The new cell for programming could be created using the + symbol in the tool bar as shown in figure 4.

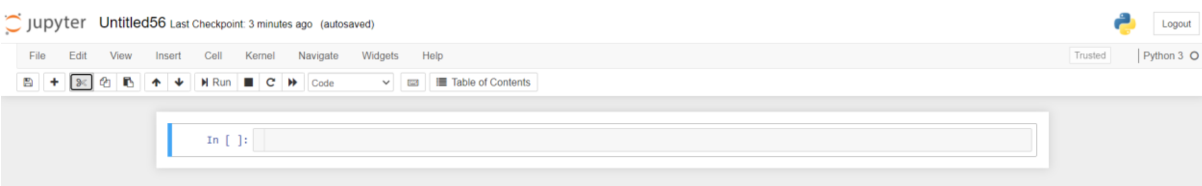


Figure 4: Jupyter notebook execution page

6 Model Implementation:

Prior to the model implementation it is imporatnt to install and load all the libraries and packages required for the model to run. Some of the basic packages are already inbuit to the anaconda environment so there is no need to install but only to import the packages. Some other packages needs to be installed into the anaconda environment. This could be done in the anaconda navigator by selecting the environment tab on the left and searching for the required packages as shown in figure 5.

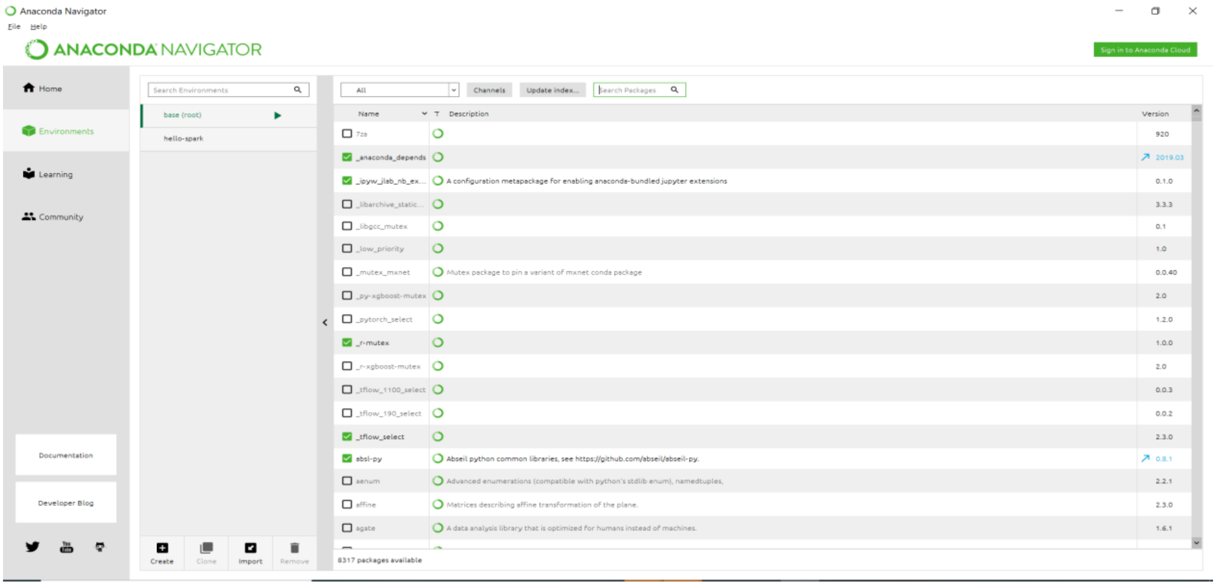


Figure 5: Anaconda Environment setup

Following packages and libraries in figure 6 needs to be installed or exported in to the jupyter notebook

```
In [ ]: import warnings
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import statsmodels.api as sm
        from sklearn import linear_model
        from datetime import datetime
        from sklearn.preprocessing import MinMaxScaler
        import sys
        from scipy.stats import randint
        from sklearn.model_selection import train_test_split
        from sklearn import metrics
        from sklearn.metrics import mean_squared_error,r2_score
        from pandas import read_csv
        from pandas import DataFrame
        from pandas import concat
        import keras
        from keras.layers import Dense
        from keras.models import Sequential
        from keras.utils import to_categorical
        from keras.optimizers import SGD
        from keras.callbacks import EarlyStopping
        from keras.utils import np_utils
        import itertools
        from keras.layers import LSTM
        from keras.layers.convolutional import Conv1D
        from keras.layers.convolutional import MaxPooling1D
        from keras.layers import Dropout
        from statsmodels.tsa.stattools import grangercausalitytests
        from statsmodels.tsa.stattools import adfuller
```

Figure 6: Packages and libraries required.

6.1 Data import and exploratory analysis:

The file RNN-LSTM.ipynb should be opened for this execution. The figure 7 shows the code for importing the data into python and exploratory analysis

```
In [ ]: df = pd.read_csv('only_tomato.csv', parse_dates=['date'], infer_datetime_format=True,low_memory=False, index_col='date')
        print(df)

        res = sm.tsa.seasonal_decompose(df.tomato.dropna(),freq=365)
        fig = res.plot()
        fig.set_figheight(8)
        fig.set_figwidth(15)
        plt.show()
```

Figure 7: data import and exploratory analysis

Selection of the models to run based on the framework is explained. The stationary test using ADF is shwon in the figure 8.

```
733.    ]

In [25]: from statsmodels.tsa.stattools import adfuller
         def test_stationarity(timeseries, window = 12, cutoff = 0.01):

             #Determing rolling statistics
             rolmean = timeseries.rolling(window).mean()
             rolstd = timeseries.rolling(window).std()

             #Plot rolling statistics:
             fig = plt.figure(figsize=(12, 8))
             orig = plt.plot(timeseries, color='blue',label='Original')
             mean = plt.plot(rolmean, color='red', label='Rolling Mean')
             std = plt.plot(rolstd, color='black', label = 'Rolling Std')
             plt.legend(loc='best')
             plt.title('Rolling Mean & Standard Deviation')
             plt.show()

             #Perform Dickey-Fuller test:
             print('Results of Dickey-Fuller Test:')
             dftest = adfuller(timeseries, autolag='AIC', maxlag = 20 )
             dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
             for key,value in dftest[4].items():
                 dfoutput['Critical Value (%s)'%key] = value
             pvalue = dftest[1]
             if pvalue < cutoff:
                 print('p-value = %.4f. The series is likely stationary.' % pvalue)
             else:
                 print('p-value = %.4f. The series is likely non-stationary.' % pvalue)

             print(dfoutput)

In [26]: test_stationarity(dataset['onion price'])
```

Figure 8: ADF test to check stationarity

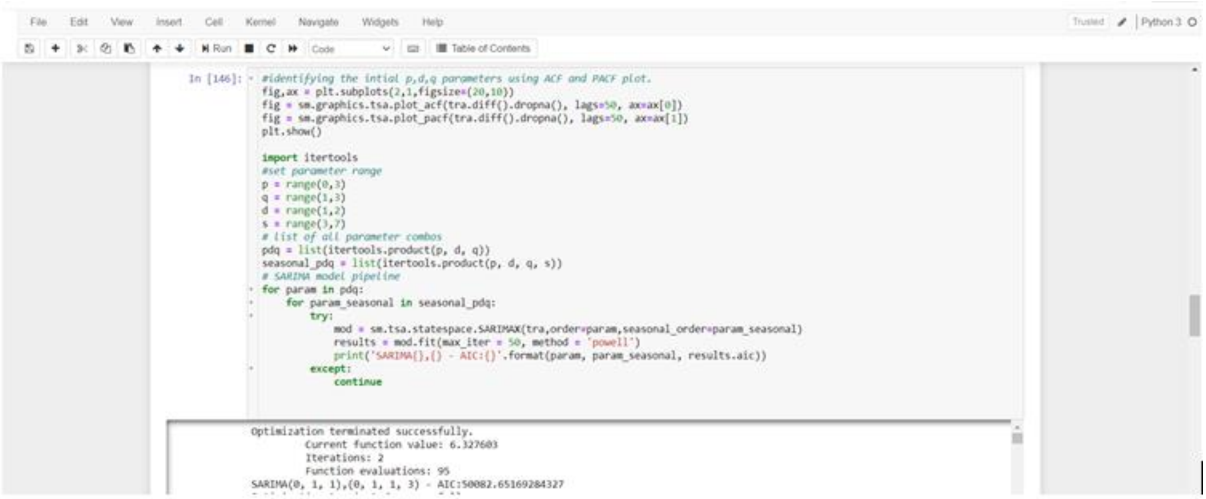
The figure 9 shows the granger causality test to select the model's based on framework.

```
In [35]: from statsmodels.tsa.stattools import grangercausalitytests
         maxlag=12
         test = 'ssr_chi2test'
         def grangers_causation_matrix(data, variables, test='ssr_chi2test', verbose=False):
             """Check Granger Causality of all possible combinations of the Time series.
             The rows are the response variable, columns are predictors. The values in the table
             are the P-Values. P-Values lesser than the significance level (0.05), implies
             the Null Hypothesis that the coefficients of the corresponding past values is
             zero, that is, the X does not cause Y can be rejected.

             data      : pandas dataframe containing the time series variables
             variables : list containing names of the time series variables.
             """
             df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)
             for c in df.columns:
                 for r in df.index:
                     test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)
                     p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
                     if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
                     min_p_value = np.min(p_values)
                     df.loc[r, c] = min_p_value
             df.columns = [var + '_x' for var in variables]
             df.index = [var + '_y' for var in variables]
             return df

         dataset1 = read_csv('final_py.csv',  parse_dates = [['year', 'month', 'day']], index_col=0, date_parser=parse)
         dataset1 = dataset1.drop(['date','state','location'], axis=1)

         grangers_causation_matrix(dataset1, variables = dataset1.columns)
```

Figure 9: granger causality test

6.2 The implementation of Seasonal ARIMA model

The file Seasonal_ARIMA_FINAL.ipynb file should be opened for execution. The acf and pacf model, box jenkins method and grid search technique used to find the precise P,D,Q parameters is explained in the figure 10.



```
In [146]:
#identifying the intial p,d,q parameters using ACF and PACF plot.
fig,ax = plt.subplots(2,1,figsize=(20,10))
fig = sm.graphics.tsa.plot_acf(tra.diff().dropna(), lags=50, ax=ax[0])
fig = sm.graphics.tsa.plot_pacf(tra.diff().dropna(), lags=50, ax=ax[1])
plt.show()

import itertools
#set parameter range
p = range(0,3)
q = range(1,3)
d = range(1,2)
s = range(3,7)
# list of all parameter combos
pdq = list(itertools.product(p, d, q))
seasonal_pdq = list(itertools.product(p, d, q, s))
# SARIMA model pipeline
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(tra,order=param,seasonal_order=param_seasonal)
            results = mod.fit(max_iter = 50, method = 'powell')
            print('SARIMA{},{} - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue

Optimization terminated successfully.
         Current function value: 6.327603
         Iterations: 2
         Function evaluations: 95
SARIMA(0, 1, 1),(0, 1, 1, 3) - AIC:50082.65169284327
```

Figure 20: Selecting parameters for Sarima Model

Running the Seasonal ARIMA model and extracting the values of the evaluation metrics is shown in figure 11.

```
In [9]:
from sklearn.metrics import mean_squared_error
#seasonal ARIMA model
# sarima = sm.tsa.statespace.SARIMAX(tra,order=(0,1,2),seasonal_order=(2,1,1,3),enforce_stationarity=False, enforce_invertibili
# sarima.summary()
# Evaluation Metrics
pred = sarima.predict(tr_end,te_end)[1:]
#print(pred)
#print(tes)
RMSE = np.sqrt(mean_squared_error(pred,tes))
print('Test RMSE: %.3f' % RMSE)
r2 = r2_score(pred,tes)
#print(r2)
#pred.to_csv(r'pred.csv', index = False)
```

Figure 31: implementing the Sarima model

6.3 Implementation of RNN with LSTM (Model 2)

The file RNN-LSTM.ipynb should be opened for this execution. Exploratory analysis of the data required for implementing the RNN is shown in figure 12.

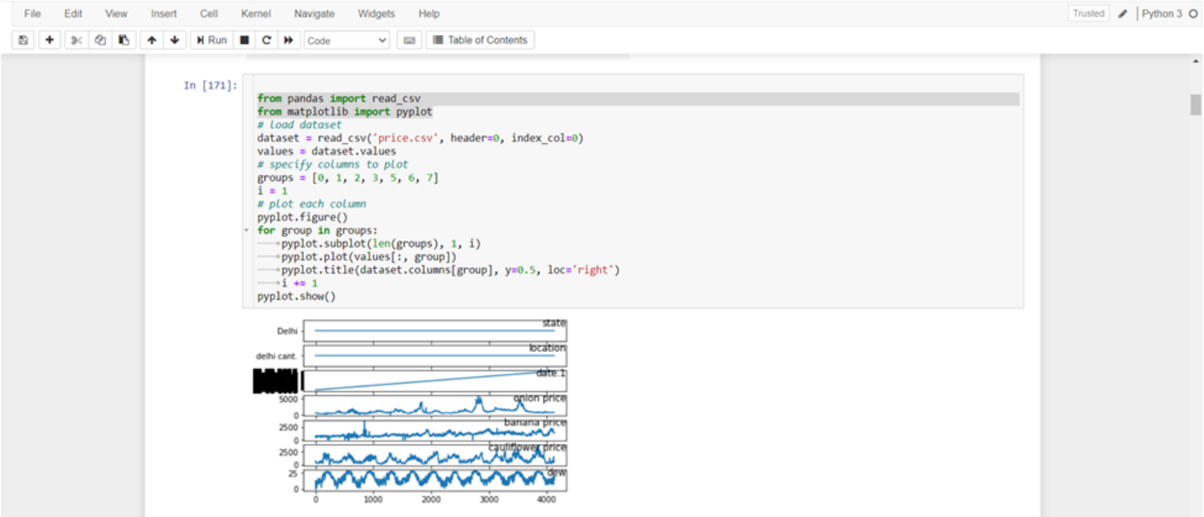


Figure 42: exploratory analysis for RNN with LSTM

Transformation of the data for applying RNN with LSTM model is shown in figure 13.

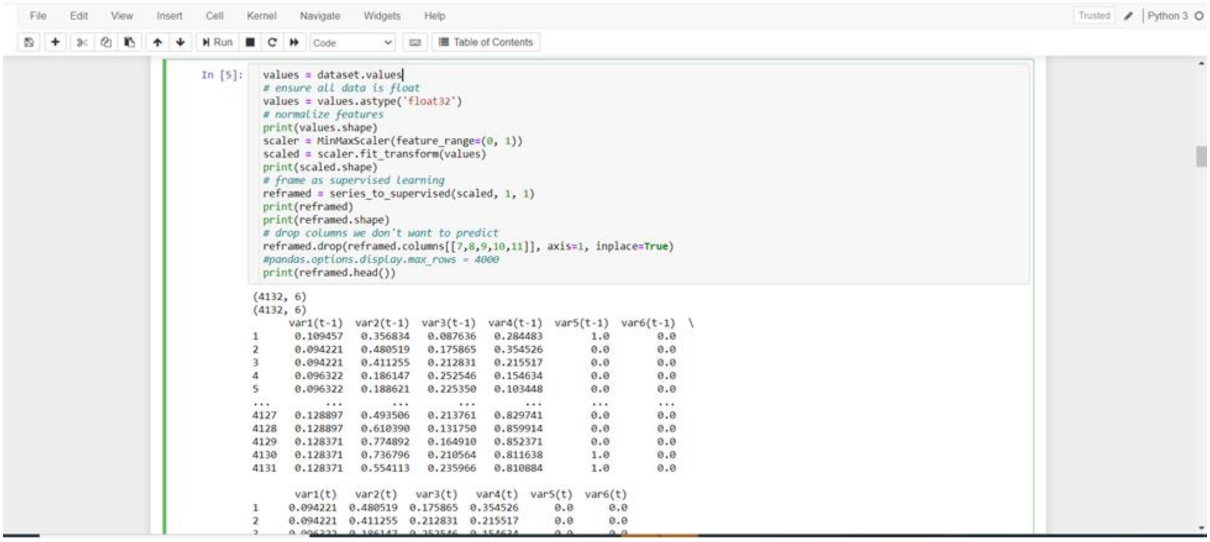


Figure 53: Data transformation for RNN with LSTM

Seperating the training and testing dataset, reshaping the data and fitting the RNN with LSTM model is shown in the figure 14.

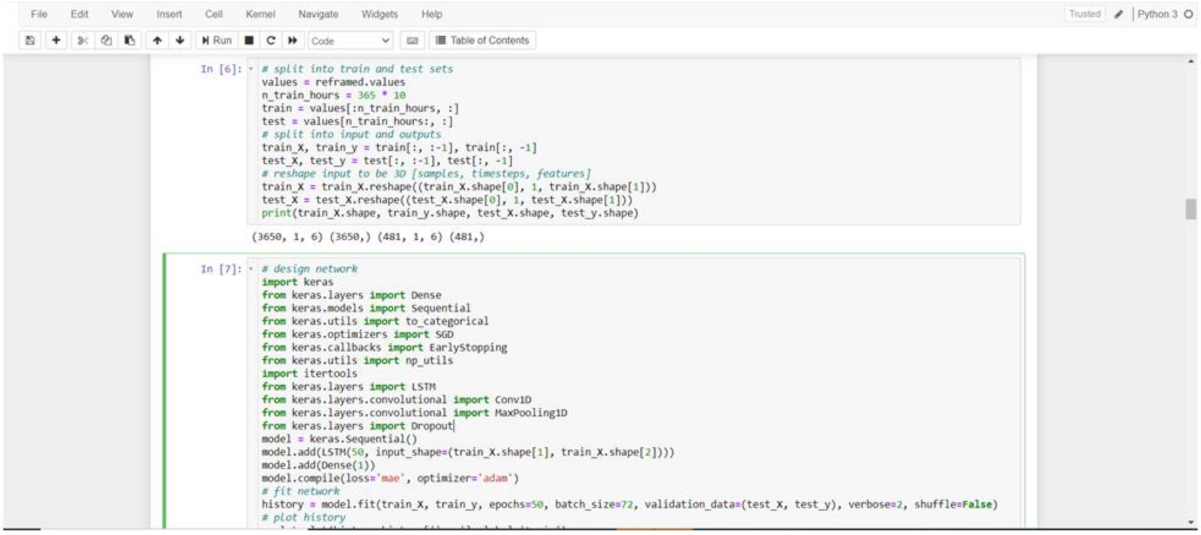


Figure 64: Implementing the RNN with LSTM model

Predicting and extarcting the evaluation metrics from the model is shown in the figure 15.

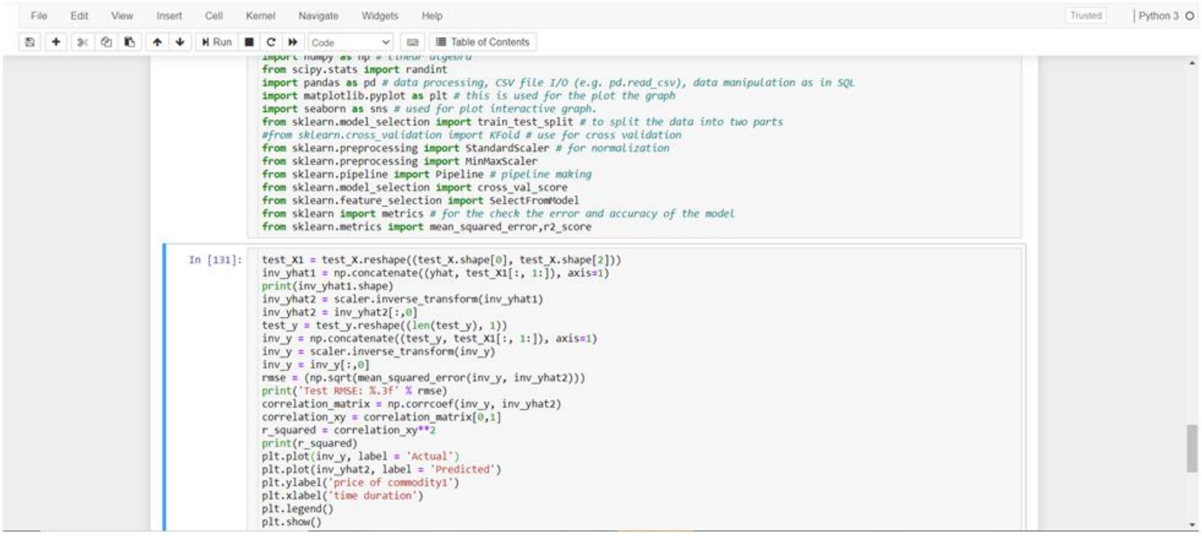


Figure 75: evaluation for RNN with LSTM

Extarcting the evaluation metrics from the model and plotting the graph between predicted and actual is shown in the figure 16.

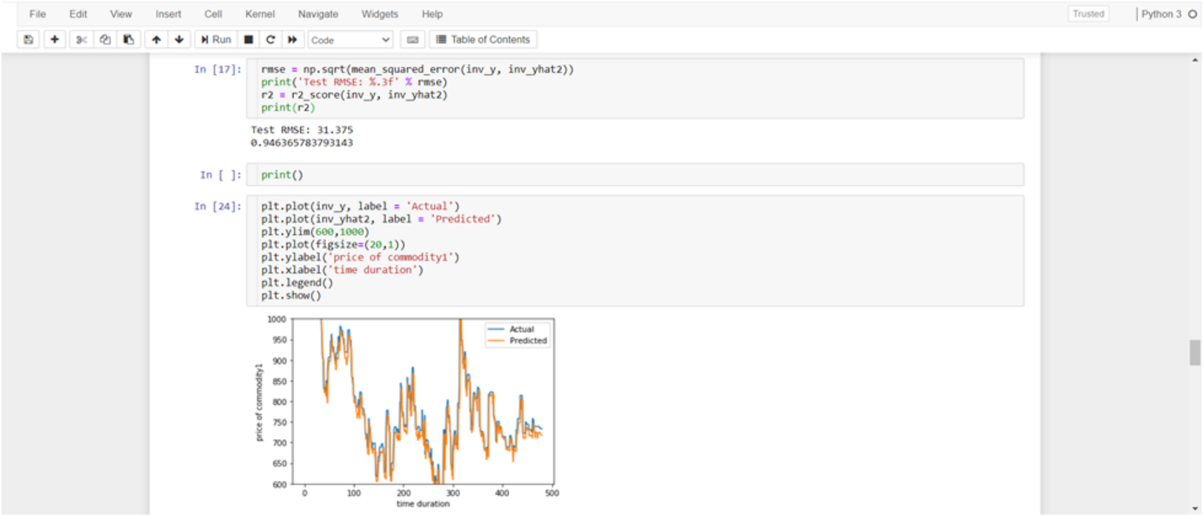


Figure 86: plot between predicted and actual data

6.4 Implementation of Multiple Linear Regression :

Open the file Multiple_linear_regression.ipynb should be opened for execution. Feature extraction done for selecting the required variables in shown in figure 17.

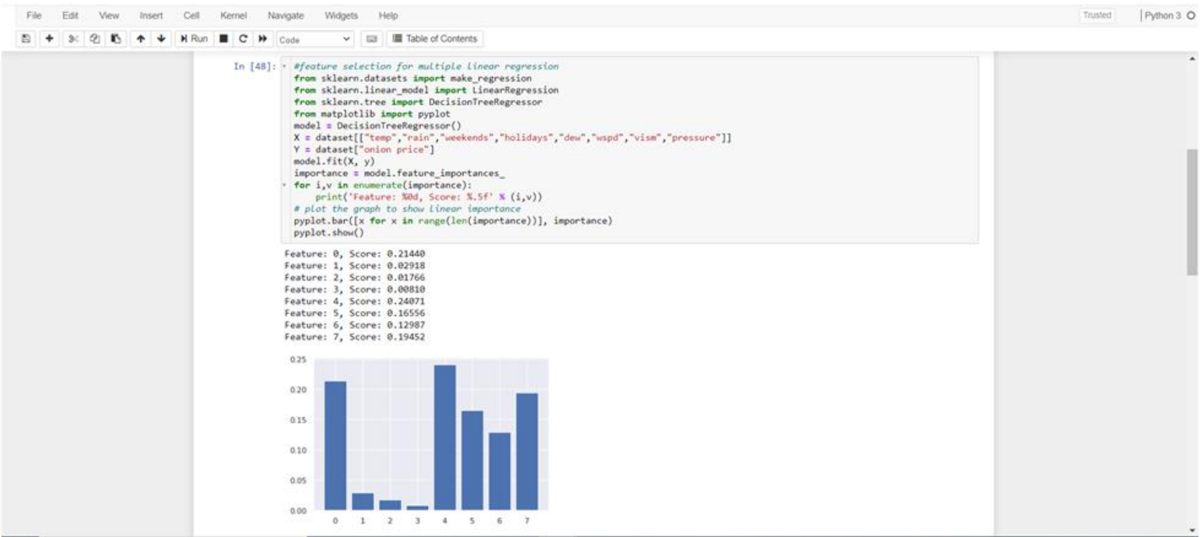


Figure 97: feature extraction for Multiple Linear Regression

The implementation of the linear regression model, extracting the evaluation metrics and plotting the graph between the predicted and the actual is shown in the figure 18.

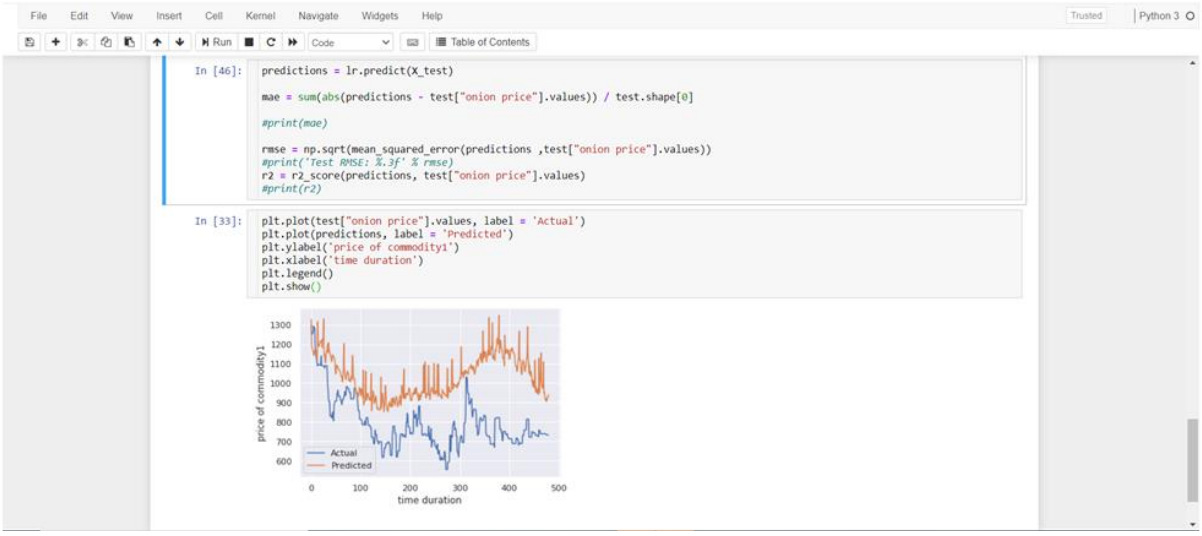


Figure 108: Implementation and evaluation for Multiple Linear Regression.

The entire code is submitted as part of the research project to national college of Ireland.