National College of Ireland

Configuration Manual

MSc Research Project
M.Sc. Data Analytic, MSCDAD_JAN21A_I

Vineet Sawant
Student ID: x19237758

School of Computing
National College of Ireland

Supervisor: Bharathi Chakravarthi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vineet Manoj Sawant
Student ID:	x19237758
Programme:	M.Sc. Data Analytics **Year:** 2021-2022
Module:	M.Sc. Research Project
Lecturer:	Bharathi Chakravarthi
Submission Due Date:	31/1/2022
Project Title:	Forecasting Carbon Dioxide Emission from Energy Consumption within the Industrial Sector in U.S.
Word Count:	1674 **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: Vineet Manoj Sawant

Date: 31/1/2022

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

Vineet Sawant
x19237758

1 Introduction

The presented configuration manual document states the hardware and software requirements or tools used in the MSc research project "Forecasting Carbon Dioxide Emission from Energy Consumption within the Industrial Sector in U.S.". It also presents the code that was used in modeling.

2 System Requirements

2.1 Hardware requirements

Processor: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz
RAM: 8GB
Storage: 512 GB
System Type: 64-bit operating system, x64-based processor
Operating System: Windows 10 (64-bit operating system)

2.2 Software requirements

Programming Environment: Jupyter Notebook by Anaconda and Google Colab.
Diagram: diagram.net (Online software for making charts and process flow diagrams)
Other tools: Microsoft Word (for making tables), Snipping tool (for taking screenshots of diagrams or tables), Microsoft Excel (for reading dataset).

3 Project Development

This section provides a full overview of the steps used to attain the research objectives from the beginning of the project to the completion.

3.1 Data collection

The dataset is downloaded from the U.S. Energy Information Administration (EIA)[1] that contains the monthly carbon dioxide emissions measured in metric tonnes from January 1973 to April 2021. This raw dataset contains 580 rows and 15 columns which contain the month

[1] https://www.eia.gov/totalenergy/data/browser/?tbl=T11.04#/?f=M

and the carbon dioxide emission from thirteen sectors along with a column for the total carbon dioxide emission. The figure 1 shows the dataset that is used for the purpose of the research project.

	A	B	C	D	E	F	
1	Month	Coal Industrial Sector CO2 Emissions	Coal Coke Net Imports CO2 Emissions	Natural Gas Industrial Sector CO2 Emissions	Distillate Fuel Oil Industrial Sector CO2 Emissions	HGL Industrial Sector CO2 Emissions	Kerosene Ir
2	1973 January	33.236	-0.127	42.413	9.501	3.252	
3	1973 February	30.609	-0.014	36.577	10.312	2.495	
4	1973 March	31.408	-0.229	38.3	9.384	1.771	
5	1973 April	30.904	-0.074	42.268	6.701	1.777	
6	1973 May	31.429	-0.323	44.609	8.865	2.457	
7	1973 June	29.806	-0.028	40.207	7.385	2.583	
8	1973 July	29.249	-0.13	43.134	7.784	2.539	
9	1973 August	28.846	-0.167	45.075	8.276	3.339	
10	1973 September	27.997	-0.136	44.928	8.93	3.144	
11	1973 October	30.832	0.198	51.862	8.685	3.442	
12	1973 November	31.702	0.139	51.279	10.588	3.117	
13	1973 December	34.828	0.04	52.52	9.83	1.47	
14	1974 January	32.855	0.43	37.621	10.332	2.639	
15	1974 February	30.818	0.337	40.048	9.083	1.589	
16	1974 March	31.168	0.419	41.261	8.353	1.765	
17	1974 April	30.787	0.504	34.296	7.699	2.3	
18	1974 May	29.955	0.589	40.398	7.353	2.412	
19	1974 June	28.533	0.385	36.919	7.205	2.915	
20	1974 July	28.323	0.408	41.252	7.054	3.114	
21	1974 August	29.187	0.439	43.617	7.423	3.292	
22	1974 September	28.45	0.744	47.997	6.046	3.527	
23	1974 October	30.327	1.036	51.21	8.187	3.319	
24	1974 November	27.05	0.555	51.068	8.685	2.797	
25	1974 December	26.567	0.558	47.756	9.953	2.522	
26	1975 January	29.883	0.971	45.972	12.08	2.748	
27	1975 February	29.944	0.928	32.16	9.709	1.557	

Figure 1 : Dataset

3.2 EDA and Data Visualization

This section contains the python code developed for the initial EDA and time series visualization. The figure 2 shows the code snippet for it.

```
In [ ]: import warnings
        warnings.filterwarnings('ignore')
        import matplotlib.pyplot as plt

In [ ]: import pandas as pd
        import datetime

        df = pd.read_excel (r'Carbon dioxide forecasting.xlsx', sheet_name='Monthly Data')

In [ ]: df.dtypes

In [ ]: df['Coal Industrial Sector CO2 Emissions'].isnull().values.any()

In [ ]: df.set_index('Month', inplace=True)

        df.index
```

Time Series Visualization

```
In [ ]: df['Coal Industrial Sector CO2 Emissions'].plot(linewidth=0.5);
```

Figure 2 : EDA and Visualization

3.3 Checking for Stationarity

The time series data needs to be checked for stationarity . The dickey fuller test is used to check if the time series is stationary. The dickey fuller test generates a p-value which is checked , if the p- value is less than 0.05 the null hypothesis is rejected and it is inferred that the time series is stationary. The figure 3 shows the code snippet for the same.

Check Stationarity of a Time Series - Dickey Fuller

```
In [ ]: from statsmodels.tsa.stattools import adfuller
        #Perform Dickey-Fuller test:
        print('Results of Dickey-Fuller Test:')
        dftest = adfuller(df['Coal Industrial Sector CO2 Emissions'], autolag='AIC')
        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
        print(dfoutput)
```

Figure 3 : Dickey Fuller Test

3.4 Making time series stationary

If the time series data is not stationary we use differencing to make the time series stationary. The time series is also decomposed into its components of trend, seasonality and residuals to better understand the time series. Figure 4 shows the code snippet for the same.

Making Time Series Stationary

```
In [ ]: from statsmodels.tsa.seasonal import seasonal_decompose
        result = seasonal_decompose(df['Coal Industrial Sector CO2 Emissions'], model='additive')
        result.plot()
        plt.show()

In [ ]: df['Difference'] = df['Coal Industrial Sector CO2 Emissions'].diff()
        plt.figure(figsize=(10, 7))
        plt.plot(df['Difference'])
        plt.title('First Order Differenced Series', fontsize=14)
        plt.xlabel('Year', fontsize=12)
        plt.ylabel('Difference', fontsize=12)
        plt.show()
```

Figure 4 : Making Time Series Stationary

Once differencing is done , dickey fuller test needs to be performed on the differenced time series and if its not become stationary differencing needs to be performed again. Usually differencing of order 1 or 2 is sufficient to make the time series stationary.

3.5 Data Split

The data used by the models is split into 80/20 where 80% of the data is used to train the model and the remaining 20% is used for testing. The figure 5 shows the code snippet for the same.

```
In [ ]: data = df.Difference.dropna()
        n= int (len(data) * 0.8)
        train = data [ : n]
        test = data[n:]
```

Figure 5 : Data Split

3.6 Time series forecasting models

This section contains the python code developed for constructing the forecasting models . Jupyter notebook by anaconda is used to develop the Simple Exponential Smoothing (SES) model, Holt-Winter Exponential Smoothing model (HW), Autoregressive Integrated Moving Average (ARIMA) model and the Prophet model . Google Colab is used to develop the Long Short-Term Memory (LSTM) model. Some python libraries like numpy , pandas , matplotlib , sklearn are used in the development of the time series analysis models.

3.6.1 SES

When there is no evident trend or seasonality, the simple exponential smoothing (ses) model is used to forecast time series data. Importing the SimpleExpSmoothing model from the statsmodel package is used to build the model. The fit() method is used to train and test the model, while the predict() function is used to predict potential values. Figure 6 shows the code snippet for the same.

Simple Exponential Model

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from pandas import datetime

# fit model
model_ses = SimpleExpSmoothing(train)
model_fit_ses = model_ses.fit()

print(model_fit_ses.summary())

#Test the model
start_index = datetime(2011, 9, 1)
end_index = datetime(2021, 4, 1)
ses_prediction = model_fit_ses.predict(start=start_index, end=end_index)

#Forecast for the next 6 months
forecast_start_index = datetime(2021,5,1)
forecast_end_index = datetime(2021,10,1)
forecast = model_fit_ses.predict(start=forecast_start_index , end=forecast_end_index )

ses_forecast = forecast.cumsum().add(forecast.cumsum(),fill_value=0)

ses_forecast
```

Figure 6 : SES Model

The test and predicted values are plotted, and MAE, MAPE, and RMSE values are computed with the sklearn and numpy libraries, which may be used to compare the different models. The figure 7 shows the code snippet for the same.

In [ ]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_absolute_error

ses_mae = mean_absolute_error(test, ses_prediction)
print("MAE:", ses_mae)
ses_mape = mean_absolute_percentage_error(test, ses_prediction)
print("MAPE:",ses_mape)
ses_rmse= np.sqrt(mean_squared_error(test, ses_prediction))
print("RMSE: ",ses_rmse)

test.plot(legend=True,label='TEST',figsize=(10,4))
ses_prediction.plot(legend=True,label='PREDICTION',xlim=[datetime(2011, 9, 1),datetime(2021, 4, 1)])
plt.title('Test and Predicted Test using Simple Exponential Model')
```

Figure 7 : SES Evaluation Metrics

The reference for forecasting from SES model was taken from the works of Singh (2018) and the reference for calculating evaluation metrics was taken from the works of Müller (2020).

3.6.2 HW

When there is a trend and seasonality in a time series, the Holt-Winter exponential smoothing model (hw) is used to forecast it. Importing the ExpotentialSmoothing model from the statsmodel library is used to build the model. The fit() method is used to train and test the model, while the predict() function is used to forecast future values. The figure 8 shows the code snippet for the same.

Holt-Winter Exponential Smoothing

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# fit model
model_hw= ExponentialSmoothing(train)
model_fit_hw = model_hw.fit()

print(model_fit_hw.summary())

#Test the model
start_index = datetime(2011, 9, 1)
end_index = datetime(2021, 4, 1)
hw_prediction = model_fit_hw.predict(start=start_index, end=end_index)


#Forecast for the next 6 months
forecast_start_index = datetime(2021,5,1)
forecast_end_index = datetime(2021,10,1)
forecast = model_fit_hw.predict(start=forecast_start_index , end=forecast_end_index )

hw_forecast = forecast.cumsum().add(forecast.cumsum(),fill_value=0)

hw_forecast
```

Figure 8 : HW Model

The test and predicted values are plotted, and MAE, MAPE, and RMSE values are computed with the sklearn and numpy libraries, which may be used to compare the different models. The figure 9 shows the code snippet for the same.

In [ ]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_absolute_error

hw_mae = mean_absolute_error(test, hw_prediction)
print("MAE:", hw_mae)
hw_mape = mean_absolute_percentage_error(test, hw_prediction)
print("MAPE:",hw_mape)
hw_rmse= np.sqrt(mean_squared_error(test, hw_prediction))
print("RMSE: ",hw_rmse)

test.plot(legend=True,label='TEST',figsize=(10,4))
hw_prediction.plot(legend=True,label='PREDICTION',xlim=[datetime(2011, 9, 1),datetime(2021, 4, 1)])
plt.title('Test and Predicted Test using Holt-Winter Exponential Smoothing')
```

Figure 9 : HW Evaluation Metrics

The reference for forecasting from HW model was taken from the works of Singh (2018) and the reference for calculating evaluation metrics was taken from the works of Müller (2020).

3.6.3 ARIMA

Importing the ARIMA model from the statsmodel package creates the Autoregressive Integrated Moving Average (ARIMA) model. We need to pass the order of (p,d,q). The order of ‘d’ is determined by the number of times we difference the time series to make it stationary. To determine the value of ‘p’ and ‘q’ the PACF (Partial autocorrelation function) and ACF (Autocorrelation function) plots are plotted respectively using the plot_pacf() and plot_acf() functions from the statsmodel library. The figure 10 shows the code snippet for the same.

```
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_pacf(df['Difference'].dropna(),lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_acf(df['Difference'].dropna(),lags=40,ax=ax2)
```

Figure 10 : PACF AND ACF Plots

Once the order of (p,d,q) is determined the ARIMA model is trained and fitted using the fit() function. Using the predict() function the future values are forecasted. The figure 11 shows the code snippet for the same.

```
In [ ]: import warnings
        warnings.filterwarnings('ignore')
        from statsmodels.tsa.arima.model import ARIMA
        fig = plt.figure(figsize=(12,8))

        # fit model
        model_arima= ARIMA(train, order=(2, 0, 2))
        model_fit_arima = model_arima.fit()

        print(model_fit_arima.summary())

        #Test the model
        start_index = datetime(2011, 9, 1)
        end_index = datetime(2021, 4, 1)
        arima_prediction = model_fit_arima.predict(start=start_index, end=end_index)


        #Forecast for the next 6 months
        forecast_start_index = datetime(2021,5,1)
        forecast_end_index = datetime(2021,10,1)
        forecast = model_fit_arima.predict(start=forecast_start_index , end=forecast_end_index )

        arima_forecast = forecast.cumsum().add(forecast.cumsum(),fill_value=0)

        arima_forecast
```

Figure 11 : ARIMA Model

The test and predicted values are plotted, and MAE, MAPE, and RMSE values are computed with the sklearn and numpy libraries, which may be used to compare the different models. The figure 12 shows the code snippet for the same.

```
In [ ]: import numpy as np
        from sklearn.metrics import mean_squared_error
        from sklearn.metrics import mean_absolute_percentage_error
        from sklearn.metrics import mean_absolute_error

        arima_mae = mean_absolute_error(test, arima_prediction)
        print("MAE:", ses_mae)
        arima_mape = mean_absolute_percentage_error(test, arima_prediction)
        print("MAPE:",ses_mape)
        arima_rmse= np.sqrt(mean_squared_error(test, arima_prediction))
        print("RMSE: ",arima_rmse)

        test.plot(legend=True,label='TEST',figsize=(10,4))
        arima_prediction.plot(legend=True,label='PREDICTION',xlim=[datetime(2011, 9, 1),datetime(2021, 4, 1)])
        plt.title('Test and Predicted Test using ARIMA')
```

Figure 12 : ARIMA Evaluation Metrics

The reference for forecasting from ARIMA model was taken from the works of Singh (2018) and the reference for calculating evaluation metrics was taken from the works of Müller (2020).

3.6.4 Prophet

Importing the Prophet model from the fbprophet library is used to create the Prophet model. The data for the month and timeseries are loaded in a dataframe, with the column names changed to 'ds' and 'y'. The data is split into two categories: 80% for training and 20% for training. The figure 13 shows the code snippet for the same.

```
In [ ]: from fbprophet import Prophet

        df = pd.read_excel (r'Carbon dioxide forecasting.xlsx', sheet_name='Monthly Data')

        df= df[['Month','Coal Industrial Sector CO2 Emissions']]
        print(df)

In [ ]: df.columns = ["ds","y"]
        df.head()

In [ ]: n= int (len(df) * 0.8)
        train = df [:n]
        test = df[n:]
```

Figure 13 : Data Load & Data Spilt Prophet

The Prophet() method is used to create the model. The fit() method is used to train and fit the model on the training dataset. To generate future predictions, the make future dataframe() method is used to build a dataframe containing future dates, with the periods and frequency parameters provided in. The forecasts' frequency is set to 'MS' (Month Start). To generate the forecast, the future dataframe is supplied to the predict() method. The figure 14 shows the code snippet for the same.

```
In [ ]: model = Prophet()

In [ ]: model_fit = model.fit(train)

In [ ]: model

In [ ]: future = model.make_future_dataframe(periods=116 , freq = 'MS')
        forecast = model.make_future_dataframe(periods=122, freq ='MS')

In [ ]: pred= model.predict(future)
        forecast = model.predict(forecast)

In [ ]: pred
        forecast

In [ ]: pred[["ds","yhat","yhat_lower","yhat_upper"]].tail()
        forecast[["ds","yhat","yhat_lower","yhat_upper"]].tail()
```

Figure 14 : Prophet Model

The test and predicted values are plotted, and MAE, MAPE, and RMSE values are computed with the sklearn and numpy libraries, which may be used to compare the different models. The figure 15 shows the code snippet for the same.

```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_absolute_error

prophet_mae = mean_absolute_error(test.y, pred[n:].yhat)
print("MAE:", prophet_mae)
prophet_mape = mean_absolute_percentage_error(test.y, pred[n:].yhat)
print("MAPE:",prophet_mape)
prophet_rmse= np.sqrt(mean_squared_error(test.y, pred[n:].yhat))
print("RMSE: ",prophet_rmse)
model.plot(pred,uncertainty=True)
plt.show()
```

```
test.set_index('ds', inplace=True)
pred.set_index('ds', inplace=True)
test['y'].plot(legend=True,label='TEST',figsize=(10,4))
pred[n:].yhat.plot(legend=True,label='PREDICTION',color='red')
plt.title('Test and Predicted Test using Prophet')
```

Figure 15 : Prophet Evaluation Metrics

The reference for forecasting from Prophet model was taken from the works of Brownlee (2020) and the reference for calculating evaluation metrics was taken from the works of Müller (2020).

3.6.5 LSTM

The model for long short-term memory (LSTM) is built in Jupyter notebook with Google Colab. The time series data is imported from Excel into a Jupyter notebook dataframe. The seed() parameter is set to ensure that the model is repeatable. The dataframe index is assigned to the 'Month' column with index frequency set to 'MS', and the 'Month' column is transformed to a datetime. Training and testing data are separated from the time series data. Figure 16 shows the code snippet for the same.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import seed
seed(1)

df = pd.read_excel (r'/content/Carbon dioxide forecasting.xlsx', sheet_name='Monthly Data')

df.head()

df= df[['Month','Coal Industrial Sector CO2 Emissions']]
df['Month'] = pd.to_datetime(df.Month)
df.set_index('Month', inplace=True)
df.index.freq='MS'
print(df.head())

print(df.info())

df.plot(figsize=(12,6))
plt.show()

train = df [ : 574]
test = df[574:]
```

Figure 16 : Data Load & Data Split LSTM

The MinMaxScaler, which converts values in the range of 0 and 1, is used to change the values in the train and test sets. The model's sample input and output components are generated by the TimeseriesGenerator. Figure 17 shows the code snippet of the same.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

scaler.fit(train)
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)

scaled_train[:10]

from keras.preprocessing.sequence import TimeseriesGenerator

# define generator
n_input = 6
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

Figure 17 : LSTM Tranformation and Generator

The loss function is plotted after the LSTM model has been trained and fitted using 50 epochs on the train dataset values. The figure 18 shows the code snippet for the same.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

model.summary()

# fit model
model.fit(generator,epochs=50)

loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

Figure 18 : LSTM Model

To forecast values, the predict() function is used. The code snippet is shown in figure 19.

```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))

for i in range(len(test)):

    # get the prediction value for the first batch
    current_pred = model.predict(current_batch)[0]

    # append the prediction into the array
    test_predictions.append(current_pred)

    # use the prediction to update the batch and remove the first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)

test_predictions

test.head()

true_predictions = scaler.inverse_transform(test_predictions)

test['Predictions'] = true_predictions
```

Figure 19 : Prediction using LSTM

The test and predicted values are plotted, and MAE, MAPE, and RMSE values are computed with the sklearn and numpy libraries, which may be used to compare the different models. The figure 20 shows the code snippet for the same.

```
test.plot(figsize=(14,5))
```

```
test
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(test['Coal Industrial Sector CO2 Emissions'],test['Predictions'])
print("MAE:", mae)
mape = mean_absolute_percentage_error(test['Coal Industrial Sector CO2 Emissions'],test['Predictions'])
print("MAPE:",mape)
rmse=sqrt(mean_squared_error(test['Coal Industrial Sector CO2 Emissions'],test['Predictions']))
print("RMSE:",rmse)
```

Figure 20 : LSTM Evaluation Metric

The reference for forecasting from LSTM model was taken from the works of Hebbar (2021) and the reference for calculating evaluation metrics was taken from the works of Müller (2020).

References

Brownlee, J 2020, Time Series Forecasting With Prophet in Python, Machine Learning Mastery, viewed 16 November 2021, <https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/>

Hebbar, N 2021, RNN_Youtube.ipynb, Available at: https://github.com/nachi-hebbar/Time-Series-Forecasting-LSTM/blob/main/RNN_Youtube.ipynb

Müller, F 2020, Evaluate the Performance of Time Series Forecasting Models with Python, relataly.com, viewed 16 November 2021, https://www.relataly.com/evaluating-time-series-forecasting-models/923/

Singh, G 2018, 7 methods to perform Time Series forecasting (with Python codes), Analytics Vidya, viewed 16 November 2021, < https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/>