National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shambhu Chandrakant Patole
Student ID: x19213743

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shambhu Chandrakant Patole
Student ID:	x19213743
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Noel Cosgrave
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	868
Page Count:	11

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:	
Date:	16th August 2021

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

Shambhu Chandrakant Patole
x19213743

1 Introduction

This is the configuration manual that documents the software and the hardware requirements and/or tools that are used for the final research project. The title of the research is, 'Time Series Forecasting of Methane Emissions from Livestock using Machine Learning' and it also provides with the code snippets used for implementation and evaluation of this project.

2 Required System Configuration

2.1 Hardware Used

Processor: Inter(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40GHz
RAM: 16 GB
Storage Capacity: 512 GB SSD (Solid State Drive)
System Type: 64-bit Operating System, x-64 based processor
GPU: NVIDIA GEFORCE
Operating System: Windows 10 (64-bit Operating System)

2.2 Software Used

Data Cleaning: RStudio
Programming and forecasting graphs: Jupyter Notebook
Visualization: Microsoft Excel, Matplotlib
Flow chart and diagrams: Draw.io (Onlin portal for preparing diagrams
Other tools: Snipping tool, Microsoft Excel,

3 Research Project Development

In this section, a brief explanation of the steps performed from start to end of the project is addressed.

3.1 Data Collection and Pre-processing

The dataset for this research contain data for Methane Emissions (CH4) from Enteric Fermentation downloaded from Food and Agriculture Organization of the United Nations website [1], who has made this data publicly available for research, statistics and science. The data contains annual CH4 emissions from 1961 to 2019 having projections for 2030 and 2050. The data contains data by country, regions and special groups, with global coverage. The data set contains 369580 rows. A sample set of rows from the raw data is shown below in Figure 1.

Area Code	Area	Item Code	Item	Element Code	Element	Year Code	Year	Source Code	Source	Unit	Value	Flag	Note
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1961	1961	3050	FAO TIER 1	kilotonnes	240.6831	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1962	1962	3050	FAO TIER 1	kilotonnes	245.3106	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1963	1963	3050	FAO TIER 1	kilotonnes	255.8285	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1964	1964	3050	FAO TIER 1	kilotonnes	259.065	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1965	1965	3050	FAO TIER 1	kilotonnes	265.598	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1966	1966	3050	FAO TIER 1	kilotonnes	276.994	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1967	1967	3050	FAO TIER 1	kilotonnes	280.094	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1968	1968	3050	FAO TIER 1	kilotonnes	288.821	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1969	1969	3050	FAO TIER 1	kilotonnes	286.382	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1970	1970	3050	FAO TIER 1	kilotonnes	290.26	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1971	1971	3050	FAO TIER 1	kilotonnes	287.79	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1972	1972	3050	FAO TIER 1	kilotonnes	231.527	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1973	1973	3050	FAO TIER 1	kilotonnes	244.979	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1974	1974	3050	FAO TIER 1	kilotonnes	262.836	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1975	1975	3050	FAO TIER 1	kilotonnes	282.074	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1976	1976	3050	FAO TIER 1	kilotonnes	288.225	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1977	1977	3050	FAO TIER 1	kilotonnes	280.876	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1978	1978	3050	FAO TIER 1	kilotonnes	280.285	A	
2	Afghanistan	1755	All Anima	7225	Emissions (CH4)	1979	1979	3050	FAO TIER 1	kilotonnes	274.235	A	

Figure 1: Raw Data Sample

Figure 2 mentioned below shows the data cleaning code snippet,

```
mel_data <- read.csv('C:\\Users\\91773\\Downloads\\Emissions_Agriculture_Enteric_Fermentation_E_All_Data_(Normalized)\\Emissions_Agri
head(mel_data)

mel_data_sel <- mel_data[c("Area","Item","Element","Year","Source","Unit","Value")]
head(mel_data_sel)

mel_data_faoch4 <- mel_data_sel[which(mel_data_sel$Element == 'Emissions (CH4)' & mel_data_sel$Source == 'FAO TIER 1'),]
head(mel_data_faoch4)

mel_data_ani <- mel_data_faoch4[which(mel_data_faoch4$Item == 'All Animals'),]
head(mel_data_ani)

mel_data_final <- mel_data_ani[c("Area","Item","Year","Value")]
head(mel_data_final)

mel_data_last <- mel_data_final[which(mel_data_final$Area == 'World'),]
head(mel_data_last)

mel_data_last_final <- mel_data_last[c("Year","Value")]
head(mel_data_last_final)

write.csv(mel_data_last_final,"mel_data_world.csv", row.names = FALSE)
```

Figure 2: Cleaning Data and Exporting cleaned CSV data file

Below mentioned steps explain the detailed steps performed in above figure,

1. In the beginning, from 14 columns of raw data set, only 7 columns are included. Other columns are neglected.

[1] http://www.fao.org/faostat/en/#data/GE

2. From the Source column only 'FAO Tier 1' data is selected and 'UNFCCC' is removed as there were some missing values.

3. From Element column only 'Emissions (CH4)' values are included, 'Stocks' vales were were removed as they were not required for this research purpose.

4. From the Item column, only the data for 'All Animals' is selected belonging to only for 'World' values from Area column.

5. In the end, the dataframe was exported to a comma separated values (csv) file and this cleaned data was further used for modelling.

6. Before performing modelling steps, rows for 2030 and 2050 projections are also excluded.

This concludes the exploratory data analysis performed

4 Machine Learning Algorithms

In this section, the code snippets and it explanation are included using which ARIMA, PROPHET and SVM forecasting models were used. The models were coded in Jupyter Notebook using Python programming language where libraries like pandas, numpy, matplotlib, and scikit-learn.

4.1 ARIMA

```
# Importing required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from matplotlib.pylab import rcParams
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import acf,pacf
from statsmodels.tsa.stattools import adfuller
```

```
import warnings
warnings.filterwarnings("ignore") #for ignoring the unnecessary warnings if any
```

```
#For parsing year column as date

def parser(x):
    return datetime.strptime(x,'%Y')
```

```
#Importing the dataset with Year column as index and removing 2030 and 2050 projection rows

df = pd.read_csv('mel_data_world.csv', header=0, parse_dates=[3], index_col=3, squeeze=True, date_parser=parser)[:-2]
```

```
df.tail()
```

Figure 3: ARIMA - Importing libraries, parsing and indexing year column

Figure 3 shows the libraries that were imported. A parser function is used to parse the year column as datetime column and also index the year column.

```
#Excluding the columns that are not required

del df['Area']
del df['Item']
del df['Element']
del df['Source']
del df['Unit']
```

```
df.head()
```

```
#intial plot
df.plot(figsize=(12, 8))
```

```
# To Check if the Time series is Stationary or not using Dickey-Fuller Test

def test_stat(ts):
    movingavg = ts.rolling(window=1).mean()
    movingstd = ts.rolling(window=1).std()

    original = plt.plot(ts, color='blue', label = 'Original')
    mean = plt.plot(movingavg, color='red', label = 'Rolling Mean')
    std = plt.plot(movingstd, color = 'black', label = 'Rolling Std deviation')
    plt.legend(loc = 'best')
    plt.title('Rolling mean and Std deviation')
    plt.show(block=False)

    print("Dickey fuller has been performed and the results are: ")
    out = adfuller(ts['Value'], autolag = 'AIC')
    op = pd.Series(out[0:4], index = ['Test-Statistic', 'p-value', 'Lags', 'Observations Used'])
    for key, value in out[4].items():
        op['Critical Value(%s)' %key] = value
    print(op)
```

```
test_stat(df)
```

Figure 4: ARIMA - Check stationarity

As seen in Figure 4, a new stationarity function was created that will perform Dickey-Fuller test. As the p value was ¿ 0.005, the given time series is considered non-stationary. Next, log of the data is taken and subtracted and the differencing value is decided. Later, Autocorrelation and Partial Autocorrelation are plotted, as shown in Figure 5 and accordingly p, d and q values were decided.

```
df_log = np.log(df)
plt.plot(df_log)
```

```
#1 order differencing

df_log_diff = df_log - df_log.shift()
df_log_diff.dropna(inplace=True)
test_stat(df_log_diff)
```

```
test_stat(df_log_diff)
```

```
#ACF PLOT

lag_acf = acf(df_log_diff, nlags = 25, fft=False)
plt.figure(figsize=(16,7))
plt.plot(lag_acf,marker='*')
plt.axhline(y=0, linestyle='--',color = 'black')
plt.axhline(y=-1.96/np.sqrt(len(df_log_diff)), linestyle='-', color='black')
plt.axhline(y=1.96/np.sqrt(len(df_log_diff)), linestyle='-', color='black')
plt.title('ACF PLOT')
```

```
#PACF PLOT

lag_pacf = pacf(df_log_diff, nlags = 25)
plt.figure(figsize=(16,7))
plt.plot(lag_pacf,marker='*')
plt.axhline(y=0, linestyle='--',color = 'black')
plt.axhline(y=-1.96/np.sqrt(len(df_log_diff)), linestyle='-', color='black')
plt.axhline(y=1.96/np.sqrt(len(df_log_diff)), linestyle='-', color='black')
plt.title('PACF PLOT')
```

Figure 5: ARIMA - differencing, acf, pacf

For this research, the ARIMA model was built with order (3,1,2) and fitted using .fit() function as shown in Figure 6. Next, .forecast() gives the prediction of number of further time steps.

The reference for the codes has been taken from machine learning mastery website [2].

4.2 PROPHET

For PROPHET model, seaborn and marplotlib libraries were used to create charts. Initially the model prophet is loaded using the fbprophet package as seen in Figure 8. out of the Train and test data, train data is used to train the model. Next, .make_future_dataframe() function is used to create a new dataframe with future dates and later called using .predict() function as seen in Figure 9. 'YS' is taken as the frequency as for this research, the data was annually. Atlast, the plots are taken to show the forecast and the original values followed by evaluating the performance metrics as shown in Figure 10.

[2] https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

```
# Dividing data into train and test

train = np.log(df.Value[:int(len(df)*.90)])
test = np.log(df.Value[len(train):])
print(len(train))
print(len(test))
```

```
# Preparing the model with order parameters
#from tests and graphs for Yearly Data ('YS')

model = ARIMA(train, order=(3,1,2), freq = 'YS')
```

```
#Fitting the model with

results = model.fit(disp = -1)
```

```
results.summary()
```

```
# Original vs Fitting graph

newdata = df_log_diff[0:len(train)]
plt.plot(newdata, label = 'Original')
plt.plot(results.fittedvalues, color='red', label = 'Fitting')
plt.legend(loc='upper left', fontsize=10)
plt.tight_layout()
plt.title('Original vs Fitting')
```

```
frcst, se, conf = results.forecast(len(test))
```

```
frcst_df = pd.Series(frcst, index = test.index)
```

Figure 6: ARIMA - Building model, train and fit

```
plt.figure(figsize=(12,5), dpi=100)
plt.plot(np.exp(test), label='Original', color='blue')
plt.plot(np.exp(frcst_df), label = 'Forecast', color='red')
plt.title('Forecast vs Original')
plt.legend(loc='upper left', fontsize=10)
plt.show()
```

```
# Calculating performance metrics

from math import sqrt
from sklearn import metrics

maeerror = sqrt(metrics.mean_absolute_error(test,frcst_df)) #calculate MAE
print('MAE value of the SVR Model is:', maeerror)
rmseerror = sqrt(metrics.mean_squared_error(test,frcst_df)) #calculate rmse
print('RMSE value of the SVR Model is:', rmseerror)
mapeerror = np.mean(np.abs((test - frcst_df) / test)) * 100
print('MAPE value of the SVR Model is:', mapeerror)
```

Figure 7: ARIMA - Final plots and Evaluation Metrics

```
[ ]  # Importing required libraries

     from fbprophet import Prophet
     from fbprophet.plot import plot_plotly
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import plotly.offline as py
     from datetime import datetime
     from sklearn.metrics import mean_absolute_error
     from statsmodels.tools.eval_measures import rmse
```

```
[ ]  #For parsing year column as date

     def parser(x):
         return datetime.strptime(x,'%Y')
```

```
[ ]  df = pd.read_csv('mel_data_world.csv', header=0, parse_dates=[3], squeeze=True, date_parser=parser)[:-2]
```

```
[ ]  df.info()
```

```
[ ]  del df['Area']
     del df['Item']
     del df['Element']
     del df['Source']
     del df['Unit']
     df.head()
```

Figure 8: Prophet - Importing libraries, parsing and indexing year column

4.3 SVM

Support vector machine (SVM) is another model used for this research. Figure 11 shows the initial import of libraries required and parsing and indexing the year column to convert it into datetime function. Initially, the data is divided into train and test as seen in Figure 12. SVR library from sklearn is imported. First, the kernel is chosen that suits the data and then actual training happens using 'rbf' kernel, which can be seen in Figure 13. Once the predicted values are given by the model, lastly the performance metrics are calculated using the built in function from sklearn as captured in Figure 14.

```
# Dividing the dat training and testing

train = df[:int(len(df)*.92)]
test = df[len(train):]
```

```
test.tail
```

```
X= train[['Year']]
y= train.iloc[:,1]
```

```
# Renaming columns to ds and y, as requirements from Prohet model

train_ds= pd.DataFrame()
train_ds['ds'] = X["Year"]
train_ds['y']=y
train_ds.tail(2)
```

```
# Modelling and fitting

prop_model = Prophet()
prop_model.fit(train_ds)
```

```
future= prop_model.make_future_dataframe(periods=len(test), freq='YS')
prop_pred = prop_model.predict(future)
prop_pred.tail(5)
```

Figure 9: Prophet - Modelling

```
prop_prediction = pd.DataFrame({"Year" : prop_pred[-len(test):]['ds'], "Prediction" : prop_pred[-len(test):]['yhat']})
prop_prediction = prop_prediction.set_index("Year")
prop_prediction.index.freq = 'YS'
```

```
test.columns = ['ds','y']
```

```
test.head()
```

```
test["Prophet_Pred_Val"] = prop_prediction['Prediction'].values
test = test.set_index("ds")
```

```
plt.figure(figsize=(20,7))
sns.lineplot(x=test.index, y=test["y"], color="blue",marker='.', label='Original')
sns.lineplot(x=test.index, y= test["Prophet_Pred_Val"],color='red',marker='.',label='Prediction')
```

```
#  Calculating the performance metrics

print('Test Mean Absolute Error for Prophet: ', mean_absolute_error(test['y'],test['Prophet_Pred_Val']))
print('Test Root Mean Squared Error for Prophet: ', np.sqrt(rmse(test['y'], test['Prophet_Pred_Val'])))
print('Test Mean Absolute Percentage Error for Prophet: ', np.mean(np.abs((test['y'] - test['Prophet_Pred_Val']) / test['y'])) * 100)
```

Figure 10: Prophet - Calculating evaluation metrics

```
[ ]  # Importing necessary packages

     import numpy as np
     from sklearn.svm import SVR
     import pandas as pd
     from sklearn.preprocessing import  MinMaxScaler
     import matplotlib.pyplot as plt
     import seaborn as sns
     from datetime import datetime
     from sklearn.metrics import mean_absolute_error
     from sklearn.metrics import mean_squared_error
```

```
[ ]  #For parsing year column as date

     def parser(x):
         return datetime.strptime(x,'%Y')
```

```
[ ]  df = pd.read_csv('mel_data_world.csv', header=0, parse_dates=[3], squeeze=True, date_parser=parser)[:-2]
```

```
[ ]  del df['Area']
     del df['Item']
     del df['Element']
     del df['Source']
     del df['Unit']
     df.head()
```

Figure 11: SVM - Importing libraries, parsing and indexing year column

```
[ ]  X = df.drop(['Value'], axis = 1)
     y = df['Value']
     y=np.log1p(y)
```

```
[ ]  y.tail()
```

```
[ ]  train = df[:int(len(df)*.92)]
     test = df[len(train):]
     X= train[['Year']]
     y= train.iloc[:,1]
```

```
[ ]  X_train=X[:int(len(df)*.92)]
     y_train=y[:int(len(df)*.92)]
     X_test=X[int(len(df)*.92):]
     y_test=y[int(len(df)*.92):]
```

Figure 12: SVM - Dividing dataset into train and test

```
[ ]  from sklearn.svm import SVR
     from sklearn import svm
```

```
[ ]  # To choose which kernel is the best for our data

     for k in ['linear','poly','rbf','sigmoid']:
         clf = svm.SVR(kernel=k)
         clf.fit(X_train, y_train)
         confidence = clf.score(X_train, y_train)
         print(k,confidence)
```

```
[ ]  Svr=SVR(kernel='rbf', C=1, gamma= 0.5)
```

```
[ ]  Svr.fit(X_train,y_train)
     print(Svr.score(X_train,y_train))

     0.409626446617971
```

```
[ ]  y_pred = Svr.predict(X_test)
     print(y_pred)
```

Figure 13: SVM - Choosing kernel and building model

```
[ ]  #Calculating the Evaluation Metrics

     from math import sqrt
     from sklearn import metrics

     maeerror = sqrt(metrics.mean_absolute_error(y_test,y_pred)) #calculate MAE
     print('MAE value of the SVR Model is:', maeerror)
     rmseerror = sqrt(metrics.mean_squared_error(y_test,y_pred)) #calculate rmse
     print('RMSE value of the SVR Model is:', rmseerror)
     mapeerror = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
     print('MAPE value of the SVR Model is:', mapeerror)
```

Figure 14: SVM - Calculating Evaluation Metrics