National College of Ireland

Configuration Manual

Analysis and predictions of CO_2 emissions using Neural Networks

Msc Data Analytics

Jeet Jaikishan Vyas

Student ID: x19197161

School of Computing

National College of Ireland

Supervisor: Prof. Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jeet Jaikishan Vyas
Student ID:	X19197161
Programme:	Msc Data Analytics **Year:** 2021
Module:	Research Project
Lecturer:	……………………………………………………………………………………………………
Submission Due Date:	16/12/2021
Project Title:	Analysis and prediction of CO2 emissions using deep learning
Word Count:	1736 **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: Jeet Jaikishan Vyas

Date: 16/12/2021

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

Jeet Jaikishan Vyas
Student ID: x19197161

1 Introduction

The configuration manual document will state the required software and hardware required for the research project "Analysis and predictions of CO_2 emission using Neural Networks". The manual contains the code for the Data collection/cleaning, clustering and models.

2 System and Software Requirements

Below are the required system configurations required for the research to be conducted.

2.1 Hardware Requirements:

Processor: Intel Core i5 – 7300U CPU @ 2.30 GHz to 2.40 GHz
Storage Capacity: 1 TB (Terabyte) HDD Hard Disk
System Type: 64-bit processor, x64
GPU: AMD Radeon / NVIDIA GeForce
Operating System: Windows 10 or 11 (64-bit operating system)
Ram: 8 GB

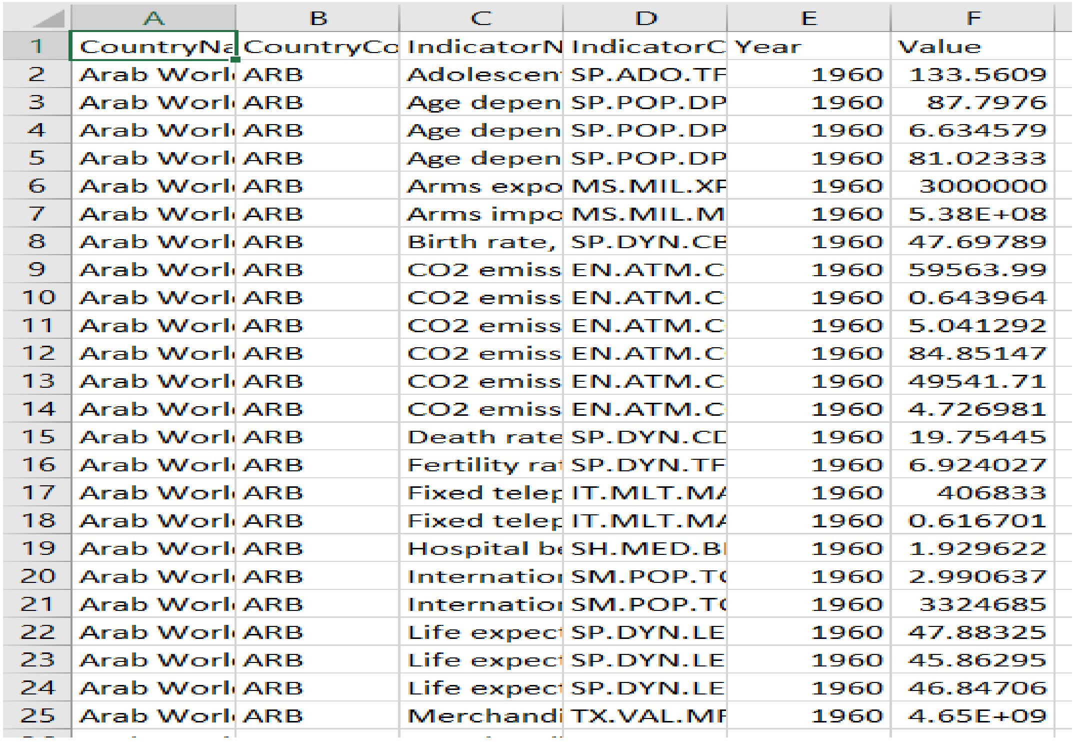
2.2 Software Requirements:

Programming and Data loading / Processing / Cleaning / Modelling: Jupyter Notebook by Anaconda.
Clustering: RStudio
Visualizations: Python / PowerBI.
Other tools: Microsoft word, Snipping tool, Microsoft excel.
Microsoft word are used for creation of tables and showcasing the figures. Snipping tool is used for getting the screenshot. Microsoft excel contains the data used for the research.

3 Data collection and cleaning

The flow of the research project is showcased below:

3.1 Collection of data and cleaning
The data used for the research in the format of CSV file has been presented. This showcases the raw data which has been used for the research implementation.



	A	B	C	D	E	F
1	CountryNa	CountryCo	IndicatorN	IndicatorC	Year	Value
2	Arab Worl	ARB	Adolescen	SP.ADO.TF	1960	133.5609
3	Arab Worl	ARB	Age depen	SP.POP.DP	1960	87.7976
4	Arab Worl	ARB	Age depen	SP.POP.DP	1960	6.634579
5	Arab Worl	ARB	Age depen	SP.POP.DP	1960	81.02333
6	Arab Worl	ARB	Arms expo	MS.MIL.XF	1960	3000000
7	Arab Worl	ARB	Arms impo	MS.MIL.M	1960	5.38E+08
8	Arab Worl	ARB	Birth rate,	SP.DYN.CE	1960	47.69789
9	Arab Worl	ARB	CO2 emiss	EN.ATM.C	1960	59563.99
10	Arab Worl	ARB	CO2 emiss	EN.ATM.C	1960	0.643964
11	Arab Worl	ARB	CO2 emiss	EN.ATM.C	1960	5.041292
12	Arab Worl	ARB	CO2 emiss	EN.ATM.C	1960	84.85147
13	Arab Worl	ARB	CO2 emiss	EN.ATM.C	1960	49541.71
14	Arab Worl	ARB	CO2 emiss	EN.ATM.C	1960	4.726981
15	Arab Worl	ARB	Death rate	SP.DYN.CD	1960	19.75445
16	Arab Worl	ARB	Fertility ra	SP.DYN.TF	1960	6.924027
17	Arab Worl	ARB	Fixed telep	IT.MLT.MA	1960	406833
18	Arab Worl	ARB	Fixed telep	IT.MLT.MA	1960	0.616701
19	Arab Worl	ARB	Hospital be	SH.MED.BI	1960	1.929622
20	Arab Worl	ARB	Internation	SM.POP.TO	1960	2.990637
21	Arab Worl	ARB	Internation	SM.POP.TO	1960	3324685
22	Arab Worl	ARB	Life expec	SP.DYN.LE	1960	47.88325
23	Arab Worl	ARB	Life expec	SP.DYN.LE	1960	45.86295
24	Arab Worl	ARB	Life expec	SP.DYN.LE	1960	46.84706
25	Arab Worl	ARB	Merchandi	TX.VAL.MF	1960	4.65E+09

Figure 3.1: Raw data Indicators

To know about the data being used for the research, Figure 1 shows the comma separated (CSV) file for the indicators data. There are a total of 6 columns and 5656458 rows in the data. The data includes the countries name and code, indicators name and code, years and the value for the indicators.

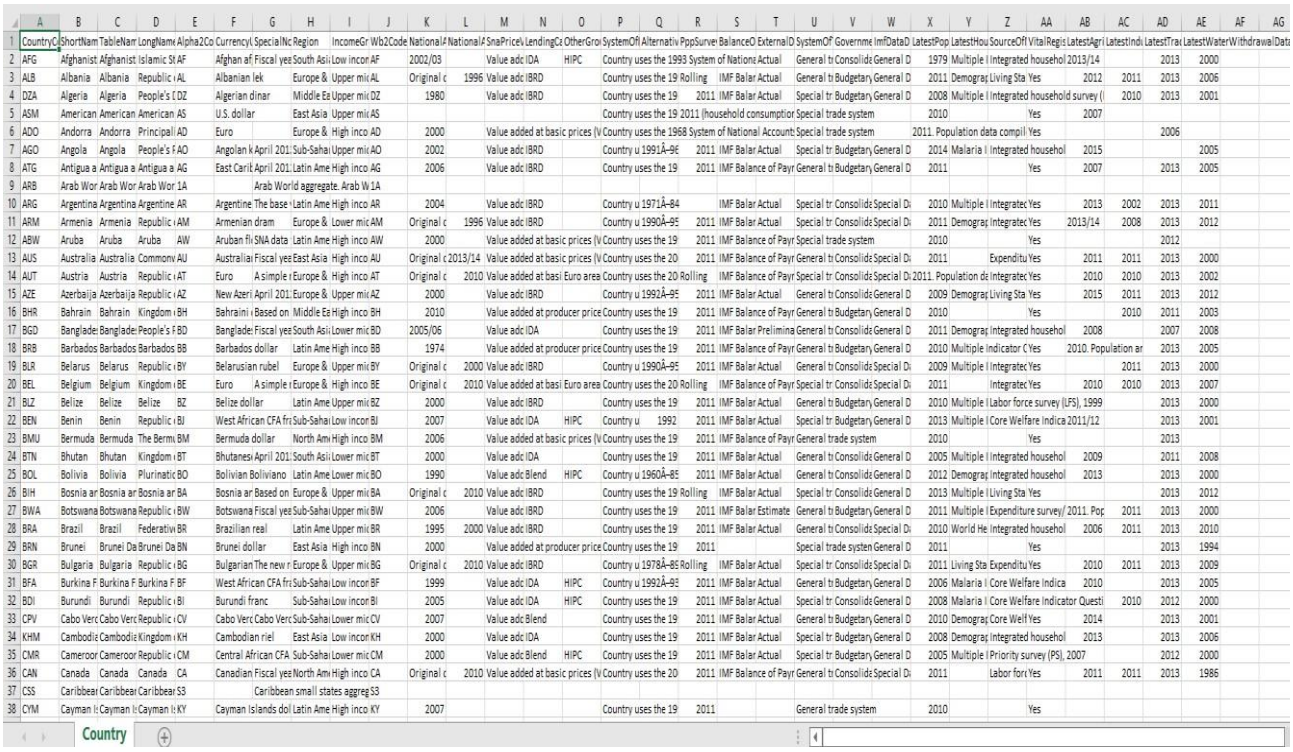


Figure 3.2: Raw Data Countries

The data about the countries has been shown in Fig 2. The data contains 247 rows and 31 columns. The information about all the countries is distinguished in the data. The country names in three different columns, currency units for the countries, encoded country names, the population census and the industrial, water withdrawal, agricultural data. The total data shows the information that are relevant for a country.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df_itr = pd.read_csv('E:\\NCI\\Semester 3\\Thesis\\Dataset\\Indicators.csv')
df_ctrs = pd.read_csv("E:\\NCI\\Semester 3\\Thesis\\Dataset\\Country.csv")
df_series = pd.read_csv("E:\\NCI\\Semester 3\\Thesis\\Dataset\\Series.csv")
```

Figure 3.3: Loading Libraries and Data Frame Creation

The raw data was loaded on the python environment by anaconda. The tables indicators, country and series were selected for the research. For the research, indicators and country data is being considered. The series data has been just used for information purposes. The pandas library in python used for data manipulation and analysis. The data should be loaded into data frames.

```
df_ctrs.isna().sum()
```

```
CountryCode                                    0
ShortName                                      0
TableName                                      0
LongName                                       0
Alpha2Code                                     3
CurrencyUnit                                  33
SpecialNotes                                  83
Region                                        33
IncomeGroup                                   33
Wb2Code                                        1
NationalAccountsBaseYear                      42
NationalAccountsReferenceYear                193
SnaPriceValuation                             49
LendingCategory                              103
OtherGroups                                  188
SystemOfNationalAccounts                      33
AlternativeConversionFactor                  200
PppSurveyYear                                 56
BalanceOfPaymentsManualInUse                  66
ExternalDebtReportingStatus                  123
SystemOfTrade                                 47
GovernmentAccountingConcept                   86
ImfDataDisseminationStandard                  64
LatestPopulationCensus                        34
LatestHouseholdSurvey                        100
SourceOfMostRecentIncomeAndExpenditureData    89
VitalRegistrationComplete                    135
LatestAgriculturalCensus                     105
LatestIndustrialData                         134
LatestTradeData                               61
LatestWaterWithdrawalData                     67
dtype: int64
```

Figure 3.4: Data cleaning for country data

```
1  df_itr.isna().sum()
```

```
CountryName      0
CountryCode      0
IndicatorName    0
IndicatorCode    0
Year             0
Value            0
```

Figure 3.5: Sum of null values in indicators data.

3.1.1 Data Cleaning

To ensure the there are no null values in the data data cleaning techniques were implemented. The pandas library allows calculating the null values and filling the null values. Figure 3.3 shows the representation. The data frame was stored in a new data frame named countries which will be used for further research steps. On the other hand, after implementation there were no null values found in the indicators data.

```
1  df_itr.duplicated()
```

```
0          False
1          False
2          False
3          False
4          False
           ...
5656453    False
5656454    False
5656455    False
5656456    False
5656457    False
Length: 5656458, dtype: bool
```

```
1  df_countries.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
242    False
243    False
244    False
245    False
246    False
Length: 247, dtype: bool
```

```
1  df_series.duplicated()
```

```
0       False
1       False
2       False
3       False
```

Figure 3.5: Duplicated values evaluation on the data

In order to find any duplicates in the data, the duplicated function shows if there are any duplicates in the data.

4 Data Visualisations

For the research, there were bar graphs visualized using the matplotlib library in python. Three indicators were chosen for this step. The indicators were CO2 emissions per metric tons per capita, Age dependency ratio, CO2 emissions from liquid fuel consumptions. Visualisations for 6 countries were implemented for the research. The countries for this are United states of America, Australia, Ireland, India, China and Ireland. The graphs showed the trend of the indicators from the year 1960 – 2010. The behaviour of the indicators according to the countries shows the flow. The indicators have been named and the plot function for bar graphs should be implemented. The world data shows the overall values of the specified indicator.

The following images show the graphical representations of overall world for the three indicators, the representations for the 6 countries including the world are visualized in python.

```
indicator1 = 'CO2 emissions \(metric'
world0 = 'World'

x = df_process['IndicatorName'].str.contains(indicator1)
y = df_process['CountryName'].str.contains(world0)

total = df_process[x & y]

x = total['Year'].values
y = total['Value'].values
plt.title("CO2 Emissions metric ton per capita  - World")
plt.xlabel("Years")
plt.ylabel("Emission range")
plt.bar(x,y)
```

Figure 4.1: CO2 emissions metric tons per capita – World

The trends of CO2 emissions metric tons per capita had seen a fluctuating one. From 1960 there has been a rise in the emissions up to 1980. There was slight fall after which the trend was quite stable and a rise in the emissions were seen after 2000 which kept rising until 2010.

```
indicator2 = 'Age dependency ratio'
world0 = 'World'

x1 = df_process['IndicatorName'].str.contains(indicator2)
y1 = df_process['CountryName'].str.contains(world0)

total = df_process[x1 & y1]

x1 = total['Year'].values
y1 = total['Value'].values
plt.title("Age dependency ratio - World")
plt.xlabel("Years")
plt.ylabel("Total %")
plt.bar(x1,y1)
plt.show()
```

Figure 4.2: Age dependency Ratio – World

The age dependency ratio follows the formula for the number of people engaged in the working class compared to the people not engaged in the working class. The people distinguished in the working population are between the age group of 18 – 59 comparing the people in age group of 0 – 17 and 59 and above. The classes are defined as minors and senior citizens or retired individuals. The indicator should be considered as an effect on the economy of an individual country.

```
indicator3 = 'CO2 emissions from liquid fuel consumption'
world0 = 'World'

x2 = df_process['IndicatorName'].str.contains(indicator3)
y2 = df_process['CountryName'].str.contains(world0)

total = df_process[x2 & y2]

x2 = total['Year'].values
y2 = total['Value'].values
plt.title("CO2 Emissions liquid fuel consumption - World")
plt.xlabel("Years")
plt.ylabel("Emission range")
plt.bar(x2,y2)
plt.show()
```

Figure 4.3: CO2 emissions from liquid fuel consumptions – world

The code showcases the indicator liquid fuel consumptions for the world. The overall trends represent the indicator name, country name includes the world and the value for the same.

5 Clustering

```r
library(readr)
library(reshape)
library(cluster)

Indicators <- read.csv("E:\\NCI\\Semester 3\\Thesis\\Dataset\\Indicators.csv")

code <- c('SE.XPD.CTOT.ZS','GB.XPD.RSDV.GD.ZS','SE.ADT.LITR.ZS','EG.ELC.ACCS.ZS',
          'SL.UEM.LTRM.ZS','SL.UEM.TOTL.ZS','FB.CBK.BRCH.P5','GC.DOD.TOTL.GD.ZS','EN.ATM.CO2E.PC',
          'BX.KLT.DINV.WD.GD.ZS','NY.GDP.PCAP.CD','NY.GNP.MKTP.CD','FP.CPI.TOTL.ZG','NY.ADJ.NNTY.PC.CD'
          'PA.NUS.FCRF','DT.DOD.PVLX.GN.ZS','IC.REG.DURS','FI.RES.TOTL.DT.ZS','SH.MED.BEDS.ZS',
          'SP.POP.1564.TO.ZS','TX.VAL.TECH.CD','SP.DYN.CDRT.IN','SP.DYN.CBRT.IN','SI.POV.GINI',
          'IC.BUS.EASE.XQ','IT.NET.USER.P2','IT.CEL.SETS.P2','IQ.CPA.TRAD.XQ','TM.VAL.FUEL.ZS.UN',
          'TX.VAL.MANF.ZS.UN','NE.TRD.GNFS.ZS')

indic <- cast(Indicators, CountryName ~ IndicatorCode, mean)
indic <- indic[,c('CountryName',code)]
row.names(indic) <- indic[,1]
indic <- indic[,c(-1,-2,-4,-6,-9,-14,-17,-19,-25,-29)]
indic <- na.omit(indic)
Cnames <- rownames(indic)

indic.norm <- sapply(indic, scale)
row.names(indic.norm) <- Cnames

set.seed(42)
indic_cluster <- kmeans(indic.norm, 3)
indic_cluster$size

par(mfrow=c(1,1))
clusplot(indic.norm,indic_cluster$cluster,color = TRUE, shade = TRUE, labels = 2, lines = 0)

indic.norm <- data.frame(cbind(indic_cluster$cluster, rownames(indic), indic.norm))
colnames(indic.norm) <- c("Cluster_No","CountryName",colnames(indic.norm)[c(-1,-2)])
rownames(indic.norm) <- NULL

developed_countries <- subset(indic.norm, Cluster_No == 1, select = CountryName)
developing_countries <- subset(indic.norm, Cluster_No == 2, select = CountryName)
underdeveloped_countries <- subset(indic.norm, Cluster_No == 3, select = CountryName)

print(developed_countries)
print(developing_countries)
print(underdeveloped_countries)
```

Figure 5: Clustering of Countries

The aim for clustering was to develop clusters of the countries in the indicators data. For this necessary step were carried in RStudio. The libraries for conducting are readr, reshape and cluster. The readr library helps in reading the CSV format files, reshape library helps in reshapes a data frame between wide and long formats in order to achieve repeated measurements in separate columns. The cluster library helps in getting the clusters using the algorithms on a specific data. The indicator data loaded in the environment as the first initial step. The indicator codes should be taken into consideration for creation of clusters. 31 indicators were chosen for the same. The transformation of data carried out by changing the formation of rows to columns. The cast function used to get the aggregate data with the aggregate function and formula. In this case, the relationship between the columns using the tilde function. The transformation makes the data ready to standardise and apply the algorithm for clustering. The sapply function takes the data frame as an input and the potential outcome will be a vector or matrix, therefore the data (indic) has been input as an object along with the function of scale. for the same to get a vector or matrix.
The K means clustering algorithm has been used for the cluster implementation. As KMeans being the most simple and quick clustering algorithm as it creates clusters with the mean

value close to the one feature to another. This helps in making a sophisticated form of clustering. Three clusters were formed as the KMean value was set at 3. Plotting the cluster created three clusters and the information about the clusters with the countries in the appropriate countries will be achieved.

6 Time Series and Models

```
from tqdm import tqdm
from time import time
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as f
```

```
from torch.utils.data import Dataset, DataLoader, TensorDataset
```

Figure 6: Libraries and utils

The libraries required to get the models using Pytorch. nn is the module for neural network models. Optim is the module which helps in optimizing the models deployed. Nn functional involves all functions which are applied on models. Tqdm library is used to show the progress bar.

```
def get_valid_range(country_code, ECON_COLS, CO2_COL, time_step=5, verbose=False):

    try:
        x = COUNTRIES_DATA[country_code].loc[ECON_COLS + [CO2_COL], :]
    except KeyError:
        if verbose:
            print('{} - {} miss the following indicators:\n'.format(country_code, CODE_TO_NAMES[country_code]),
                  ', '.join(set(ECON_COLS + [CO2_COL]) - set(COUNTRIES_DATA[country_code].index)))
        return [], None

    y_econ = (x.loc[ECON_cols, :].isna().sum(axis=0) == 0)['Value']
    y_co2  = ~x.loc[CO2_COL, :].isna()['Value']

    has_co2 = [year for year in y_co2[y_co2].index]
    for interval in continuous_subarrays(has_co2, step=1, min_len=time_step):
        y_co2.iloc[max(0, interval[0] - time_step):interval[0]] = True

    y = y_econ & y_co2

    is_valid = [year if y[year] else 0 for year in y.index]
    res = continuous_subarrays(is_valid, step=1, min_len=time_step)
    if verbose and len(res) == 0:
        print('{} - {} has no valid interval.'.format(country_code, CODE_TO_NAMES[country_code]))

    return res, x
```

Figure 6: Time series Data processing

In order to get the data ready for implementation of time series, a valid range of data need to be prepared which will withstand the factors while implementing the time series. Here, the country code (countries data), ECON_COLS are the economic indicator columns, CO2_COL are the CO2 emission indicator columns which are the target features for implementation. There has to be a set time step that will consider the time interval for the time series. Any null values have been ignored. A continuous subarray created for validating the time step with the length of the same. The countries data and time step were returned.

```python
def CO2TimeSeries(ECON_COLS, CO2_COL, time_step, valid_pct=0.2):

    train_data, val_data = None, None
    train_targets, val_targets = np.array([]), np.array([])

    for country in df_process['CountryCode'].unique():

        valid_ranges, df_countries = get_valid_range(country, ECON_COLS, CO2_COL, time_step)
        if len(valid_ranges) == 0:
            continue

        for indicator in df_countries.index:
            if indicator != CO2_COL:
                df_countries.loc[indicator, :] = \
                    (df_countries.loc[indicator, :] - indicators_mean[indicator]) / indicators_std[indicator]
            else:
                df_countries.loc[indicator, :] = [np.log(num) if not np.isnan(num) else np.nan \
                                                  for num in df_countries.loc[indicator, :]]

                country_data, country_targets = None, np.array([])
        econ_df, co2_df = df_countries.loc[ECON_COLS, :], df_countries.loc[CO2_COL, :]

        for interval in valid_ranges:
            for i in range(interval[0], interval[1] - time_step + 1):
                obs = econ_df.iloc[:, i:i+time_step].to_numpy()
                obs = obs[np.newaxis,:, :]
                if country_data is not None:
                    country_data = np.concatenate((country_data, obs), axis=0)
                else:
                    country_data = obs
                country_targets = np.append(country_targets, co2_df.iloc[i + time_step - 1])

        num_train = round((1 - valid_pct) * country_data.shape[0])
        country_train_data = country_data[:num_train,:,:]
        country_train_targets = country_targets[:num_train]
        country_val_data = country_data[num_train:,:,:]
        country_val_targets = country_targets[num_train:]

        assert country_data.shape[0] == len(country_targets)
        assert len(country_val_targets) + len(country_train_targets) == country_data.shape[0]
        assert len(country_val_targets) == country_val_data.shape[0]
        assert len(country_train_targets) == country_train_data.shape[0]

        if train_data is not None:
            train_data = np.concatenate((train_data, country_train_data), axis=0)
            val_data = np.concatenate((val_data, country_val_data), axis=0)
        else:
            train_data = country_train_data
            val_data = country_val_data
        train_targets = np.append(train_targets, country_train_targets)
        val_targets = np.append(val_targets, country_val_targets)

    assert train_data is not None,
    return TensorDataset(torch.tensor(train_data), torch.tensor(train_targets)),\
           TensorDataset(torch.tensor(val_data), torch.tensor(val_targets))
```

Figure 6.1: Time Series Implementation

Implementing the time series with the indicators and countries data. The train and validation were created using arrays in numpy library. The range of indicators if not null for economic and CO2 emissions will be considered. The second range of countries in valid range are not null then the data will concatenate else it will append. The same process was carried out to prepare the training and validation implementation.

```
train_ds, val_ds = CO2TimeSeries(ECON_cols, CO2_COL='EN.ATM.CO2E.KT', time_step=TIME_STEP, valid_pct=VALID_PCT)
print("Number of Training observations:", len(train_ds))
print("Number of Validation observations:", len(val_ds))

BATCH_SIZE = 32

train_dl = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
val_dl = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=True)
```

Figure 6.2: Time series training and validation

The training and validation for the time series with the economic indicators and CO2 emissions metric ton per capita for CO2_COL with the valid time step declared in the implementation phase. The batch size set to 32 shows the number of samples to be shown to the networks. The batch size used to fit the model and control the number of predictions made at a certain time.

```
class LSTM(nn.Module):

    def __init__(self, input_dims, hidden_dims, output_dims,):
        super().__init__()
        self.rec_layer = nn.LSTM(input_dims, hidden_dims, batch_first=True)
        self.fc = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):

        x = self.reshape(x)
        outputs, (hidden_states, cell_states) = self.rec_layer(x)
        outputs = outputs[:, -1, :]
        outputs = outputs.squeeze(1)
        outputs = self.fc(outputs)

        return outputs.squeeze(1)

    def reshape(self, x):
        return torch.reshape(x, shape=(x.shape[0], x.shape[2], x.shape[1]))
```

Figure 6.3: 1Layer LSTM Model

```
class GRU(nn.Module):

    def __init__(self, input_dims, hidden_dims, output_dims,):
        super().__init__()
        self.rec_layer = nn.GRU(input_dims, hidden_dims, batch_first=True)
        self.fc = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):

        x = self.reshape(x)
        outputs, hidden_states = self.rec_layer(x)
        outputs = outputs[:, -1, :]
        outputs = outputs.squeeze(1)
        outputs = self.fc(outputs)

        return outputs.squeeze(1)

    def reshape(self, x):
        return torch.reshape(x, shape=(x.shape[0], x.shape[2], x.shape[1]))
```

Figure 6.3: Gated Recurring Unit (GRU)

```python
class FeedForward(nn.Module):

    def __init__(self, input_dims, time_step):
        super().__init__()

        # 5 layers
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(input_dims * time_step, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 256)
        self.fc4 = nn.Linear(256, 128)
        self.fc5 = nn.Linear(128, 1)

        self.activation = nn.ReLU()

    def forward(self, x):

        x = self.flatten(x)
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.activation(self.fc3(x))
        x = self.activation(self.fc4(x))
        x = self.fc5(x)

        return
```

Figure 6.4: Feed Forward Model creation

Three models were created 1 layer LSTM, GRU and Feed Forward. They were obtained form pytorch in the neural network. All models are recurrent models only LSTM and GRU were created a with the required dimensions for the models. The last hidden layer is grabbed and getting a reshaped form of hidden and output batch. The feed forward model was created with the linear functions consisting 4 layers and a dimension with inputs along with the time step. The observations are flattened.

```python
import torch.backends.cudnn as cudnn
torch.cuda.empty_cache()
cudnn.benchmark = True

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```python
lstm_model = LSTM(input_dims=len(ECON_cols), hidden_dims=32, output_dims=1).to(device)
lstm_optimizer = optim.Adam(
    lstm_model.parameters(),
    lr=0.001
)
lstm_loss_fn = torch.nn.MSELoss()
```

```python
train_dl = train_dl
train_model = lstm_model
train_optimizer = lstm_optimizer
train_loss_fn = lstm_loss_fn
```

```python

```

```python
gru_model = GRU(input_dims=len(ECON_cols), hidden_dims=32, output_dims=1).to(device)
gru_optimizer = optim.Adam(
    gru_model.parameters(),
    lr=0.001
)
gru_loss_fn = torch.nn.MSELoss()
```

```python
train_dl = train_dl
train_model = gru_model
train_optimizer = gru_optimizer
train_loss_fn = gru_loss_fn
```

The cudnn benchmark is set true for hardware validation. CPU has been set for carrying out the models to run the dimensions.
The LSTM and GRU models are trained and optimized with calculating the loss using the loss function. The loss function will calculate the MSE loss.

```
train_np_x = np.stack([obs for obs, label in train_ds], axis=0)
train_np_y = np.stack([label for obs, label in train_ds])
val_np_x = np.stack([obs for obs, label in val_ds], axis=0)
val_np_y = np.stack([label for obs, label in val_ds])

train_np_x = train_np_x.reshape((train_np_x.shape[0], -1))
val_np_x = val_np_x.reshape((val_np_x.shape[0], -1))
```

```
xgb_model = XGBRegressor()
xgb_model.fit(train_np_x, train_np_y)
```

Figure 6.5: Random Forest with Gradient Boosting

The training and validation dimensions have been converted to stacks using numpy. They are reshaped and fitted in the model. The results forecasted using matplotlib and box plots should give the validation loss for the number of observations.

References

[1] Adi Bronstein (2017) *A quick introduction to the "Pandas" Python Library* Available at: https://towardsdatascience.com/a-quick-introduction-to-the-pandas-python-library-f1b678f34673 [25 October 2021].

[2] Jason Brownlee (2017) *Stacked Long Short-term Memory Networks* Available at: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/ [10 November 2021]

[3] Tasnuva Zaman (2019) Building Neural Network Using Pytorch Available at: https://towardsdatascience.com/building-neural-network-using-pytorch-84f6e75f9a [12 November 2021]

[4] GeeksforGeeks (2020) *K-Means Clustering in R programming* Available at: https://www.geeksforgeeks.org/k-means-clustering-in-r-programming/ [23 October 2021]