Configuration Manual

MSc Research Project
Data Analytics

Ruchita Patil
Student ID: 19197411

School of Computing
National College of Ireland

Supervisor: Bharathi Chakravarthi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ruchita Patil
Student ID:	19197411
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Bharathi Chakravarthi
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	756
Page Count:	9

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:	Ruchita Patil
Date:	20th September 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

Ruchita Patil
19197411

1 Introduction

This document explains what hardware and software are necessary for this project. Implementation of the project, which includes libraries, data preparation, and modeling in python. In this research, we used a variety of models to compare them and calculate the error rate.

2 System Configuration

2.1 Hardware requirement

Table 1: Device Specification

Hardware	**Configuration**
System	HP Pavilion Laptop
System type	64-bit operating system
RAM	8 GB
SSD	256 GB
Processor	Intel(R) core(TM) i5-1035G1
CPU	1 GHz

Table 2: Windows Specification

Parameter	**Configuration**
Edition	Windows 10 Home Single Language
Version	21H1
OS build	19043.1110

2.2 Software requirement

For this research, we used a variety of methods to obtain data in a csv file. The python language was used to clean and prepare the data, and it was ran on the Google Colab tool. In Google Colab, compare the results and plot some graphs.

Table 3: Software requirement

Software	**Version**
Python	3.7(64bit)
Microsoft Excel	2020

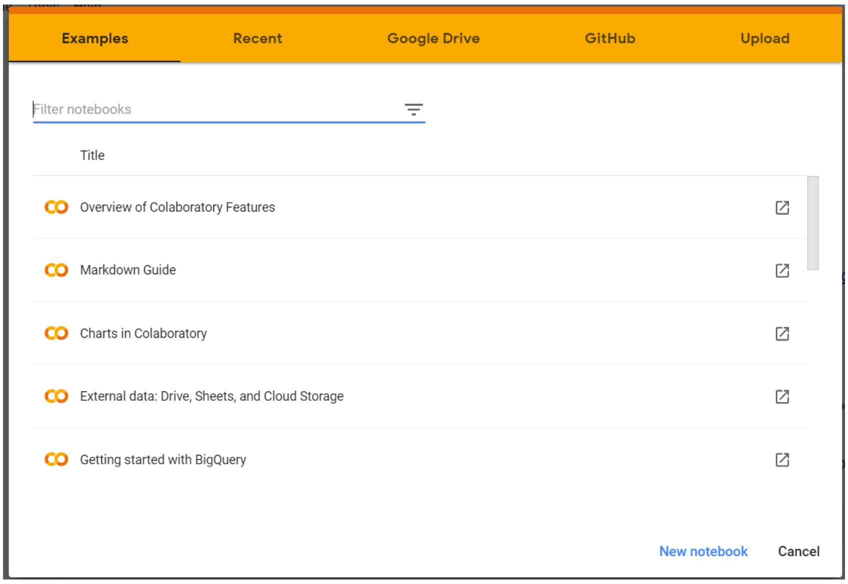


Figure 1: Google Colab website

3 Project Implementation

3.1 Data Collection

Figure 2: Central pollution control board

In India, air quality is monitored using Air Quality Index data, and monitoring stations are set up, with data updated hourly and daily on the Central Pollution Control Board's website. I acquired the data from Kaggle, which came from the CPCB's website. They provide data by city and station. I chose city-level data from a CSV file for prediction purposes(Yousefi and Hadei; 2019).

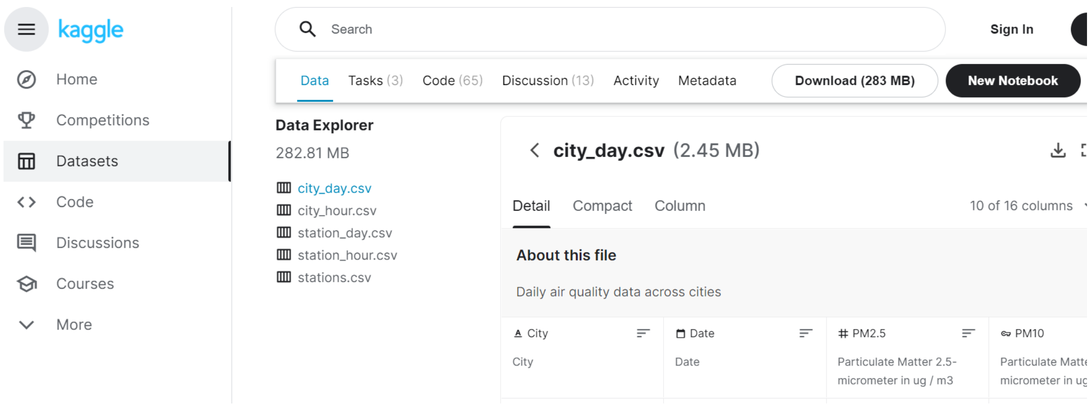
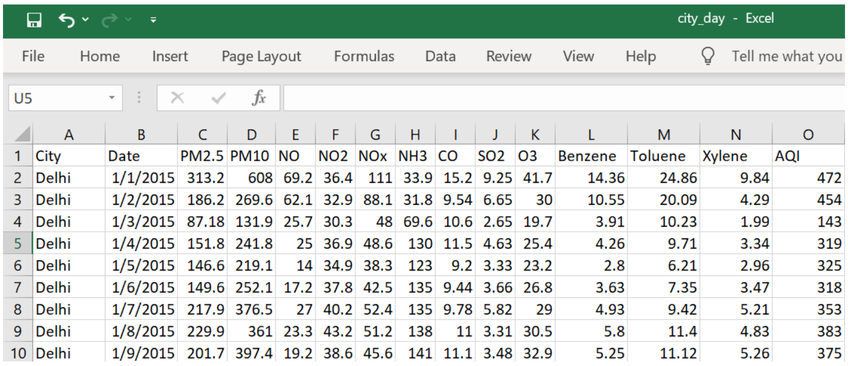


Figure 3: Dataset website

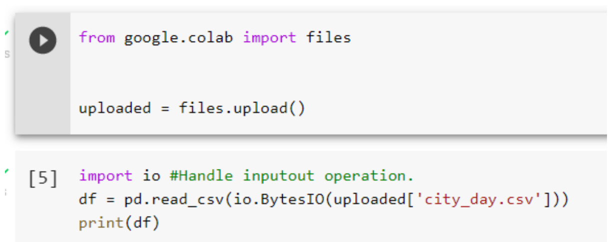
3.2 Data Preparation

The information is already in CSV format. Using python code, upload the file to Google Colab and read the csv file. After you've chosen the right data, combine it all into one parameter.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI
2	Delhi	1/1/2015	313.2	608	69.2	36.4	111	33.9	15.2	9.25	41.7	14.36	24.86	9.84	472
3	Delhi	1/2/2015	186.2	269.6	62.1	32.9	88.1	31.8	9.54	6.65	30	10.55	20.09	4.29	454
4	Delhi	1/3/2015	87.18	131.9	25.7	30.3	48	69.6	10.6	2.65	19.7	3.91	10.23	1.99	143
5	Delhi	1/4/2015	151.8	241.8	25	36.9	48.6	130	11.5	4.63	25.4	4.26	9.71	3.34	319
6	Delhi	1/5/2015	146.6	219.1	14	34.9	38.3	123	9.2	3.33	23.2	2.8	6.21	2.96	325
7	Delhi	1/6/2015	149.6	252.1	17.2	37.8	42.5	135	9.44	3.66	26.8	3.63	7.35	3.47	318
8	Delhi	1/7/2015	217.9	376.5	27	40.2	52.4	135	9.78	5.82	29	4.93	9.42	5.21	353
9	Delhi	1/8/2015	229.9	361	23.3	43.2	51.2	138	11	3.31	30.5	5.8	11.4	4.83	383
10	Delhi	1/9/2015	201.7	397.4	19.2	38.6	45.6	141	11.1	3.48	32.9	5.25	11.12	5.26	375

Figure 4: Data into CSV format



```
from google.colab import files

uploaded = files.upload()
```

```
[5] import io #Handle inputout operation.
    df = pd.read_csv(io.BytesIO(uploaded['city_day.csv']))
    print(df)
```

Figure 5: Read CSV file

After read the data python provide various function to check data. Head function provide first five records of the data. info function provide the details about each variables such as data type and size of each variables. Also shape we used for check the array size, how many rows and columns are present in the data.

Some libraries are used for preparation of data:

Table 4: Python libraries

Library	Description
Pandas	Read csv file
Pandas	To change date format
IO	Handle input/output operation

```
[ ] # Data description
    df.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 29531 entries, 0 to 29530
    Data columns (total 16 columns):
     #   Column      Non-Null Count  Dtype
    ---  ------      --------------  -----
     0   City        29531 non-null  object
     1   Date        29531 non-null  object
     2   PM2.5       24933 non-null  float64
     3   PM10        18391 non-null  float64
     4   NO          25949 non-null  float64
     5   NO2         25946 non-null  float64
     6   NOx         25346 non-null  float64
     7   NH3         19203 non-null  float64
     8   CO          27472 non-null  float64
     9   SO2         25677 non-null  float64
     10  O3          25509 non-null  float64
```

Figure 6: Check data type of all parameters

All variables' data types should be checked. We need to cast datatype to Date format in our situation because Date is an index variable with an object data type.

```
[ ] df['Date'] = pd.to_datetime(df['Date'], infer_datetime_format=True)

[ ] Delhi= df.loc[df['City'] == 'Delhi']

    Delhi.info()

    <class 'pandas.core.frame.DataFrame'>
    Int64Index: 2009 entries, 10229 to 12237
    Data columns (total 16 columns):
     #   Column      Non-Null Count  Dtype
    ---  ------      --------------  -----
     0   City        2009 non-null   object
     1   Date        2009 non-null   datetime64[ns]
     2   PM2.5       2007 non-null   float64
     3   PM10        1932 non-null   float64
     4   NO          2007 non-null   float64
     5   NO2         2007 non-null   float64
     6   NOx         2009 non-null   float64
     7   NH3         2000 non-null   float64
     8   CO          2009 non-null   float64
     9   SO2         1899 non-null   float64
     10  O3          1925 non-null   float64
```

Figure 7: Change data type

3.3 Data Pre-processing

Check for null values in data pre-processing, and if any are found, replace all null values with median data obtained using the median function. Splitting the data into train and test sets is required for applying the model to the dataset.

Describe function calculate the mean, median, max, min values of the data and showing into to one table.

```
df_input.describe()
```

Figure 8: Describe code

For example:

Below figure showing calculation of each pollutant so we can modify and analysis according to results.

	AQI	PM10	PM2.5	CO	NO	NO2	NOx	NH3	SO2	O3
count	1999.000000	1932.000000	2007.000000	2009.000000	2007.000000	2007.000000	2009.000000	2000.000000	1899.000000	1925.00000
mean	259.487744	232.809229	117.196153	1.976053	38.985595	50.785182	58.567023	41.997150	15.901253	51.32361
std	119.537333	121.873025	82.912945	2.560253	33.389456	22.696721	37.690350	17.301221	7.966770	26.06234
min	29.000000	18.590000	10.240000	0.000000	3.570000	10.630000	0.000000	6.780000	2.340000	6.94000
25%	161.500000	137.040000	57.095000	0.910000	15.895000	33.895000	31.150000	31.157500	10.335000	33.71000
50%	257.000000	216.730000	94.620000	1.240000	27.200000	47.150000	52.750000	38.040000	14.450000	44.44000
75%	345.500000	311.667500	153.030000	1.870000	50.790000	63.570000	75.360000	48.792500	19.700000	60.84000
max	716.000000	796.880000	685.360000	30.440000	221.030000	162.500000	254.800000	166.700000	71.560000	257.73000

Figure 9: Describe output

```
[43] # Check missing values
     df_input.isnull().sum()

     AQI       10
     PM10      77
     PM2.5      2
     CO         0
     NO         2
     NO2        2
     NOx        0
     NH3        9
     SO2      110
     O3        84
     dtype: int64
```

Figure 10: Find null values

In describe output we get each pollutant avg, minimum, maximum values. We analyzed some pollutant have null values which is harm-full for model. There are many options to handle null values some time we drop the records, when dataset is huge because is not reflected to results.

But in out case data is not huge so we decided to used median fiction. Like median function mean is also available.

The form of the original distribution is preserved by MinMaxScaler. It has no effect on the material included in the source information. It's worth noting that MinMaxScaler doesn't lessen the significance of outliers. MinMaxScaler returns a feature with a default range of 0 to 1.

```
[44] #For missing values I used median to fill Null values.
     df_input['AQI']=df_input['AQI'].fillna((df_input['AQI'].median()))
     df_input['PM2.5']=df_input['PM2.5'].fillna((df_input['PM2.5'].median()))
     df_input['PM10']=df_input['PM10'].fillna((df_input['PM10'].median()))
     df_input['CO']=df_input['CO'].fillna((df_input['CO'].median()))
     df_input['NO']=df_input['NO'].fillna((df_input['NO'].median()))
     df_input['NO2']=df_input['NO2'].fillna((df_input['NO2'].median()))
     df_input['NOx']=df_input['NOx'].fillna((df_input['NOx'].median()))
     df_input['NH3']=df_input['NH3'].fillna((df_input['NH3'].median()))
     df_input['SO2']=df_input['SO2'].fillna((df_input['SO2'].median()))
     df_input['O3']=df_input['O3'].fillna((df_input['O3'].median()))
```

Figure 11: Fill NA values with median

```
[47] # Split train data to X and y
     X_train = train_dataset.drop('AQI', axis = 1)
     y_train = train_dataset.loc[:,['AQI']]

     # Split test data to X and y
     X_test = test_dataset.drop('AQI', axis = 1)
     y_test = test_dataset.loc[:,['AQI']]
```

```
[48] y_train.shape

     (1607, 1)
```

Figure 12: Split data into train and test

```
[49] # Transform X_train, y_train, X_test and y_test

     # Different scaler for input and output
     scaler_x = MinMaxScaler(feature_range = (0,1))
     scaler_y = MinMaxScaler(feature_range = (0,1))

     # Fit the scaler using available training data
     input_scaler = scaler_x.fit(X_train)
     output_scaler = scaler_y.fit(y_train)

     # Apply the scaler to training data
     train_y_norm = output_scaler.transform(y_train)
     train_x_norm = input_scaler.transform(X_train)

     # Apply the scaler to test data
```

Figure 13: Split data into train and test

3.4 Model building

We employed a variety of models during modeling. Python has various libraries for each model, so import all of them first. Keras is required for the LSTM and GRU models. Keras featured a variety of layers that are useful while building a model. We also utilized a dropout layer to prevent over-fitting.

Table 5: Python libraries

Library	**Description**
Tensor flow	Import keras
Keras	Import sequential, Layers , Callback
Layers	Dense, LSTM, Dropout, GRU

```
# Create LSTM or GRU model
def create_model(units, m):
    model = Sequential()
    # First layer of LSTM or GRU
    model.add(m (units = units, return_sequences = True,
                input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(Dropout(0.2))
    # Second layer of LSTM or GRU
    model.add(m (units = units))
    model.add(Dropout(0.2))
    model.add(Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    return model
```

Figure 14: LSTM and GRU model

sklearn library included linear model, from that we import linear-regression. We can build model using Linear regression function. After executing model fit train dataset into model using fit function. Predict test data using prediction function.

```
[ ]  mreg = LinearRegression()
     mreg.fit(x_train1,y_train1)

     mlr_y_predict = mreg.predict(x_test1)
     mlr_y_predict_train = mreg.predict(x_train1)
```

Figure 15: Linear Regression

In python there is in-build function for Decision tree, which included parameters so we can build model according to our requirement. For decision tree sklearn included tree library so we can import Decision tree regression file. After executing model fit train dataset into model using fit function. Predict test data using prediction function.

```
dec_tree = DecisionTreeRegressor(random_state = 0)
dec_tree.fit(x_train1,y_train1)

dt_y_predict = dec_tree.predict(x_test1)
dt_y_predict_train = dec_tree.predict(x_train1)
```

Figure 16: Decision Tree

```
# Create LSTM - GRU model
def create_model_GRU_LSTM(units):
    model = Sequential()
    # First layer-GRU
    model.add(GRU(units = units, return_sequences = True,
                 input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(Dropout(0.3))
    # Second layer-LSTM
    model.add(LSTM(units = units, return_sequences=False))
    model.add(Dropout(0.3))
    model.add(Dense(units = 64))
    model.add(Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    return model


# GRU and LSTM
model_gru = create_model(64, GRU)
model_lstm = create_model(64, LSTM)

model_gru_lstm = create_model_GRU_LSTM(128)
```

Figure 17: GRU-LSTM proposed model

3.5 Evaluation

Evaluation purpose we used various measures such as root mean square error, mean absolute error and r-square. Using metrics library we calculate all measures.

Table 6: Python libraries

Library	Description
Math	Import Square-root
sklearn	Import metrics

```
#LSTM
rmse_LSTM = sqrt(metrics.mean_squared_error(y_test, prediction_lstm_test))
mae_LSTM = metrics.mean_absolute_error(y_test, prediction_lstm_test)
mdae_LSTM = metrics.median_absolute_error(y_test,prediction_lstm_test)
```

```
[ ] #GRU
rmse_GRU = sqrt(metrics.mean_squared_error(y_test, prediction_gru_test))
mae_GRU = metrics.mean_absolute_error(y_test, prediction_gru_test)
mdae_GRU = metrics.median_absolute_error(y_test,prediction_gru_test)
```

```
[ ] #Logistic regression
rmse_mlr = sqrt(metrics.mean_squared_error(y_test1, mlr_y_predict))
mae_mlr = metrics.mean_absolute_error(y_test1, mlr_y_predict)
mdae_mlr = metrics.median_absolute_error(y_test1,mlr_y_predict)
```

```
[ ] #decsison tree
rmse_dt = sqrt(metrics.mean_squared_error(y_test1, dt_y_predict))
mae_dt = metrics.mean_absolute_error(y_test1, dt_y_predict)
mdae_dt = metrics.median_absolute_error(y_test1,dt_y_predict)
```

```
[ ] #GRU_LSTM
rmse_GRU_LSTM = sqrt(metrics.mean_squared_error(y_test, prediction_gru_lstm_test))
mae_GRU_LSTM = metrics.mean_absolute_error(y_test, prediction_gru_lstm_test)
mdae_GRU_LSTM = metrics.median_absolute_error(y_test,prediction_gru_lstm_test)
```

Figure 18: Result of measure

References

Yousefi, S., S. A. and Hadei, M. (2019). Applying epa's instruction to calculate air quality index (aqi) in tehran, *Journal of Air Pollution and Health* pp. 81–6.