Configuration Manual

MSc Research Project
MSc In Data Analytics

Ankit Singh
Student ID: x18127321

School of Computing
National College of Ireland

Supervisor: Pierpaolo Dondio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. ………… AnkitSingh……………………………………………………………………………
Student ID:	………x18127321…………………………………………………………………………………
Programme:	MSc In Data Analytics **Year:** …2019-2020.
Module:	Research Project
Lecturer:	……….
Submission Due Date:	……….
Project Title:	Air Pollution Forecasting and Performance Evaluation Using Advanced Time Series and Deep Learning Approach for Gurgaon
Word Count:	1319 **Page Count:** ……………13….……..………

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: ………………………………………Ankit Singh………………………………………

Date: 12-12-2019

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

Ankit Singh
Student ID: x18127321

1 Introduction

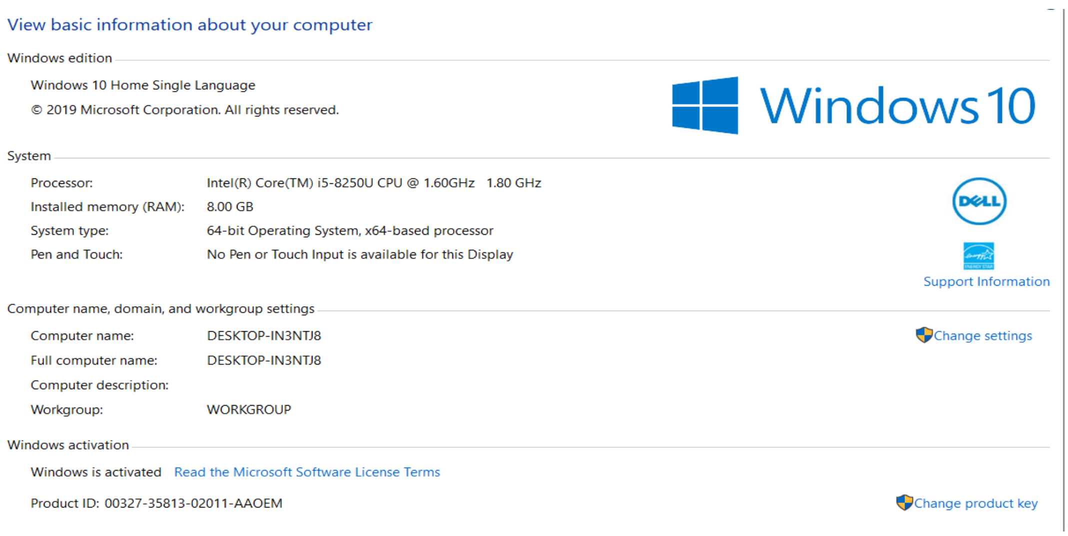
In order to control the rising crisis of air pollution, the research project focusses on the forecasting of air quality index for a north Indian city Gurgaon. The project has been implemented using several tools and software including Python Spyder from Anaconda Navigator, R Studio, Excel and Word. A total of eight forecasting models including novel Prophet have been compared as per Mean Absolute Error, Mean Squared Error, Root Mean Squared Error and Mean Absolute Percentage Error. After evaluation it has been found out that Prophet outperforms all compared models in terms of forecasting errors. The novel Prophet model also gave good performance on a new dataset of Delhi air quality.

2 System Summary

The project was implemented on a specific set of hardware and software configurations. This section mentions the system configurations used for this research.

System Configuration

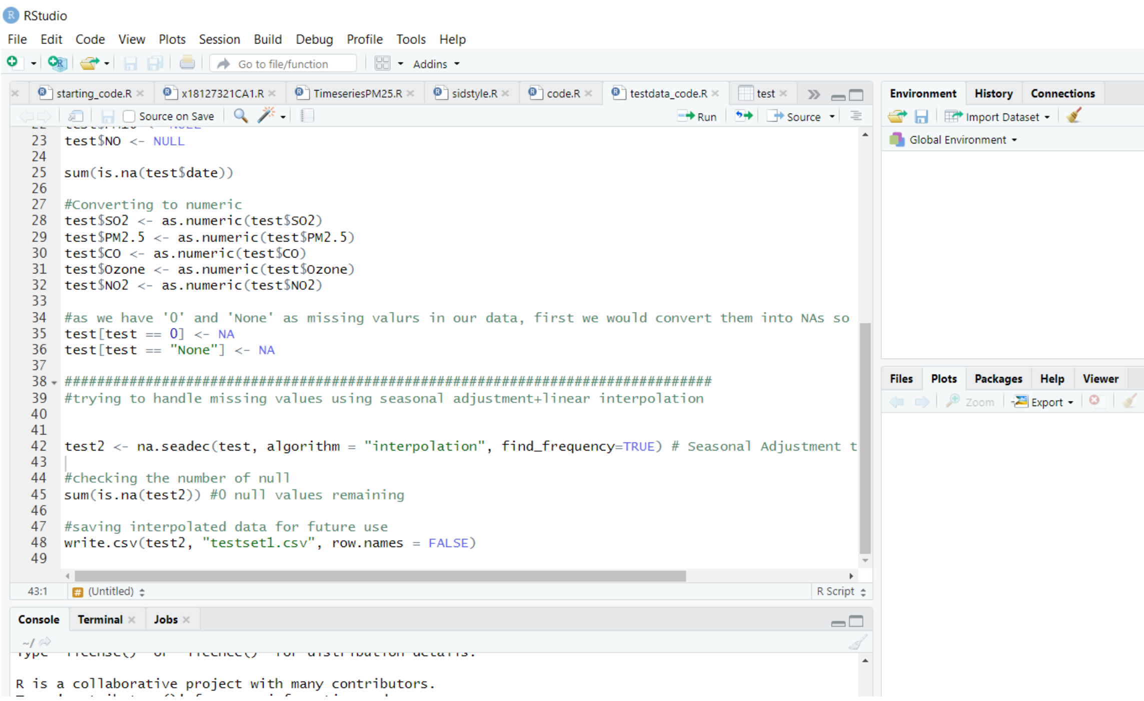
Operating System : Windows 10 – 64 bit
RAM: 8 GB
Processor: Intel i5-8250U
Hard Disk: 256 GB SSD



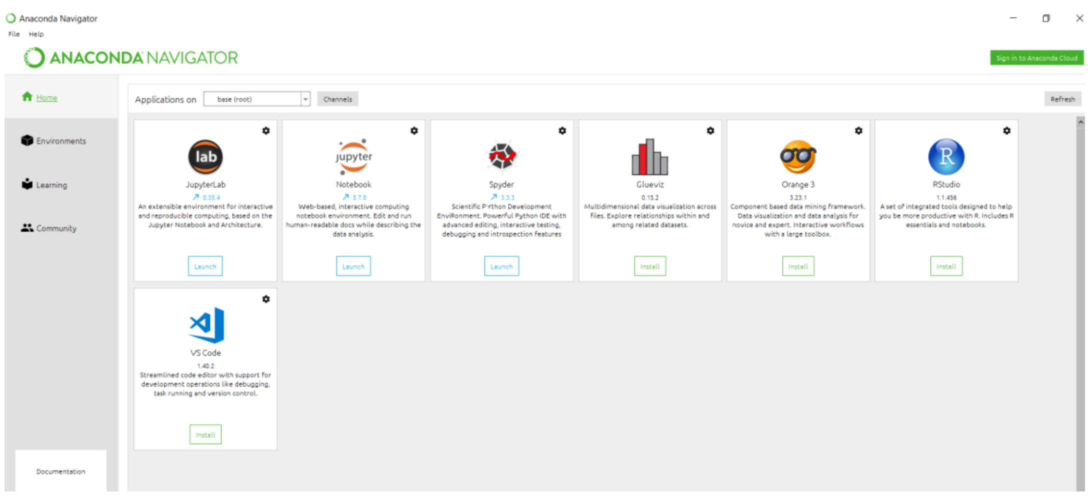
Software Used: The following tools and technologies have been used in this project :

Windows 10 : Windows 10 has been used for the project implementation. The steps for windows installation has been discussed in the following url : https://www.windowscentral.com/how-do-clean-installation-windows-10

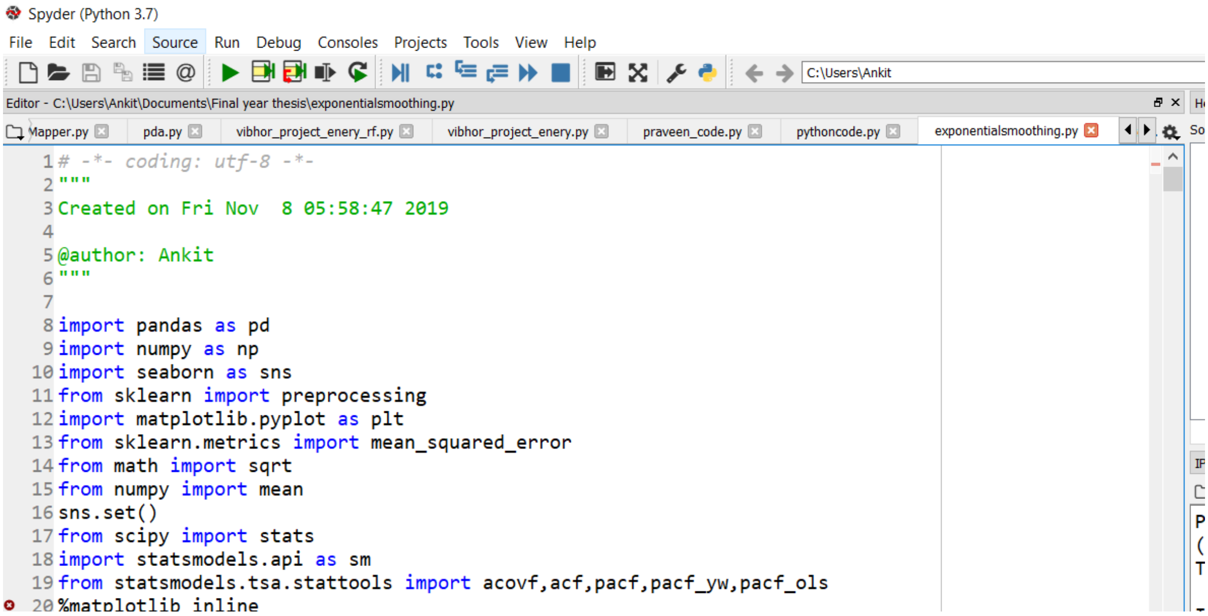
R Studio: R programming language can be used by installing an IDE called R studio. It is an open source data analytics software. It has a user friendly design with separate variable window, console and terminal. R has been used for cleaning the dataset and imputing the missing values. It is available to download from https://rstudio.com/.



Anaconda Navigator: Anaconda navigator is a GUI of collection of different IDE's such as Python Spyder, Jupyter Notebooks and R Studio. The major benefit of using anaconda navigator is that it allows to install most packages directly without using the command line.



Python (Spyder)(3.7): Spyder has been used for the data transformation, implementation and evaluation along with results. Visualisations have also been created in Spyder. It is an open source IDE for multiple platforms such as Windows and Linux. It can be directly accessed by installing Anaconda Navigator from https://docs.anaconda.com/anaconda/navigator/install/#:~:targetText=Installing%20Navigator,command%20conda%20install%20anaconda%2Dnavigator%20.



Microsoft Excel: Excel is an software used by analysts to perform multiple calculations on data as well as creating visualisations. It is available at https://products.office.com/en-ie/excel. It is not an open source software and a license is needed to fully use it. AQI calculation has been performed in Excel.

3 Process Flow of the Project

AQI Calculation: For this project, Air Quality Index has been used as the research variable. In order to calculate the AQI, the Indian AQI calculator has been used which is available at https://app.cpcbccr.com/ccr_docs/AQI%20-Calculator.xls. This calculator has been created by the Central Pollution Control Board of India. The formulas for different pollutants were taken from the calculator and used in Excel to get the sub-indices values for pollutants used.


daily_data - DataFrame

Index	SO2	CO	Ozone	PM2.5	NO2
2017-01-01…	7.98333	39.67	13.85	224.237	10.66
2017-01-02…	9.41	43.67	6.58333	344.077	10.2867
2017-01-03…	10.2467	52.7533	7.46667	288.577	10.3967
2017-01-04…	7.57	45.52	8.59333	324.537	10.5767
2017-01-05…	6.01	39.36	13.5367	258.647	10.5467
2017-01-06…	4.79	20.84	12.53	220.34	10.3233
2017-01-07…	4.94667	23.1433	3.98667	190.02	10.3267
2017-01-08…	4.22	16.0667	16.6267	180.32	10.2233
2017-01-09…	6.24333	19.7333	12.0267	176.807	10.23
2017-01-10…	3.45667	16.07	21.56	191.333	10.2533

Format Resize Background color Column min/max Save and Close Close

Calculation of AQI

Date		Station	NSIT	
DD-MM-YYYY		City	Delhi	
		State	Delhi	

Pollutants		concentration in µg/m3 (except for CO)	Sub-Index	check
PM10	24-hr avg	121.00	114	1
PM2.5	24-hr avg	34.00	57	1
SO2	24-hr avg	0.00	0	0
NOx	24-hr avg	8.00	10	1
*CO (mg/m3)	max 8-hr	0.00	0	0
O3	max 8-hr	57.00	57	1
NH3	24-hr avg	34.00	9	1

Air Quality Index

AQI = 114

* Concentrations of minimum three pollutants are required; one of them should be PM10 or PM2.5

• The check displays "1" when a non-zero value is entered

Good (0–50)	Minimal Impact	Poor (201–300)	Breathing discomfort to people on prolonged exposure
Satisfactory (51–100)	Minor breathing discomfort to sensitive people	Very Poor (301–400)	Respiratory illness to the people on prolonged exposure
Moderate (101–200)	Breathing discomfort to the people with lung, heart disease, children and older adults	Severe (>401)	Respiratory effects even on healthy people

R Studio: First, the dataset has been loaded into R and cleaning has been done. Libraries used are as follows :

```
#reading the data

library("readxl")
library("imputeTS")
#setting working directory
setwd("~/Final year thesis/Dataset")

#reading the files and skipping redundant rows
air<- read_excel("pollution.xlsx", skip = 15)

#changing the column names using the first row
colnames(air) = air[1, ] # the first row will be the header
air = air[-1, ]


names(air)[1] <- "date"
#to seperate date and time into seperate columns
air <- tidyr::separate(air, date, c("date", "time"), sep = " ")
```

1) Library(readxl) has been used for reading the excel file with function ‘read_excel’. ‘colnames’ has been used to modify the columns and ‘tidyr’ has been used to separate date and time from datetime column.

2) Library(“imputeTS”) has been used to impute missing time series datapoints
3) ‘Is.na’ function has been used to check for the null values.
4) ‘as.numeric’ has been used to convert the datatype of columns to numeric

5) 'na.seadec' function has been used to fill the missing time series values by seasonal adjustment and linear interpolation (SINGH, 2019)

```
#removing to date column
air$`To Date`<- NULL
sum(is.na(air$date))

#Converting to numeric
air$SO2 <- as.numeric(air$SO2)
air$PM2.5 <- as.numeric(air$PM2.5)
air$CO <- as.numeric(air$CO)
air$Ozone <- as.numeric(air$Ozone)
air$NO2 <- as.numeric(air$NO2)

air2 <- air

################################################################################
#trying to handle missing values using seasonal adjustment+linear interpolation

air$SO2 <- na.seadec(air$SO2, algorithm = "interpolation") # Seasonal Adjustment then Linear I
air$PM2.5 <- na.seadec(air$PM2.5, algorithm = "interpolation")
air$CO <- na.seadec(air$CO, algorithm = "interpolation")
air$Ozone <- na.seadec(air$Ozone, algorithm = "interpolation")
air$NO2 <- na.seadec(air$NO2, algorithm = "interpolation")

air2 <- na.seadec(air, algorithm = "interpolation", find_frequency=TRUE) # Seasonal Adjustment

#saving interpolated data for future use
write.csv(air2, "using_interpolation.csv", row.names = FALSE)
```

6) 'write.csv' has been used to save the cleaned dataset into a csv

The same code has been used to clean both Gurgaon and Delhi(testset) as well.

Python: After imputation of missing values in R, Python has been used for more preprocessing, transformation and implementation. (Peixeiro, 2019)

Libraries Used has been discussed below :

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  8 05:58:47 2019

@author: Ankit
"""

import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from math import sqrt
from numpy import mean
sns.set()
from scipy import stats
import statsmodels.api as sm
from statsmodels.tsa.stattools import acovf,acf,pacf,pacf_yw,pacf_ols
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from tbats import BATS, TBATS
import time

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

1) Pandas is used for data manipulation
2) Numpy is a scientific library used for scientific calculations and mathematical funcitons
3) Sklearn or scikit learn is a python library used to import various machine learning algorithm
4) Matplotlib is the basis plotting library which has been used for visualisation
5) Evaluation metrics like MSE,MAE,MAPE and RMSE has been imported from sklearn.metrics library
6) 'sqrt' and 'mean' function has been used to calculate square root and mean
7) Stasmodel library has been used for statistical plotting and tests such as acf and pacf
8) Warning has been used to ignore the warnings
9) Keras deep learning library has been used to import deep learning layers such as Sequential, Dense, LSTM and Dropout

Process Flow : The output file from R Studio has been imported into Python, and 'pd.to_datetime' function has been used to convert datetime column to the datetime format for python to understand. An user defined function 'positive_average' was created to aggregate all 8 hourly data to daily data as novel Prophet only takes daily data as an input.

```
#taking aggregate valuees to daily
def positive_average(num):
    return num[num > 0].mean()

daily_data = dataset.drop('time', axis=1).groupby('date').apply(positive_average)
daily_data.info()
```

The dataset has been saved using 'to_csv' function with the name of 'ready.csv'. Pollutant sub-indices has been calculated on this ready.csv dataset using excel and a new dataset with the name of 'ready_new' has been created and imported back to python. AQI has been calculated using 'max' function. 'corr' function has been used for correlation analysis and 'sns.heatmap' for visualising it. (Brownlee, 2019)

```
''' Checking correlation'''
corr = dataset.corr()
ax = sns.heatmap(corr, annot=True)
#the correlation between AQI and pm2.5 is .99
```

'adfuller' function has been used to perform Augmented Dickey Fuller Test

```
# ADF testing for stationarity
from statsmodels.tsa.stattools import adfuller
#dickey fuller test
adfuller(dataset['AQI'])
dftest = adfuller(dataset['AQI'])
dfout = pd.Series(dftest[0:4],index=['ADF Test Statistics','p-value','# lags used','# observations'])
for key,val in dftest[4].items():
    dfout[f'critical value ({key})'] = val

dfout
#data is coming to be stationary as p value is less than .05 and test stats is even lower than the crit
```

‘kpss’ function has been used to perform kpss test of stationarity

```
#KPSS test for stationary
#define function for kpss test
from statsmodels.tsa.stattools import kpss
#define KPSS
def kpss_test(series, **kw):
    statistic, p_value, n_lags, critical_values = kpss(series, **kw)
    # Format Output
    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critial Values:')
    for key, value in critical_values.items():
        print(f'   {key} : {value}')
    print(f'Result: The series is {"not " if p_value < 0.05 else ""}stationary')

#p vlue is less than 0.05 and test stats is greater than the crit values so we reje
#But
kpss_test(dataset['AQI'], regression='ct')
```

Following Models have been applied:

1) Exponential Smoothing

```
#exponential smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
start_time = time.time()
model_exp = ExponentialSmoothing(train_data['AQI'], trend='mul',seasonal='mul',seasonal_periods=4)
fitted_model_exp = model_exp.fit()
test_predictions_exp = fitted_model_exp.forecast(30)
print("--- %s seconds ---" % (time.time() - start_time))

train_data['AQI'].plot(legend=True,label='Train',figsize=(12,8))
test_data['AQI'].plot(legend=True,label='Test')
test_predictions_exp.plot(legend=True,label='Prediction')

#calling all evalaution metrics and defining mape
from sklearn.metrics import mean_squared_error,mean_absolute_error
from statsmodels.tools.eval_measures import rmse,mse
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


mse_exp = mean_squared_error(test_data,test_predictions_exp)
mae_exp = mean_absolute_error(test_data,test_predictions_exp)
rmse_exp = rmse(test_data['AQI'],test_predictions_exp)
mape_exp = mean_absolute_percentage_error(test_data, test_predictions_exp)
#comparing the standard deviation of the prediction with the real data
dataset.describe()
```

2) Auto – Regression (Order 1,2 and 19)

```
#finding the best order value of P
start_time = time.time()
ARfit = model_ar.fit(ic='t-stat')
ARfit.params

#order 19 is the best
prediction_ar19 = ARfit.predict(start,end)
prediction_ar19 = prediction_ar19.rename('AR(19) Predictions')
print("--- %s seconds ---" % (time.time() - start_time))

labels = ['AR1','AR2','AR19']

preds = [prediction_ar1,prediction_ar2,prediction_ar19]

for i in range(3):
    error = mean_absolute_error(test_data['AQI'],preds[i])
    print(f'{labels[i]} MAE was :{error}')

test_data.plot(figsize =(12,8),legend=True)
prediction_ar1.plot(legend=True)
prediction_ar2.plot(legend=True)
prediction_ar19.plot(legend=True)

dataset.AQI.mean()

mse_ar19 = mean_squared_error(test_data,prediction_ar19)
mae_ar19 = mean_absolute_error(test_data,prediction_ar19)
rmse_ar19 = rmse(test_data['AQI'],prediction_ar19)
mape_ar19 = mean_absolute_percentage_error(test_data, prediction_ar19)
```

3) Auto-Regressive Moving Order

```
#ARMA
from statsmodels.tsa.arima_model import ARMA,ARIMA,ARMAResults,ARIMAResults
train_data = dataset.iloc[:943]
test_data = dataset.iloc[943:]

#checking for all values. got 212 to be the best
auto_arima(dataset['AQI'],seasonal=False).summary()

start = len(train_data)
end = len(train_data) + len(test_data) - 1
#applying ARMA first
start_time = time.time()
model_arma = ARMA(train_data['AQI'],order=(2,2))
result_arma = model_arma.fit()
result_arma.summary()

#making the length frame

predictions_arma = result_arma.predict(start,end).rename('ARMA (2,2) Predictions')
print("--- %s seconds ---" % (time.time() - start_time))

test_data['AQI'].plot(figsize=(12,8),legend=True)
predictions_arma.plot(legend=True)
```

4) ARIMA

```
#ARIMA(2,1,2) has the lowest AIC so its the best
start_time = time.time()
model_arima = ARIMA(train_data['AQI'],order=(2,1,2))
#fitting the model
result_arima = model_arima.fit()

#maing the predicitons
prediction_arima = result_arima.predict(start=start,end=end,typ='levels').rename('ARIMA(2,1
print("--- %s seconds ---" % (time.time() - start_time))

result_arima.summary()

#plotting
test_data['AQI'].plot(legend=True,figsize=(12,8))
prediction_arima.plot(legend=True)

#checking the errors
mse_arima = mean_squared_error(test_data,prediction_arima)
mae_arima = mean_absolute_error(test_data,prediction_arima)
rmse_arima = rmse(test_data['AQI'],prediction_arima)
mape_arima = mean_absolute_percentage_error(test_data, prediction_arima)

#forecasting with arima
model_arima = ARIMA(dataset['AQI'],order=(2,1,2))
results_arima = model_arima.fit()
forecast_arima = results_arima.predict(start=len(dataset),end=len(dataset)+7,typ='levels').
```

5) SARIMA

```
'''SARIMA'''
#there maybe a bit of seasonality so SARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
#checking the best value for sarimax
auto_arima(train_data,seasonal=True,m=7).summary()

#giving the length of the start and end
start = len(train_data)
end = len(train_data) + len(test_data) - 1
#SARIMAX (1,1,1) is the best as per auto_arima
start_time = time.time()

model_sarima = SARIMAX(train_data['AQI'],order = (1,1,1))
result_sarima = model_sarima.fit()
result_sarima.summary()

prediction_sarima = result_sarima.predict(start,end,typ='levels').rename('SARIMA Prediction')
print("--- %s seconds ---" % (time.time() - start_time))
```

6) Prophet (Facebook Research, 2019)

```
#prophet
from fbprophet import Prophet #need to import matplotlib.pyplc
import matplotlib.pyplot as plt

#prphet needs spedific names for date and data column
dataset_prophet = dataset.reset_index()
dataset_prophet.columns = ['ds', 'y']
dataset_prophet.head()
#changing the type of date column to datetime
dataset_prophet['ds'] = pd.to_datetime(dataset_prophet['ds'])

#prediction
dataset_prophet.info()
#splitting
train_data_prophet = dataset_prophet.iloc[:966]
test_data_prophet = dataset_prophet.iloc[966:]

#fitting the model
start_time = time.time()

n = Prophet()
n.fit(train_data_prophet)
#making a future empty dataframe to store the predicitons in
future_prophet = n.make_future_dataframe(periods=7,freq='D')
#predicitng the dates in the future dataframe we made
forecast_prophet = n.predict(future_prophet)
print("--- %s seconds ---" % (time.time() - start_time))
```

7) TBATS

```
train_data = dataset.iloc[:943]
test_data = dataset.iloc[943:]

# Fit the model
start_time = time.time()

estimator_tbat = TBATS(seasonal_periods=(1, 365.25))
model_tbat = estimator_tbat.fit(train_data)
# Forecast 7 days ahead
tbat_forecast = model_tbat.forecast(steps=30)
print("--- %s seconds ---" % (time.time() - start_time))

tbat_forecast = pd.DataFrame(tbat_forecast)
tbat_forecast.index = test_data.index

test_data['AQI'].plot(legend=True,label='Test', figsize=(12,8))
tbat_forecast[0].plot(legend=True,label='TBAT')
```

8) LSTM

```
len(dataset)
train_data = dataset.iloc[:966]
test_data = dataset.iloc[966:]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(train_data)
scaled_train = scaler.transform(train_data)
scaled_test = scaler.transform(test_data)

#making the time sereis for keras
from keras.preprocessing.sequence import TimeseriesGenerator
n_input = 30
n_features = 1

start_time = time.time()

train_generator = TimeseriesGenerator(scaled_train,scaled_train,length=n_input,batch_size=30)

X,y = train_generator[0]

model_lstm = Sequential()
model_lstm.add(LSTM(150,activation='relu',input_shape=(n_input,n_features)))
model_lstm.add(Dropout(0.15))

model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam',loss='mse')
```

References

Brownlee, J. (2019). *Time Series Data Visualization with Python*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/time-series-data-visualization-with-python/

SINGH, A. (2019). *A Gentle Introduction to Handling a Non-Stationary Time Series in Python*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/

Peixeiro, M. (2019). *End to End Time Series Analysis and Modelling*. [online] Medium. Available at: https://towardsdatascience.com/end-to-end-time-series-analysis-and-modelling-8c34f09a3014.

Brownlee, J. (2019). *How to Check if Time Series Data is Stationary with Python*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/time-series-data-stationary-python/.

App.cpcbccr.com. (2019). *CCR*. [online] Available at: https://app.cpcbccr.com/ccr/#/caaqm-dashboard/caaqm-landing/caaqm-data-availability.

Facebook Research. (2019). *Prophet: forecasting at scale - Facebook Research*. [online] Available at: https://research.fb.com/prophet-forecasting-at-scale/ .

4 Appendix

Code is attached in a separate archived file