National College of Ireland

Prediction of Suspended Particulate Matter Using Machine Learning-Configuration Manual

MSc Research Project
Data Analytics

Vinayak Kolekar
Student ID: x18185797

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vinayak Kolekar
Student ID:	x18185797
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	28/09/2020
Project Title:	Prediction of Suspended Particulate Matter Using Machine Learning-Configuration Manual
Word Count:	917
Page Count:	16

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:	27th September 2020

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):	

Prediction of Suspended Particulate Matter Using Machine Learning-Configuration Manual

Vinayak Kolekar
x18185797

1 Introduction

The configuration manual consists of a detailed description of the environmental setup while implementation of the research project: Prediction of Suspended Particulate Matter Using Machine Learning-Configuration Manual.

The configuration manual section 2 is about hardware and software configuration. Section 3 is about data source, whereas section 4 is about implementation.

2 System Configuration

2.1 Hardware Specification

- Operating system: Windows 10 Education (2019 Microsoft Corporation)
- Processor: Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
- RAM: 16 GB
- System type: 64- bit Operating System, x64-based processor
- Hard drive: 1TB

2.2 Software Specification

The research implemented based on two software environment R studio for statistical methods based on R language, whereas the Jupyter notebook of Anaconda used for machine learning models based on Python language. The libraries used during research implementation are as follows.

2.2.1 R studio

The R studio version 1.3.959 and R version 4.0.2 is used for implementation of R code. The installed packages in R studio are as follows.

- dplyr[1]

[1] `https://cran.r-project.org/web/packages/dplyr/dplyr.pdf`

- ggplot2[2]
- imputeTS[3]
- tseries[4]
- urca[5]
- forecast[6]
- vars[7]
- tsDyn [8]
- TSA [9]
- lubridate[10]
- MLmetrics[11]

2.2.2 Anaconda

The Anaconda version used is 1.9.12 and Jupyter notebook 6.0.3 web application platform is used to demonstrate the python code. The Python environment used in Jupyter notebook is 3.7.6. The installed python libraries are as follows.

- Pandas 1.0.1 [12]
- Numpy 1.19.0 [13]
- Scikit-learn 0.22.1[14]
- Keras 2.4. [15]
- Matplotlib 3.1.3[16]
- Tensorfow 2.2.0[17]

[2] https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf
[3] https://cran.r-project.org/web/packages/imputeTS/imputeTS.pdf
[4] https://www.rdocumentation.org/packages/tseries/versions/0.10-47
[5] https://cran.r-project.org/web/packages/urca/urca.pdf
[6] https://cran.r-project.org/web/packages/forecast/forecast.pdf
[7] https://www.rdocumentation.org/packages/vars/versions/1.5-3
[8] https://www.rdocumentation.org/packages/tsDyn/versions/0.9-44
[9] https://www.rdocumentation.org/packages/TSA/versions/1.2.1
[10] https://www.rdocumentation.org/packages/lubridate/versions/1.7.9
[11] https://cran.r-project.org/web/packages/MLmetrics/MLmetrics.pdf
[12] https://pandas.pydata.org
[13] https://www.numpy.org
[14] http://scikit-learn.org
[15] https://keras.io/
[16] https://matplotlib.org
[17] https://www.tensorflow.org/

3 Data Sources

The water quality data from the Environmental protection agency (EPA)[18] and precipitation data from Met Éireann[19], combined and formed the data set for forecasting of pH level. The data repository consists of multiple data sets for counties in Ireland, the Wicklow county data from 2015 to 2018 is considered. Similarly, the rainfall of the sample site from Wicklow county is considered from Met Éireann. On the other side, the water quality data from New York's open data[20] and the daily precipitation data from the National Climate Data Center[21] combined and formed the data set for turbidity forecasting, the similar sample site is considered for water quality and rainfall amount data. Table 1 shows the data repository.

Category	Data source	Description	Features	Target variables
Weather data	Met Éireann	Meteorological data	Daily rainfall amount in mm	Rainfall amount (mm)
Water quality monitoring data	Environmental protection agency (EPA)	Water quality monitoring data	Sample Date, Water supplier, Scheme code ,Water quality parameters	pH level
Weather data	National Climate Data Center	Meteorological data including Rainfall, temperature	Daily rainfall amount in mm	Rainfall amount (mm)
Water quality monitoring data	NYC open data	Water quality monitoring data	Sample Date, Water quality parameters, Sample site, location	Turbidity(NTU)

Table 1: Data repository

[18] http://erc.epa.ie/safer/resourcelisting.jsp?oID=10206&username=EPA%20Drinking%20Water

[19] https://www.met.ie/climate/available-data/historical-data

[20] https://data.cityofnewyork.us/Environment/Drinking-Water-Quality-Distribution-Monitoring-Dat/bkwf-xfky

[21] https://www.ncdc.noaa.gov/cdo-web/

4 Implementation

The collected data from water quality monitoring databases and meteorological stations merged for forecasting Turbidity and pH level. Two separate code execution maintained for forecasting of Turbidity and pH level. The autoregressive models (VAR, VECM, ARIMAX) are implemented in R, whereas for LSTM python programming is used.

4.1 Data Pre-processing

Mean values of Turbidity and pH level per top 10 sites are plotted, and filtration is done on one sample site to make research analysis more meaningful as described in research report.

```
#Mean Turbidity According to sample site
Mean_plot <- finaldata_NYC %>%
  group_by(Sample_Site) %>%
  summarise(Mean_Turbidity = mean(Turbidity))

# Plot the graph
ggplot(Mean_plot  %>%
         top_n(10, Mean_Turbidity), aes(x=reorder(Sample_Site,-Mean_Turbidity), y = Mean_Turbidity))+
          geom_bar(stat="identity",color='skyblue',fill='steelblue')

#Filter the data according to sample site
NYC_Water_10250 = finaldata_NYC[(finaldata_NYC$Sample_Site=="10250"),]
View(NYC_Water_10250)
```

Figure 1: Turbidity data set Filter

Similarly, on pH level Data

```
#Mean pH According to sample site
Mean_plot_pH <- finaldata_wick %>%
  group_by(Scheme.Code) %>%
  summarise(Mean_pH = mean(pH))

# Plot the graph
ggplot(Mean_plot_pH  %>%
         top_n(10, Mean_pH), aes(x=reorder(Scheme.Code,-Mean_pH), y = Mean_pH))+
         geom_bar(stat="identity",color='skyblue',fill='steelblue')

#filter with respect to scheme code
wick_water_PUB1001 = finaldata_wick[(finaldata_wick$Scheme.Code=="3400PUB1001"),]
View(wick_water_PUB1001)
```

Figure 2: pH Level data set Filter

After filtration on sample sites, the plots of Turbidity and pH level as shown below.

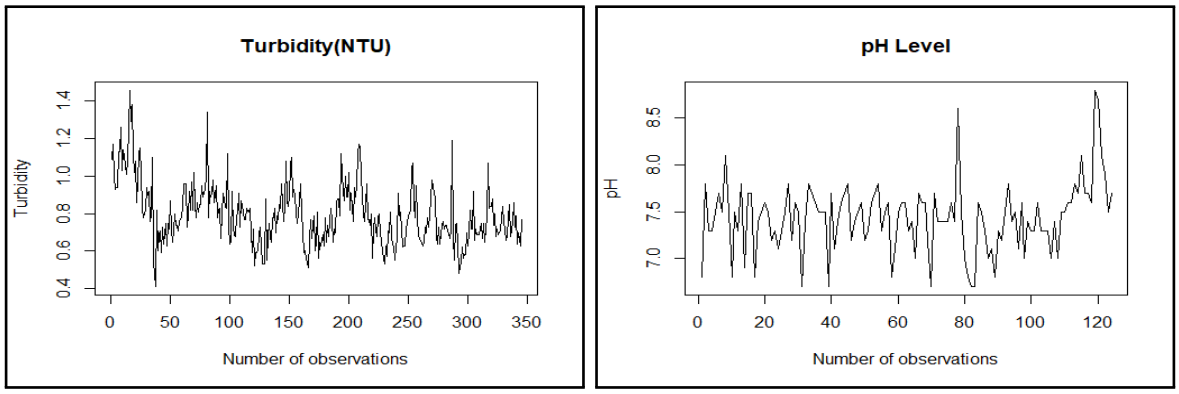

Figure 3: plots of Turbidity and pH level

The daily data obtained and filled with the linear interpolation with the help of "imputeTS" package as described in research report.

```
#Fill NA in between data to make Daily data set
Daily_data_NYC <- data.frame(Date = seq(Data_NCY_10250$Date[1],
                                        Data_NCY_10250$Date[nrow(Data_NCY_10250)], by = 1)) %>%
                  full_join(Data_NCY_10250 , by = "Date")
#check Missing values
View(Daily_data_NYC)

#TS Indivisual
Turbidity_daily<-ts(Daily_data_NYC$Turbidity)
Rain_daily<-ts(Daily_data_NYC$Rain)

#Fill missing values with interpolation
Turbidity_interpol<-na.interpolation(Turbidity_daily, option = "linear")
Rain_interpoled<-na.interpolation(Rain_daily, option = "linear")
```

Figure 4: Interpolation in Turbidity data

```
#Fill NA in between data to make Daily data set
Daily_data_wick <- data.frame(Sample_Date = seq(Data_Wick$Sample_Date[1],
                                                Data_Wick$Sample_Date[nrow(Data_Wick)], by = 1)) %>%
                                               full_join(Data_Wick, by = "Sample_Date")
#check Missing values
View(Daily_data_wick)

#TS Indivisual
pH_daily<-ts(Daily_data_wick$pH)
Rain_daily_wic<-ts(Daily_data_wick$Rain)

#add interpolated values in dataframe
Daily_data_wick$Interpolate_pH<-na.interpolation(pH_daily, option = "linear")
Daily_data_wick$Interpolate_rain<-na.interpolation(Rain_daily_wic, option = "linear")
```

Figure 5: Interpolation in pH level data

The seasonal plots are plotted to check the seasonal effect of Turbidity and pH level. The Line and polar seasonal plots are formed on two data sets. Code for visualization of seasonal plots is as follows.

```
#Seasonal plots
Daily_data_NYC$year <- year(Daily_data_NYC$Date)
Daily_data_NYC$month <- month(Daily_data_NYC$Date)

# calculate the sum  for each month
sum_rain_turbidity <- Daily_data_NYC %>%
  group_by(month, year) %>%
  summarise(summarise_tur = mean(Interpolated_turbidity),
            summarise_rain = mean(Interpolated_rain))

time_series_tur_plot<- ts(sum_rain_turbidity$summarise_tur,frequency =12,start = 2015)
time_series_rain_plot<- ts(sum_rain_turbidity$summarise_rain,frequency =12,start = 2015)

ggseasonplot(time_series_tur_plot, polar=TRUE) +
  ylab("Mean Turbidity") +
  ggtitle("Polar seasonal plot:Turbidity")

ggseasonplot(time_series_tur_plot, year.labels=TRUE, year.labels.left=TRUE) +
  ylab("Mean Turbidity") +
  ggtitle(" Turbidity Seasonal plot")
```

Figure 6: Seasonal plots for Turbidity data

```
#Seasonal plots
Daily_data_wick$year <- year(Daily_data_wick$Sample_Date)
Daily_data_wick$month <- month(Daily_data_wick$Sample_Date)

# calculate the sum  for each month
sum_rain_pH <- Daily_data_wick %>%
  group_by(month, year) %>%
  summarise(summarise_ph = mean(Interpolate_pH),
            summarise_rain_wick = mean(Interpolate_rain))

time_series_pH_plot<- ts(sum_rain_pH$summarise_ph,frequency =12,start = 2015)
time_series_rain_wick_plot<- ts(sum_rain_pH$summarise_rain_wick,frequency =12,start = 2015)

ggseasonplot(time_series_pH_plot, polar=TRUE) +
  ylab("Mean pH") +
  ggtitle("Polar seasonal plot:pH")

ggseasonplot(time_series_pH_plot, year.labels=TRUE, year.labels.left=TRUE) +
  ylab("Mean pH") +
  ggtitle(" pH Seasonal plot")
```

Figure 7: Seasonal plots for pH level data

The following plots are polar seasonal plots of Turbidity and pH level.

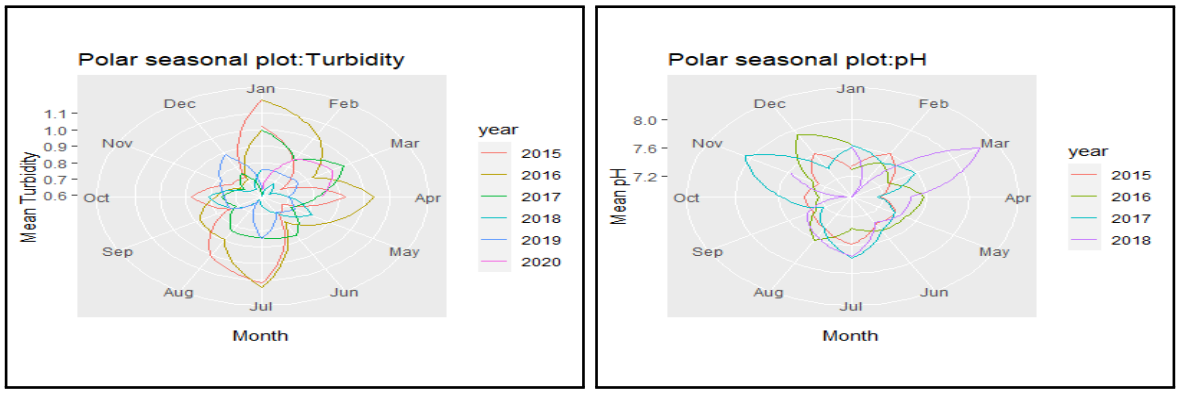

Figure 8: Polar seasonal plots of Turbidity and pH level

4.2 Data transformation

The data transformation transforms data from one form into another to make it suitable for model construction. The data conversion into a time-series format is done as described in research report. The seasonality check is done with the help of "stl" function and adjustments are done with "seasadj" function from "forecast" package.

```
#Remove seasonality from data and make it deseasonal
Seasonality_tur = stl(time_series_turbidity, s.window="periodic")
deseasonal_turt <- seasadj(Seasonality_tur)
plot(Seasonality_tur)
plot(deseasonal_turt)

#ACF and PACF plots
Acf(deseasonal_turt, main='')
Pacf(deseasonal_turt, main='')

#Remove seasonality from data and make it deseasonal
Seasonality_rain = stl(time_series_Rain, s.window="periodic")
deseasonal_rain <- seasadj(Seasonality_rain)
plot(Seasonality_rain)
plot(deseasonal_rain)

#ACF and PACF plots
Acf(deseasonal_rain, main='')
Pacf(deseasonal_rain, main='')
```

Figure 9: Seasonality adjustment in Turbidity data

```r
#Remove seasonality from data and make it deseasonal
Seasonality_ph = stl(time_series_ph, s.window="periodic")
deseasonal_ph <- seasadj(Seasonality_ph)
plot(Seasonality_ph)
plot(deseasonal_ph)

#ACF and PACF plots
Acf(deseasonal_ph, main='',lag.max = 150)
Pacf(deseasonal_ph, main='',lag.max = 20)

#Remove seasonality from data and make it deseasonal
Seasonality_Rain_wick = stl(time_series_Rain_wick, s.window="periodic")
deseasonal_Rain_wick <- seasadj(Seasonality_Rain_wick)
plot(Seasonality_Rain_wick)
plot(deseasonal_Rain_wick)

#ACF and PACF plots
Acf(deseasonal_Rain_wick, main='')
Pacf(deseasonal_Rain_wick, main='')
```

Figure 10: Seasonality adjustment in pH level data

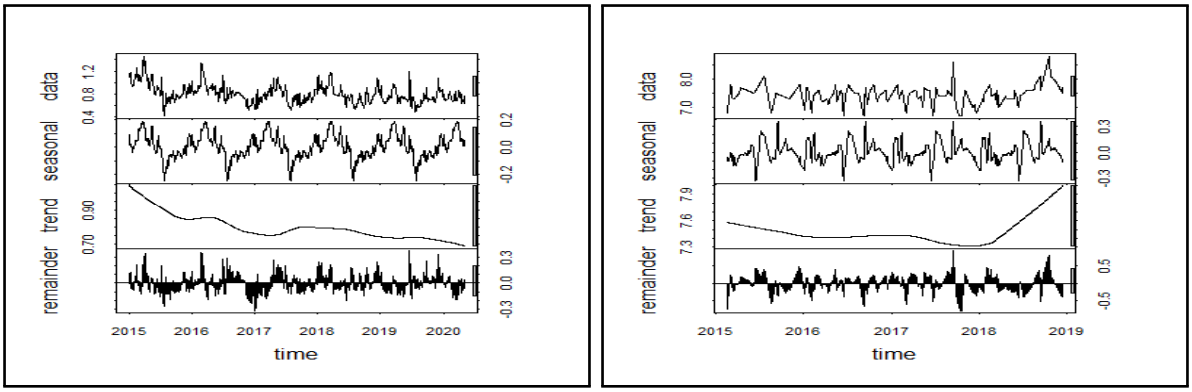

Figure 11: Seasonality in Turbidity and pH level

Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots for Turbidity are as follows.

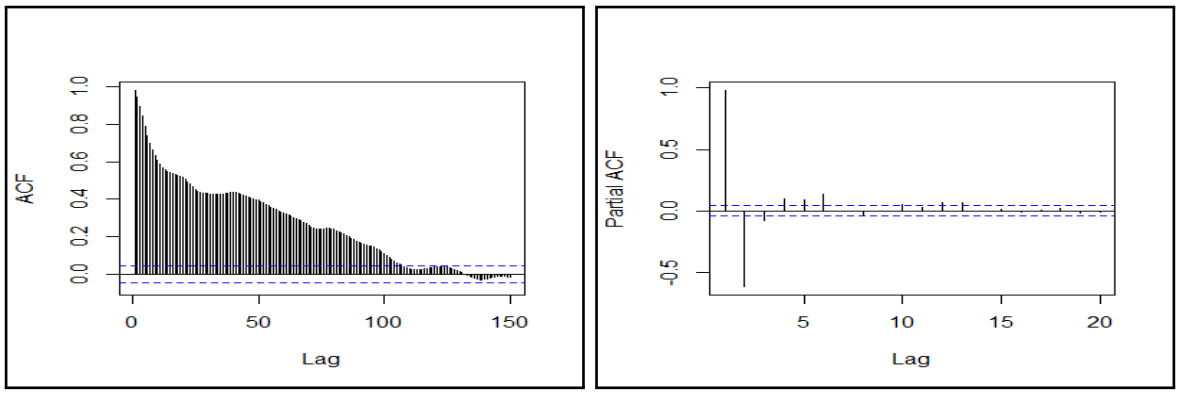

Figure 12: ACF and PACF plots of Turbidty level

Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots for pH level are as follows.

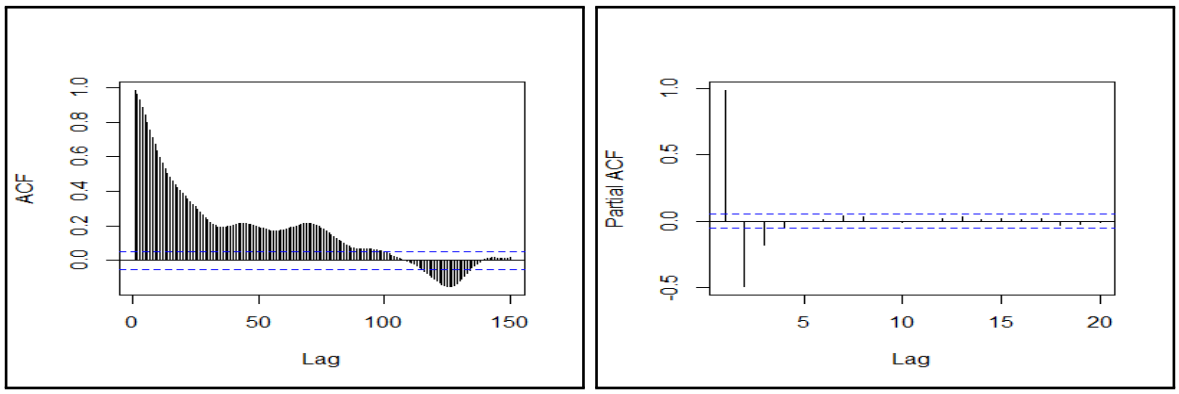

Figure 13: ACF and PACF plots of pH level

4.3 Model building and evaluation

The time series objects stationarity check has been done with the help of Augmented Dickey-Fuller (ADF) test.

```
> #ADF test for stationarity
> adf.test(deseasonal_turt, alternative = c("stationary", "explosive"),
+          k = trunc((length((deseasonal_turt)-1)^(1/3))))

        Augmented Dickey-Fuller Test

data:  deseasonal_turt
Dickey-Fuller = -5.6533, Lag order = 12, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(deseasonal_turt, alternative = c("stationary", "explosive"),
  p-value smaller than printed p-value
> #ADF test for stationarity
> adf.test(deseasonal_rain, alternative = c("stationary", "explosive"),
+          k = trunc((length((deseasonal_rain)-1)^(1/3))))

        Augmented Dickey-Fuller Test

data:  deseasonal_rain
Dickey-Fuller = -8.2268, Lag order = 12, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(deseasonal_rain, alternative = c("stationary", "explosive"),
  p-value smaller than printed p-value
```

Figure 14: ADF Test on Turbidity and Rainfall data

```
> #ADF test for stationarity
> adf.test(deseasonal_ph, alternative = c("stationary", "explosive"),
+          k = trunc((length((deseasonal_ph)-1)^(1/3))))

        Augmented Dickey-Fuller Test

data:  deseasonal_ph
Dickey-Fuller = -5.0408, Lag order = 11, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(deseasonal_ph, alternative = c("stationary", "explosive"),  :
  p-value smaller than printed p-value
> #ADF test for stationarity
> adf.test(deseasonal_Rain_wick, alternative = c("stationary", "explosive"),
+          k = trunc((length((deseasonal_Rain_wick)-1)^(1/3))))

        Augmented Dickey-Fuller Test

data:  deseasonal_Rain_wick
Dickey-Fuller = -5.9814, Lag order = 11, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(deseasonal_Rain_wick, alternative = c("stationary",  :
  p-value smaller than printed p-value
```

Figure 15: ADF Test on pH level and Rainfall data

4.3.1 VAR

The optimal lag selected with the help of VARselect and the model VAR is implemented as shown in the figure 18 and 19. The forecast plots of actual versus predicted values are described in the research report.

```
> varselect<-VARselect(Combined_time_series)
> varselect$selection
AIC(n)  HQ(n)  SC(n) FPE(n)
    10      6      6     10
> varselect$criteria
                1             2             3             4             5             6             7
AIC(n) -1.319200e+01 -1.447854e+01 -1.447802e+01 -1.449670e+01 -1.452107e+01 -1.454803e+01 -1.454587e+01
HQ(n)  -1.318566e+01 -1.446798e+01 -1.446323e+01 -1.447770e+01 -1.449784e+01 -1.452057e+01 -1.451419e+01
SC(n)  -1.317477e+01 -1.444983e+01 -1.443782e+01 -1.444502e+01 -1.445790e+01 -1.447337e+01 -1.445973e+01
FPE(n)  1.865470e-06  5.152862e-07  5.155574e-07  5.060139e-07  4.938330e-07  4.806989e-07  4.817375e-07
                8             9            10
AIC(n) -1.455096e+01 -1.454908e+01 -1.455126e+01
HQ(n)  -1.451506e+01 -1.450895e+01 -1.450691e+01
SC(n)  -1.445334e+01 -1.443997e+01 -1.443066e+01
FPE(n)  4.792884e-07  4.801930e-07  4.791468e-07
>
```

Figure 16: Varselect on Turbidity data

```
> varselectwick<-VARselect(Combined_time_series_wick)
> varselectwick$selection
AIC(n)  HQ(n)  SC(n) FPE(n)
     6      3      2      6
> varselectwick$criteria
                1             2             3             4             5             6             7
AIC(n) -8.1552076901 -9.535118e+00 -9.541630e+00 -9.5425726032 -9.544537e+00 -9.557494e+00 -9.556239e+00
HQ(n)  -8.1467383347 -9.521003e+00 -9.521868e+00 -9.5171645372 -9.513483e+00 -9.520794e+00 -9.513892e+00
SC(n)  -8.1325621309 -9.497376e+00 -9.488791e+00 -9.4746359258 -9.461504e+00 -9.459364e+00 -9.443011e+00
FPE(n)  0.0002872356  7.226878e-05  7.179972e-05  0.0000717321  7.159133e-05  7.066972e-05  7.075855e-05
                8             9            10
AIC(n) -9.552989e+00 -9.549874e+00 -9.550031e+00
HQ(n)  -9.504996e+00 -9.496235e+00 -9.490745e+00
SC(n)  -9.424664e+00 -9.406452e+00 -9.391512e+00
FPE(n)  7.098891e-05  7.121046e-05  7.119942e-05
```

Figure 17: Varselect on pH data

```
# Apply model Vector autoregression
VAR_est <- VAR(y = training, p = 10)
summary(VAR_est)

# Forecast future values
forecasts <- forecast(VAR_est,h=390)
plot(forecasts)

# Make data frame of predicted values
pred_VAR=data.frame(forecasts$forecast$deseasonal_turt$mean)

#Evaluation measures MSE, RSME, MAE and MAPE
MSE(pred_VAR$forecasts.forecast.deseasonal_turt.mean, Dataframe_comb_series$deseasonal_turt)
RMSE(pred_VAR$forecasts.forecast.deseasonal_turt.mean, Dataframe_comb_series$deseasonal_turt)
MAE(pred_VAR$forecasts.forecast.deseasonal_turt.mean, Dataframe_comb_series$deseasonal_turt)
MAPE(pred_VAR$forecasts.forecast.deseasonal_turt.mean, Dataframe_comb_series$deseasonal_turt)
```

Figure 18: VAR model on turbidity data set

```
# Apply model Vector autoregression
VAR_est_wick <- VAR(y = training_wick, p = 6)
summary(VAR_est_wick)

# Forecast future values
forecasts_wick <- forecast(VAR_est_wick,h=280)
plot(forecasts_wick)

# Make data frame of predicted values
pred_VAR_pH=data.frame(forecasts_wick$forecast$deseasonal_ph$mean)

#Evaluation measures MSE, RSME, MAE and MAPE
MSE(pred_VAR_pH$forecasts_wick.forecast.deseasonal_ph.mean, test_wick$deseasonal_ph)
RMSE(pred_VAR_pH$forecasts_wick.forecast.deseasonal_ph.mean, test_wick$deseasonal_ph)
MAE(pred_VAR_pH$forecasts_wick.forecast.deseasonal_ph.mean, test_wick$deseasonal_ph)
MAPE(pred_VAR_pH$forecasts_wick.forecast.deseasonal_ph.mean, test_wick$deseasonal_ph)
```

Figure 19: VAR model on pH data set

4.3.2 VECM

The Johansen cointegration test is applied to multivariate series and the VECM model is implemented as shown in figure 22 for turbidity and figure 23 for pH level. The forecast plots of actual versus predicted values are described in research report.

```
> # Johansens cointegration test
> jotest_NYC=ca.jo(data.frame(Combined_time_series), type="trace",
+                  K=2, ecdet="none", spec="longrun")
> summary(jotest_NYC)

######################
# Johansen-Procedure #
######################

Test type: trace statistic , with linear trend

Eigenvalues (lambda):
[1] 0.11037068 0.03733509

Values of teststatistic and critical values of test:

           test 10pct  5pct  1pct
r <= 1 |  74.12  6.50  8.18 11.65
r = 0  | 301.94 15.66 17.95 23.52

Eigenvectors, normalised to first column:
(These are the cointegration relations)

                    deseasonal_turt.l2 deseasonal_rain.l2
deseasonal_turt.l2            1.00000        1.000000000
deseasonal_rain.l2           20.24988       -0.003766074

Weights W:
(This is the loading matrix)

                   deseasonal_turt.l2 deseasonal_rain.l2
deseasonal_turt.d      -7.557333e-05       -0.025978601
deseasonal_rain.d      -3.539660e-03       -0.004012495
```

Figure 20: Johansen Cointegration test for turbidity and rainfall

```
> jotest_NYCwick=ca.jo(data.frame(Combined_time_series_wick), type="trace",
+                      K=2, ecdet="none", spec="longrun")
> summary(jotest_NYCwick)

######################
# Johansen-Procedure #
######################

Test type: trace statistic , with linear trend

Eigenvalues (lambda):
[1] 0.05694795 0.02326445

Values of teststatistic and critical values of test:

           test 10pct  5pct  1pct
r <= 1 |  32.84  6.50  8.18 11.65
r = 0  | 114.63 15.66 17.95 23.52

Eigenvectors, normalised to first column:
(These are the cointegration relations)

                         deseasonal_ph.l2 deseasonal_Rain_wick.l2
deseasonal_ph.l2                 1.00000              1.00000000
deseasonal_Rain_wick.l2         -1.33088              0.01184508

Weights W:
(This is the loading matrix)

                        deseasonal_ph.l2 deseasonal_Rain_wick.l2
deseasonal_ph.d           0.0001729966             -0.01730233
deseasonal_Rain_wick.d    0.0200009313              0.02745730
```

Figure 21: Johansen Cointegration test for pH level and rainfall

```
#VECM model for turbidity
VECM_NYC=VECM(training,lag = 10,r=1,estim = "ML")
summary(VECM_NYC)

#Predict future test values
pred_vecm<-predict(VECM_NYC,n.ahead =  390)

# Make data frame of predicted values
pred_VECM=data.frame(pred_vecm)

#Evaluation measures MSE, RSME, MAE and MAPE
MSE(pred_VECM$deseasonal_turt, Dataframe_comb_series$deseasonal_turt)
RMSE(pred_VECM$deseasonal_turt, Dataframe_comb_series$deseasonal_turt)
MAE(pred_VECM$deseasonal_turt, Dataframe_comb_series$deseasonal_turt)
MAPE(pred_VECM$deseasonal_turt, Dataframe_comb_series$deseasonal_turt)
```

Figure 22: VECM model on turbidity data set

```
#VECM model for pH level
VECM_wick=VECM(training_wick,lag = 6,r=1,estim = "ML")
summary(VECM_wick)

#Predict future test values
pred_vecm_wick<-predict(VECM_wick,n.ahead =  280)
plot(pred_vecm_wick)

# Make data frame of predicted values
pred_VECM_wick_ph=data.frame(pred_vecm_wick)

#Evaluation measures MSE, RSME, MAE and MAPE
MSE(pred_VECM_wick_ph$deseasonal_ph, test_wick$deseasonal_ph)
RMSE(pred_VECM_wick_ph$deseasonal_ph, test_wick$deseasonal_ph)
MAE(pred_VECM_wick_ph$deseasonal_ph, test_wick$deseasonal_ph)
MAPE(pred_VECM_wick_ph$deseasonal_ph, test_wick$deseasonal_ph)
```

Figure 23: VECM model on pH data set

4.3.3 ARIMAX

The ARIMAX model execution is as shown in figure 24 for turbidity and figure 25 for pH level. The forecast plots of actual versus predicted values are described in research report.

```
#Model train
Arimax_nyc <- arimax(training_tur,xreg =training_rain ,order = c(10,0,10))
preds_arimax <- predict(Arimax_nyc, newxreg = test_rain,n.ahead = 390)

#Evaluation measures MSE, RSME, MAE and MAPE
MSE(preds_arimax$pred, test_tur)
RMSE(preds_arimax$pred, test_tur)
MAE(preds_arimax$pred, test_tur)
MAPE(preds_arimax$pred, test_tur)
```

Figure 24: Arimax model on turbidity data set

```
#Model train
Arimax_wick <- arima(training_ph,xreg =training_rain_wcik ,order = c(6,0,4))
preds.temporal_wick <- predict(Arimax_wick, newxreg = test_rain_wick,n.ahead = 280)

#Evaluation measures MSE, RSME, MAE and MAPE
MSE(preds.temporal_wick$pred, test_ph)
RMSE(preds.temporal_wick$pred, test_ph)
MAE(preds.temporal_wick$pred, test_ph)
MAPE(preds.temporal_wick$pred, test_ph)
```

Figure 25: Arimax model on pH data set

4.3.4 LSTM

The pre-processed data extracted from R studio by names "NewYork_LSTM1" for Turbidity, "wicklow_LSTM1" for pH level separately, and loaded in Jupyter notebook to implement LSTM model. The time-series data converted into supervised learning data with the help of method, as shown below for both Turbidity and pH data. The data is scaled with the help of the "MinMaxScaler" function and divided into an 80:20 ratio as shown below.

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
values = dataset.values
# integer encode direction
# ensure all data is float
values = values.astype('float32')
# normalize features from 0 to 1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns that don't want to predict
reframed.drop(reframed.columns[[3]], axis=1, inplace=True)
print(reframed)
```

Figure 26: Supervised learning data

```
# split into train and test sets
values = reframed.values
n_train_hours = 1560
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(1560, 1, 2) (1560,) (389, 1, 2) (389,)
```

```
# split into train and test sets
values = reframed.values
n_train_hours = 1117
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(1117, 1, 2) (1117,) (279, 1, 2) (279,)
```

Figure 27: Train test division of Turbidity and pH data

The figure 28 is LSTM network used for both Turbidity and pH data set.

```
In [31]: ...
         # design network
         model = Sequential()
         model.add(LSTM(3, input_shape=(train_X.shape[1], train_X.shape[2])))
         model.add(Dense(1))
         model.compile(loss='mae', optimizer='adam')
         model.summary()

         Model: "sequential_3"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         lstm_2 (LSTM)                (None, 3)                 72
         _________________________________________________________________
         dense_2 (Dense)              (None, 1)                 4
         =================================================================
         Total params: 76
         Trainable params: 76
         Non-trainable params: 0
         _________________________________________________________________
```

Figure 28: LSTM network for Turbidity and pH data set

After model training with the help of linear activation function, batch size of 32 and 200 epochs for both data sets, the loss of train and validation shows in the figure 29. The forecast plots of actual versus predicted values are described in research report.

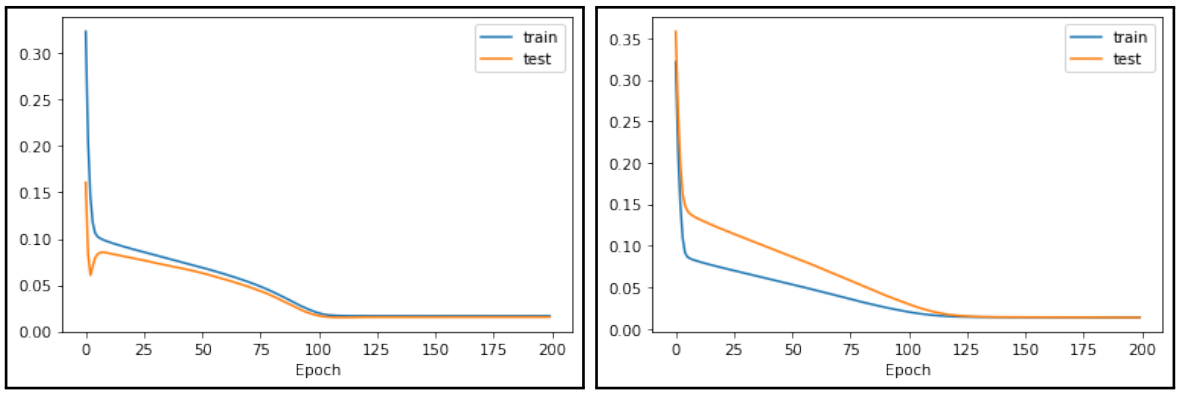

Figure 29: Loss plots train and test of Turbidity and pH level