BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

A Report On

TIME SERIES DATA ANALYSIS WITH PYTHON AND R

Prepared By

Swadesh Vaibhav	**2017A7PS0030P**
Anirudh Goyal	**2017A7PS0031P**
Gandhi Atith Nikeshkumar	**2017A7PS0062P**
Rohit Milind Rajhans	**2017A7PS0105P**
Ritik Rohit Bavdekar	**2017B4A70349P**
Gupta Pranay Pradeep	**2017B5A70831P**

At

National Centre for Polar and Ocean Research, Goa

A Practice School-1 station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(July, 2019)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

A Report On

TIME SERIES DATA ANALYSIS WITH PYTHON AND R

By

ID No.	Name	Discipline
2017A7PS0030P	Swadesh Vaibhav	Computer Science
2017A7PS0031P	Anirudh Goyal	Computer Science
2017A7PS0062P	Gandhi Atith Nileshkumar	Computer Science
2017A7PS0105P	Rohit Milind Rajhans	Computer Science
2017B4A70349P	Ritik Rohit Bavdekar	Maths + Computer Science
2017B5A70831P	Gupta Pranay Pradeep	Physics + Computer Science

Prepared in partial fulfilment of the

Practice School-I Course No. BITS F221

At

National Centre for Polar and Ocean Research, Goa

A Practice School-1 station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(July, 2019)

Acknowledgements

We would like to express our gratitude to Dr Mayank Goel, for providing his expertise and guidance for our Practice School 1 project. We would also like to thank our mentor, Mr Sakthivel Samy V., Scientist E from the Department of Information Communication Technology at National Centre for Polar and Ocean Research for teaching, guiding and using his vast experience to help us in the project. Finally, we would like to acknowledge the PS Division for giving us this exciting opportunity and the staff of NCPOR for providing the required resources to accomplish the task.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN)

Practice School Division

Station: National Centre for Polar and Ocean Research **Centre**: Goa

Duration: **From**: May 21, 2019 **To**: July 13, 2019

Date of Submission: July 11, 2019

Title of the Project: Time series data analysis with Python and R

ID No.	Name	Discipline
2017A7PS0030P	Swadesh Vaibhav	Computer Science
2017A7PS0031P	Anirudh Goyal	Computer Science
2017A7PS0062P	Gandhi Atith Nileshkumar	Computer Science
2017A7PS0105P	Rohit Milind Rajhans	Computer Science
2017B4A70349P	Ritik Rohit Bavdekar	Maths + Computer Science
2017B5A70831P	Gupta Pranay Pradeep	Physics + Computer Science

Name of Expert: Mr Sakthivel Samy V **Designation:** Scientist E, IT Head, NCPOR

Name of the PS Faculty: Dr Mayank Goel

Key Words: Time series, ARIMA, Long Short Term Memory, Recurrent Neural Network, Convolutional Neural Network, One Class SVM, Isolation Forest, K-means, Fast Fourier Transformation

Project Area: Data Science, Big Data Analysis, Web development, App development

Abstract: In this project different methods for the analysis of time-series data have been used. Machine learning algorithms have been implemented for time-series forecasting. These methods are displayed on the website and in a functional R application. Our project will enable people to gain a deeper insight into the weather that surrounds the coldest regions of the earth. Weather prediction on the website will allow researchers who explore these regions to plan their journeys with great precision.

Table of Contents

Learning Outcomes

On completion of this project, we will be well versed in performing the following tasks:

1. Analysing any type of well-formatted statistical data.
2. Using Machine Learning algorithms towards various tasks.
3. Using Python and R programming languages.
4. Plotting and visually interpreting data, including the usage of Ferret.
5. Designing a website with basic functionality, using Django, PGAdmin4 and PSQL.

Introduction

The problem statement of the project is **"Time Series Data Analysis with Python and R".** The National Centre for Polar and Ocean Research receives weather data from the sensors installed and maintained at the various stations at Antarctica, the Arctic and the Himalayas. This data is then converted into the required format (.csv) by scientists at NCPOR. It is further analysed to come to important conclusions.

The following assumptions/hypotheses must be kept in mind for the project:

1. The data has a seasonality, a trend and a behaviour. No part of the data is 100% random.
2. The sensors used for data collection are mostly accurate. They undergo proper maintenance at regular intervals.
3. The models being implemented don't overfit the data while making predictions.

Data availability:

Indian stations, Bharati, Maitri and Dakshin Gangotri, provide continuous weather data in the form of CSVs. The following table highlights the duration of the available data as well as its frequency.

Table 1: Available data

Station	Data type	Institution	Duration From	Duration To	Data Frequency
Maitri	AWS	IIG	01-01-2012	29-01-2017	Hour
		IMD	01-01-1985	19-12-2016	Hour
		Sankalp SASE	23-02-2006	31-12-2005	Hour
		Dozer SASE	01-03-2007	15-11-2015	Hour
	Surface Data	IMD	27-02-1985	31-12-2010	Hour
Bharati	AWS	IIG	28-01-2012	31-12-2016	Hour
		IMD	06-02-2015	Present	Minute
Dakshin Gangotri	Surface Data	IMD	20-02-1985	31-12-2010	Hour

The following is an example of the data as present in the CSVs.

Table 2: IIG Bharati data

obstime	tempr	rh	ws	wd	ap

1/28/2012 12:00	-0.33	34.24	4.51	155.95	982
1/28/2012 13:00	-0.44	38.07	4.19	149.7	982.02
1/28/2012 14:00	0.02	40.88	4.06	149.46	981.18
1/28/2012 15:00	-0.15	43.44	2.95	118.61	980.6
1/28/2012 16:00	0.06	44.14	3.14	134.17	979.63

Understanding the data:

1. ‘obstime’: This column specifies the observation time in the format ‘%m/%d/%y %H:%M’. It is a unique value and frequency of observations varies for different datasets.
2. ‘tempr’: This column specifies the value of temperature in Celsius.
3. ‘rh’: This column specifies the value of relative humidity as a percentage.
4. ‘ws’: This column specifies the value of wind speed in knots.
5. ‘wd’: This column specifies the value of wind direction in degrees.
6. ‘ap’: This column specifies the value of air pressure in hPa.

The naming convention is consistent throughout this report and in all the datasets. Any missing values in the dataset is replaced with ‘-999’ according to standard scientific conventions.
Code is presented in Appendix B.1.

The data in CSVs is stored in a computer database so that it can be used by applications for further analysis.

Code is presented in Appendix B.2.

Data pre-processing:

The data directly available from the stations is raw and not suitable for analysis. It has to be pre-processed before applying analysis and prediction techniques.
Before performing any operation on the data, all the column names must be verified. The time axis must be set as the index for the dataframe. Also, all values equaling -999, i.e. entries having no data, must be ignored.

```
# df is the dataframe that stores the data
df['obstime'] = pd.to_datetime(df['obstime'])
# time axis set as index
df = df.set_index('obstime')
# removing extraneous data
df = df[~(df[col] == -999)]
df = df.dropna()
```

Fig 3 - Data cleaning

Weather data that is a form of time series data often requires more preparation before being modelled with machine learning algorithms. Time series data contain trend and seasonal component that can affect the results of the algorithms. The trend shows the general tendency of data to increase or decrease over a long period of time whereas seasonal variations are rhythmic forces that operate in a regular and periodic manner over a span of less than a year.

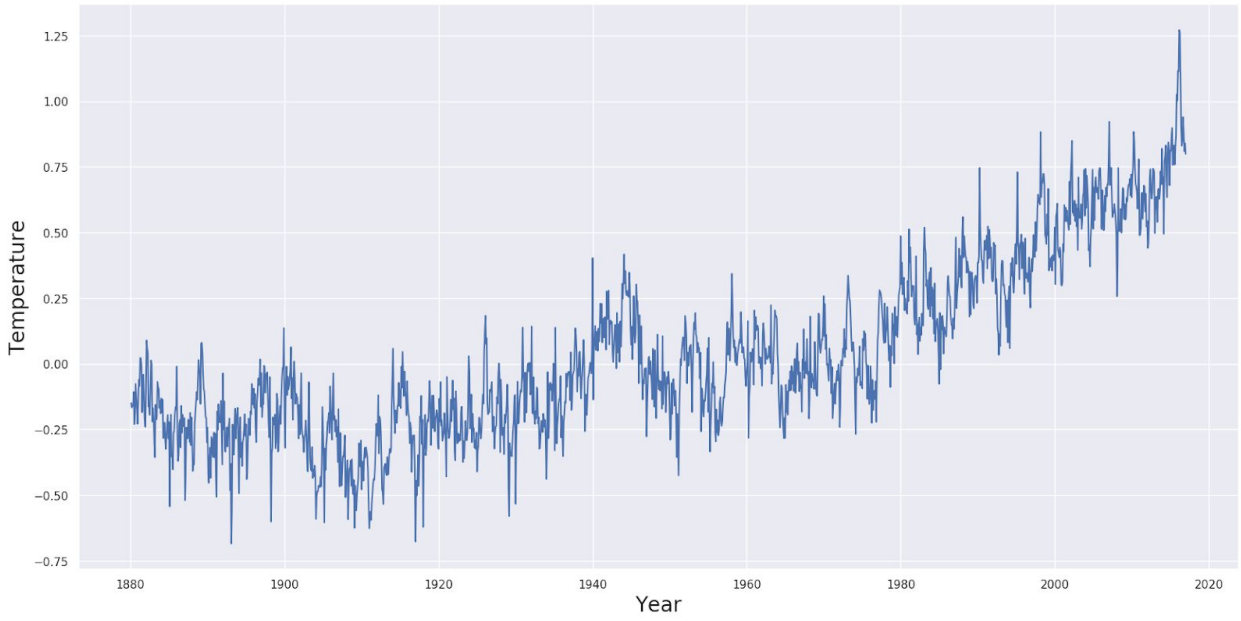


Fig 4- Upward trend in temperature data

The trend in time series data is calculated using the moving average method. A moving average (MA) is an indicator that filters out the noise from random short-term fluctuations. It is a trend-following indicator as it is based on past observations.
A simple moving average is an average of 'n' consecutive data points where n is the window for calculation.

$$SMA = \frac{A_1 + A_2 + ... + A_n}{n}$$

Fig 5- Formula for Simple Moving Average. 'N' is the moving average window

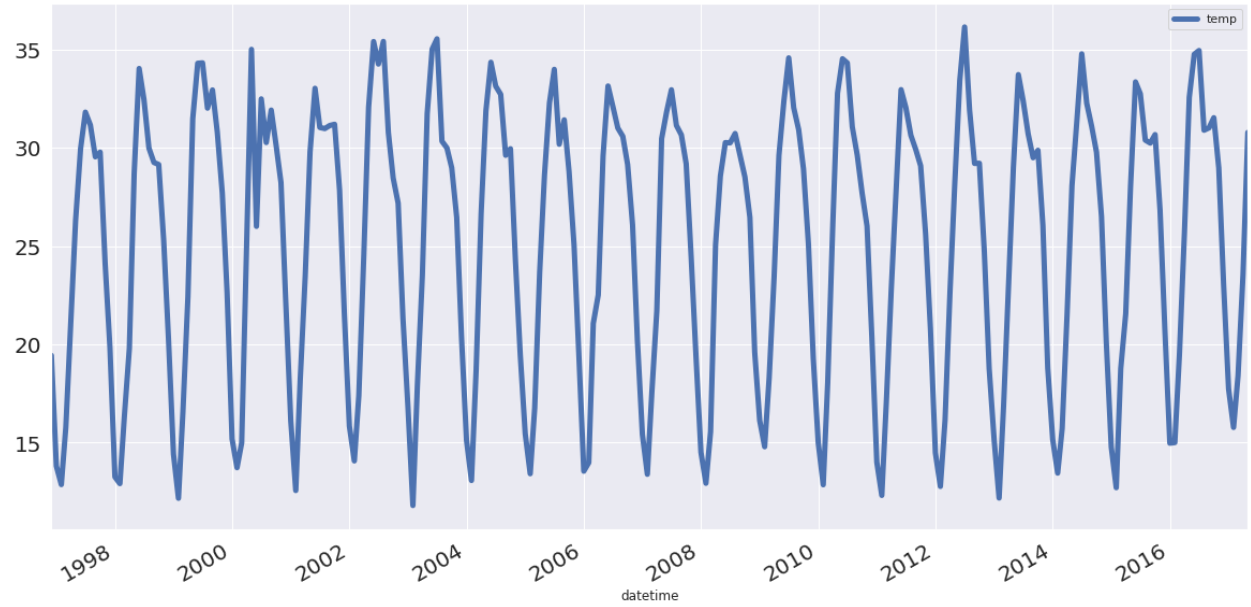


Fig 6- Seasonality in temperature data observed at an interval of 12 months

Seasonality in data can be easily observable through the graph. In the case of weather data, seasonality is generally observed at an interval of twelve months.
It is easier to model stationary time series data. A stationary time series is the one whose statistical properties such as mean, variance and autocorrelation are constant over time. A time series is not stationary if they have trends or seasonal effects. Classical time series analysis and forecasting methods are concerned with making non-stationary data stationary by identifying and removing trends and seasonal effects.

Differencing is a simple method to remove seasonality in data. A lagged difference with the width equal to the seasonal period is considered for this method. For example, to remove seasonality in the data plotted in Fig 6, a lagged difference is taken between a data point and another point 12 months prior.

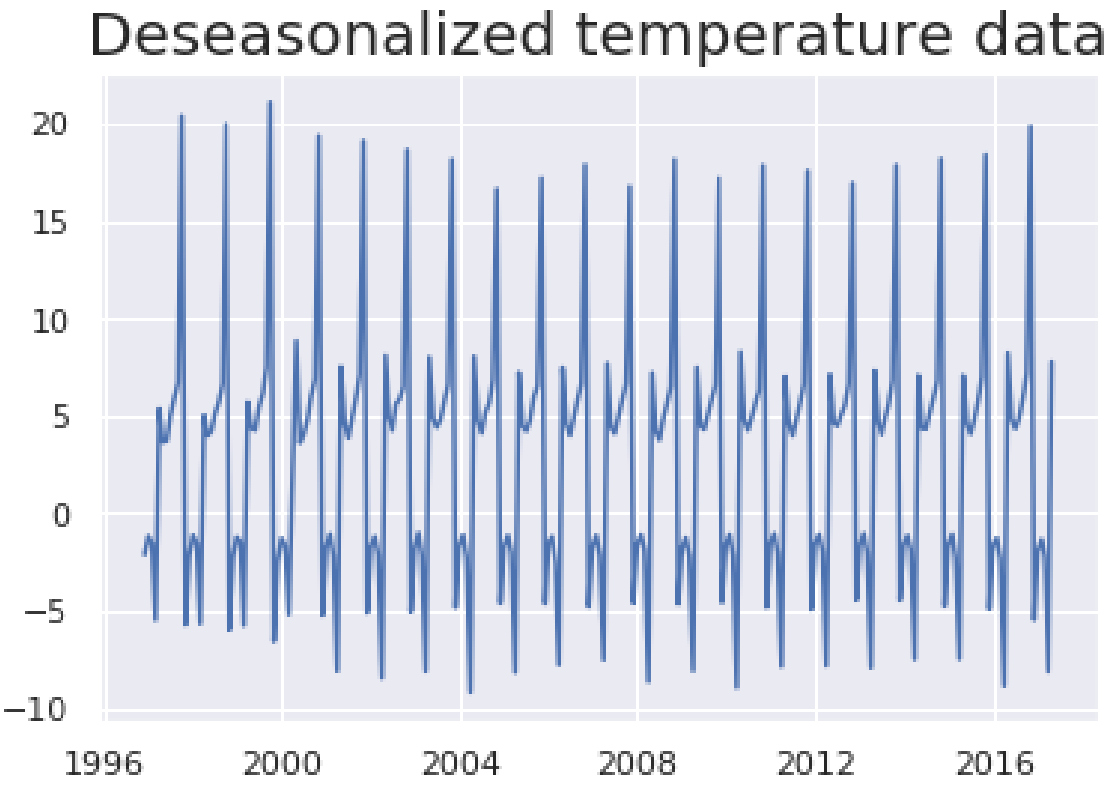


Fig 7- Data after removing seasonality

Python libraries exist that automatically detect and remove trend and seasonality. These functions fit the time series data into an additive or multiplicative model.

Additive Model: Data = Seasonal effect + Trend + Cyclical + Residual
Multiplicative Model: Data = Seasonal effect * Trend * Cyclical * Residual
Based on the model, the function then decomposes data into its resulting components.

```
# Deseasonalize temperature data
from statsmodels.tsa.seasonal import seasonal_decompose
# Data is considered as an additive model of four components
# Time Series Data = Trend + Seasonality + Cyclic + Residual
# temp is the dataframe
result_add = seasonal_decompose(temp.temp.values, model='additive',
```

```
freq=12)

deseasonalized_df = temp.temp.values / result_add.seasonal
plt.plot(temp.index, deseasonalized_df)
plt.title('Deseasonalized temperature data', fontsize=20)
```

Fig 8- Function to decompose data into components

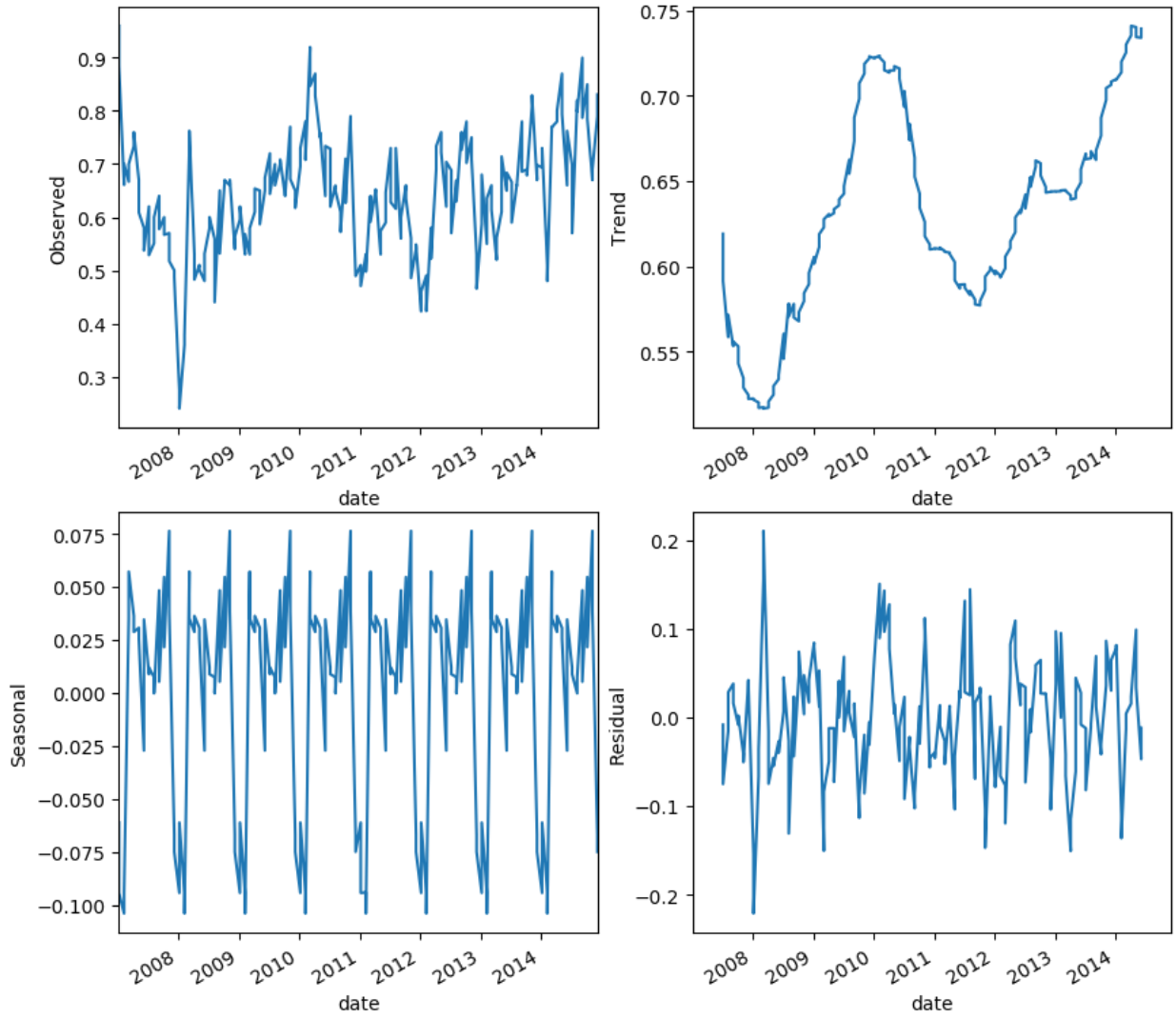


Fig 9- Result of decomposition

Anomaly Detection:

Although the data has been standardized, it may still contain some inconsistent values. These values may occur due to noise in data, fault in the instrument, human errors etc. Such inconsistent values that do not conform to the expected behaviour are called anomalies or outliers. Anomaly detection is a necessary technique to identify such anomalies and remove them from the data for better analysis and prediction.

Anomalies can be broadly categorized into 3 groups:

1. Point anomalies: A single instance of the data is anomalous if it is too far from the rest of the data.
2. Contextual anomalies: The abnormality is context specific. It is common in time-series data.
3. Collective anomalies: A set of data points collectively detect anomalies.

There are several techniques used for anomaly detection. Following are some of them.

1. Simple Statistical methods: It identifies irregularities in data by flagging data points that deviate from common statistical properties of a distribution, including mean, median or mode.
    a. Interquartile Range Method: It is a measure of statistical dispersion equal to the difference between the 75th and 25th percentiles. The algorithm flags those values as outliers that lie beyond 1.5 times the Interquartile Range.

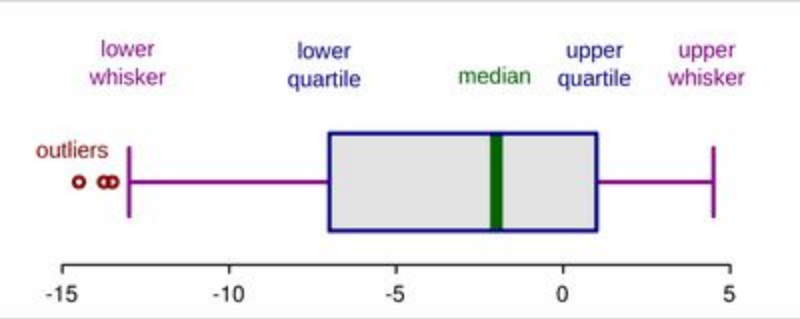


Fig 10- Interquartile Range Method

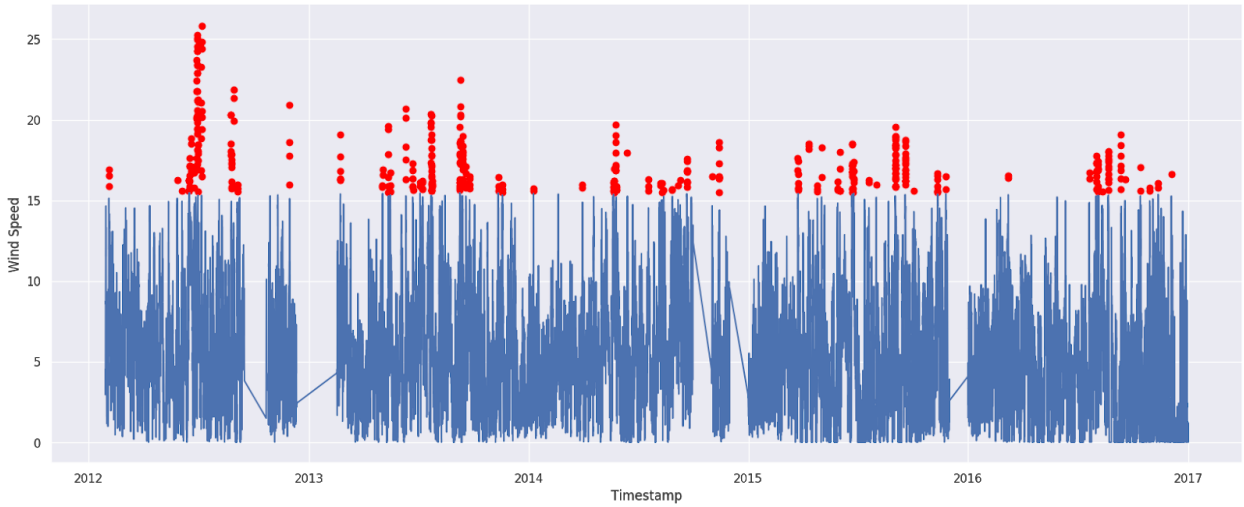


Fig 11- Anomaly detection using Interquartile Range Method on Wind Speed data

2. Machine Learning based methods: Statistical models fail to identify seasonality patterns in data. If the data contains noise, statistical models will not consider it. Machine learning algorithms identify the trends in data. As the weather data is unlabelled, unsupervised machine learning techniques had to be used. These techniques take as input the data and a select number of features. The features considered for any attribute of the data for anomaly detection are the value of that attribute, month of the year, season to which it belongs, day of the month. These features resulted in the best output amongst many other features. The outlier percentage is set at 0.5. This implies the algorithm will classify 0.5% of the data as an outlier.

    a. K-means: K-means is an unsupervised learning technique used for unlabelled data. Data in this use case is also unlabelled (i.e., data does not have defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable *K*. The algorithm works iteratively to assign each data point to one of the *K* groups based on the features that are provided. Data points are clustered based on feature similarity. The algorithm returns the centroid of each of the K clusters which can be used to label the data. Also, the training data is labelled according to the cluster to which it belongs.

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left\| x_i - v_j \right\| \right)^2$$

Fig 12- Formula for calculating distances from the clusters
x_i: the point from where distance is to be calculated

$v_{j:}$ centroid of a cluster
c: number of clusters
c_i : number of points in the *ith* cluster

The K-means algorithm requires the number of clusters, K as an input. A graph between the number of clusters and the score for each cluster is plotted. K can be calculated from the graph.

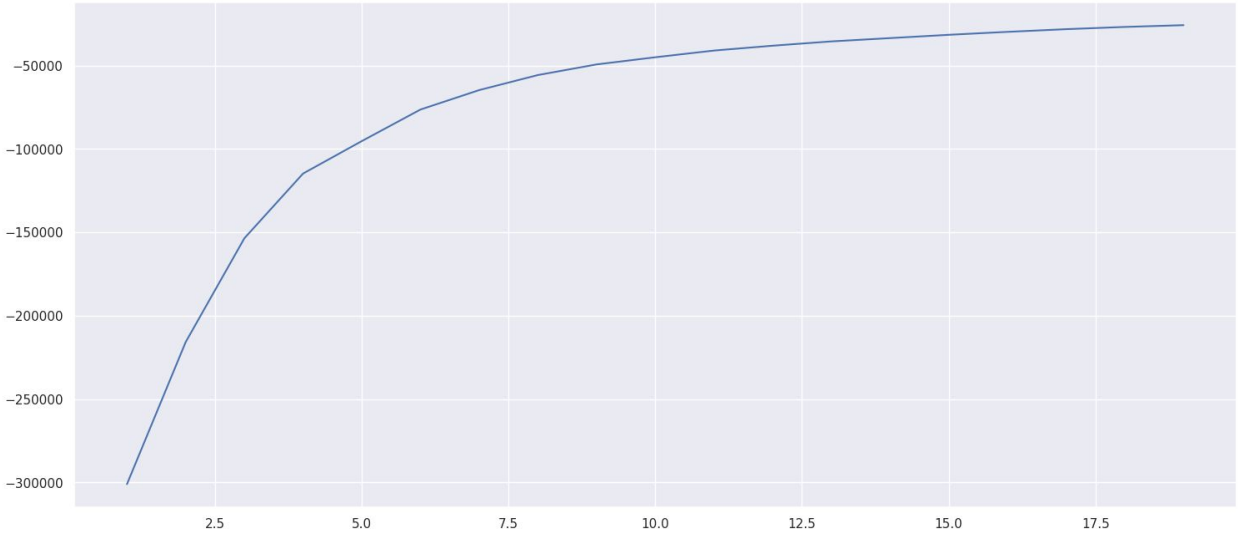


Fig 12- K-means: Number of clusters(x) vs Score(y)

The minimum value of K after which the score remains almost constant is chosen as the number of clusters. In our use case, the number of clusters has been fixed at 10.

Code presented in Appendix A.1.

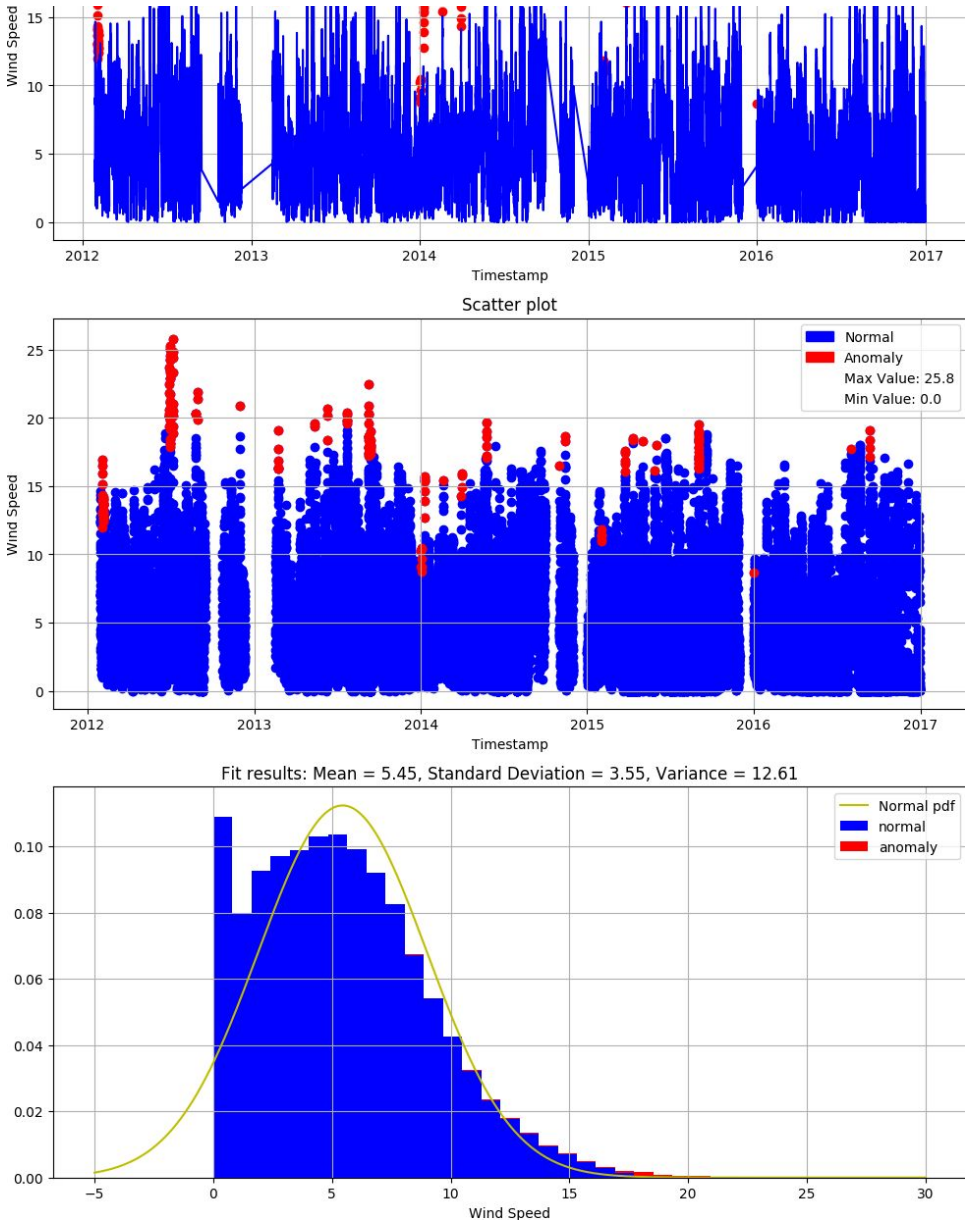


Fig 13- K-means algorithm results on wind speed data

b. Isolation Forest: This algorithm is based on the fact that anomalies are few and different. As a result of these properties, anomalies are susceptible to a mechanism called isolation. It has low linear time complexity. It builds a good performing model with a small number of trees using small sub-samples of fixed

size, regardless of the size of the data. The algorithm isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. Isolating anomalous observations is easier because only a few conditions are needed to separate them from normal observations. An anomaly score, is thus, based on the number of conditions required to separate a given observation. The way that the algorithm constructs the separation is by first creating isolation or random decision trees. The score is calculated as the path length to isolate the observation.
Code presented in Appendix A.2.

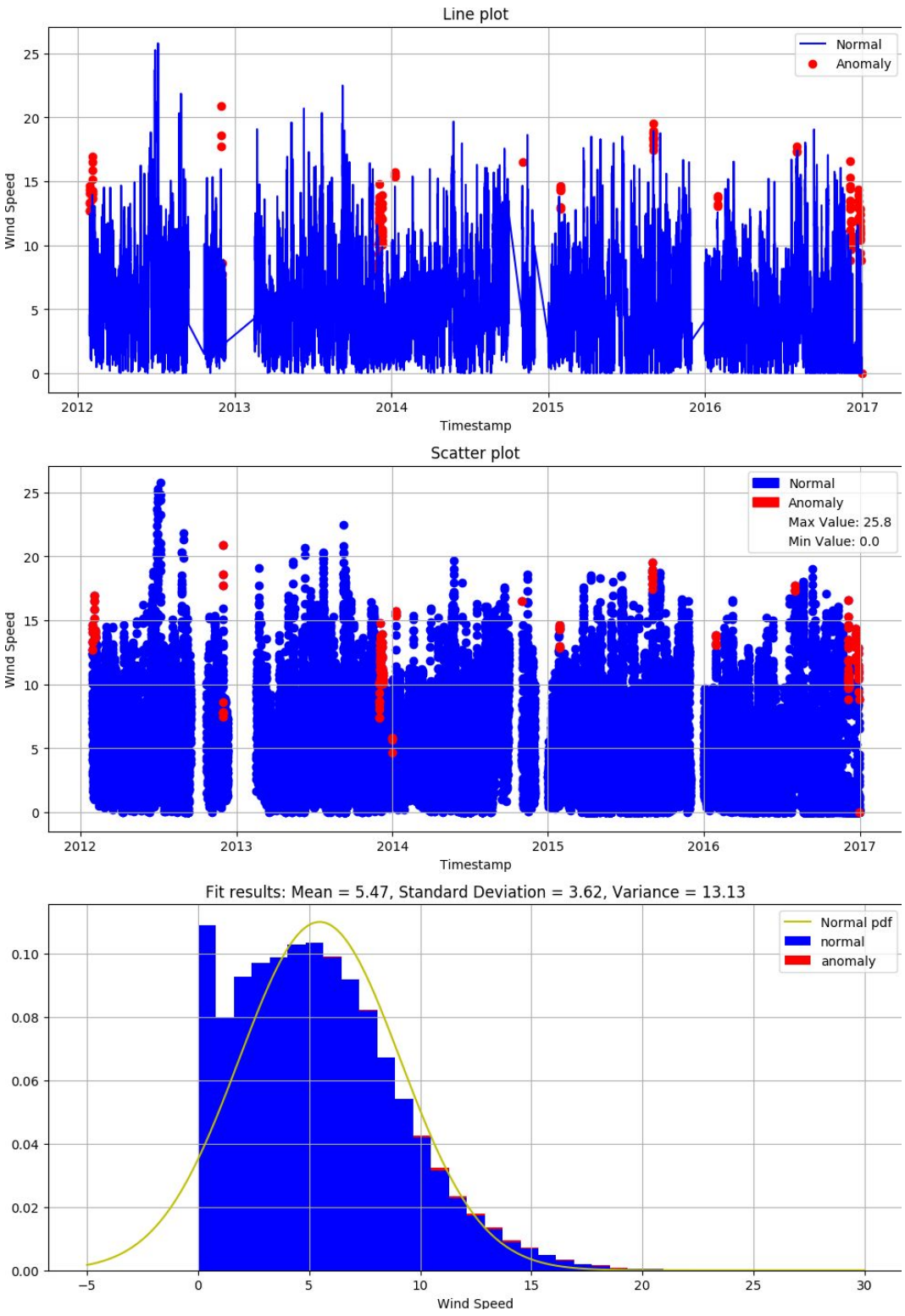


Fig 14- Isolation forest on wind speed data

c. One Class SVM: It is a novelty detection technique. The idea of novelty detection is to detect rare events. Support Vector Machines are max-margin

methods. The idea is to find a function that is positive for high-density regions and negative for low-density regions. In one class SVM, the support vector model is trained on the data that has only one class, the ‘normal’ class. It infers the properties from normal cases and from these properties predicts which examples are unlike the normal examples.
Code presented in Appendix A.3.

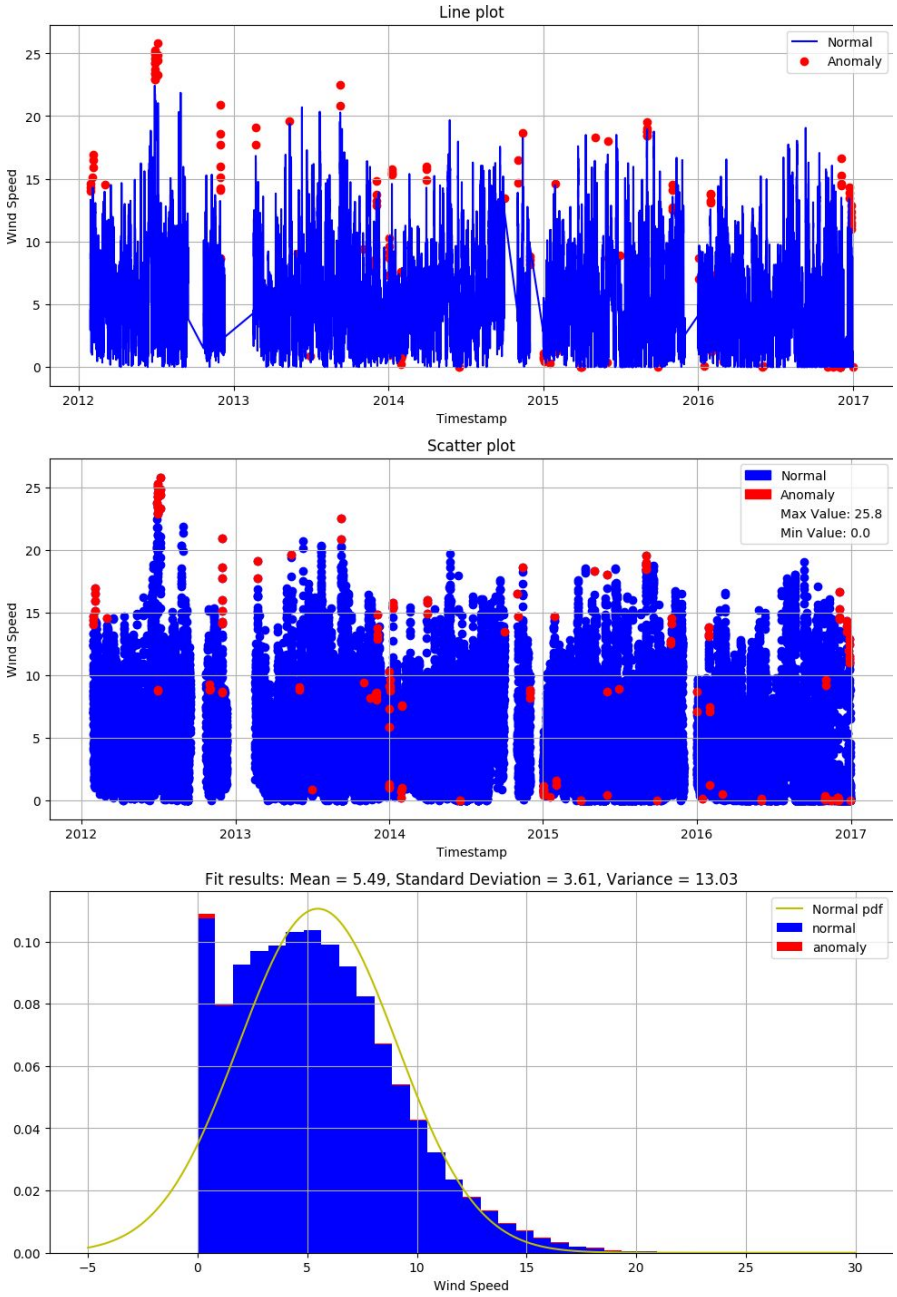


Fig 15- One Class SVM on wind speed data

Data Analysis:

Once the data has been processed, it is analysed using different plots to identify trends, patterns, correlation etc.

Seasonal Analysis:

Seasonal analysis of weather data is important to realise the change in weather patterns across seasons in the Antarctic. The Antarctica has four seasons-

1. Summer: December, January, February
2. Fall: March, April, May
3. Winter: June, July, August
4. Spring: September, October, November

Seasonal analysis for Antarctic weather data is done in three different ways.

1. Quarterly Plot: Data for a given year is plot for all the four seasons. This plot is used to compare the climate across different seasons within the same year.
   Code is presented in Appendix C.1.

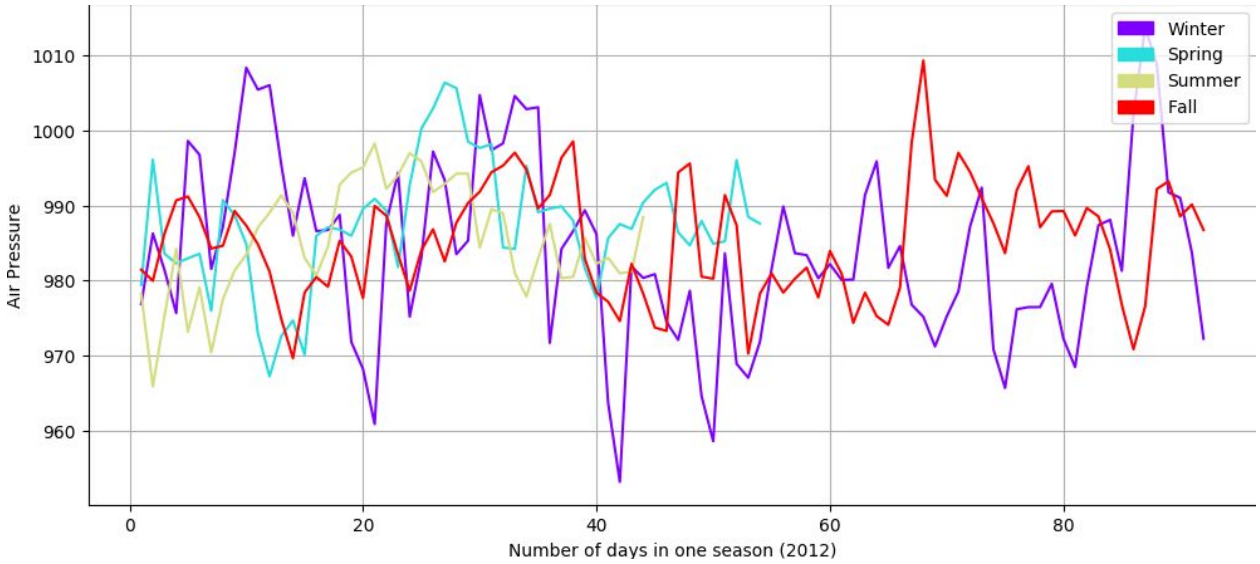


Fig 15- Quarterly plot of air pressure in the year 2012

2. Year Wise Comparison: Data is plot for a range of years. This plot is used to compare the change in climate every year in the selected range.
   Code is presented in C.2.

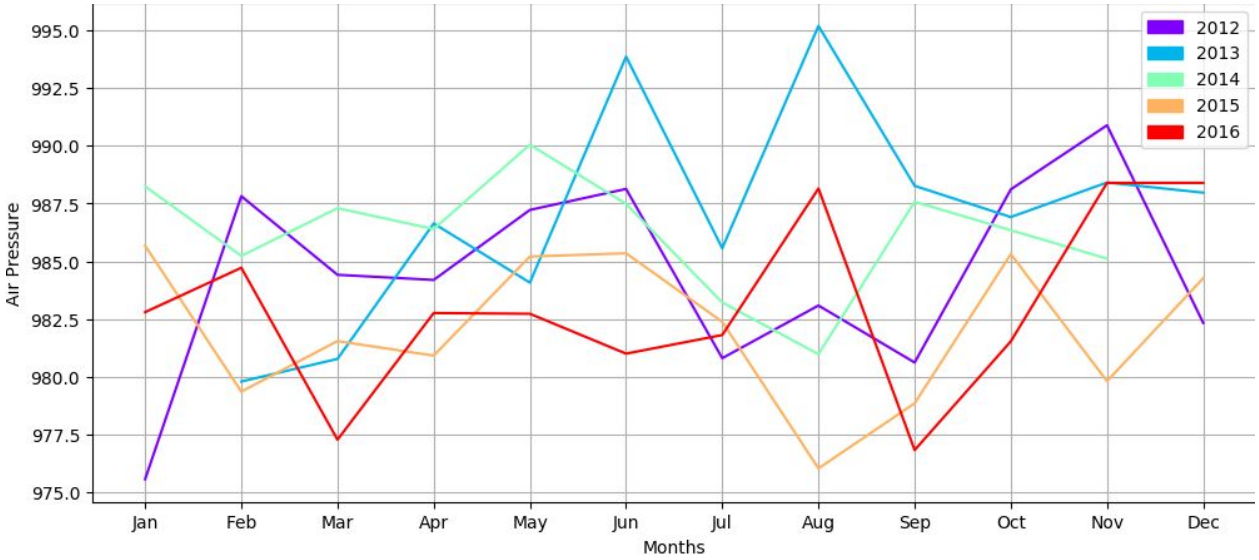


Fig 16- Year wise comparison plot of air pressure from 2012-2016

3. Seasonal Comparison: This plot is used to compare a specific season across different years. It gives an idea of how a specific season has undergone climate change along the years.
   Code is presented in C.3.

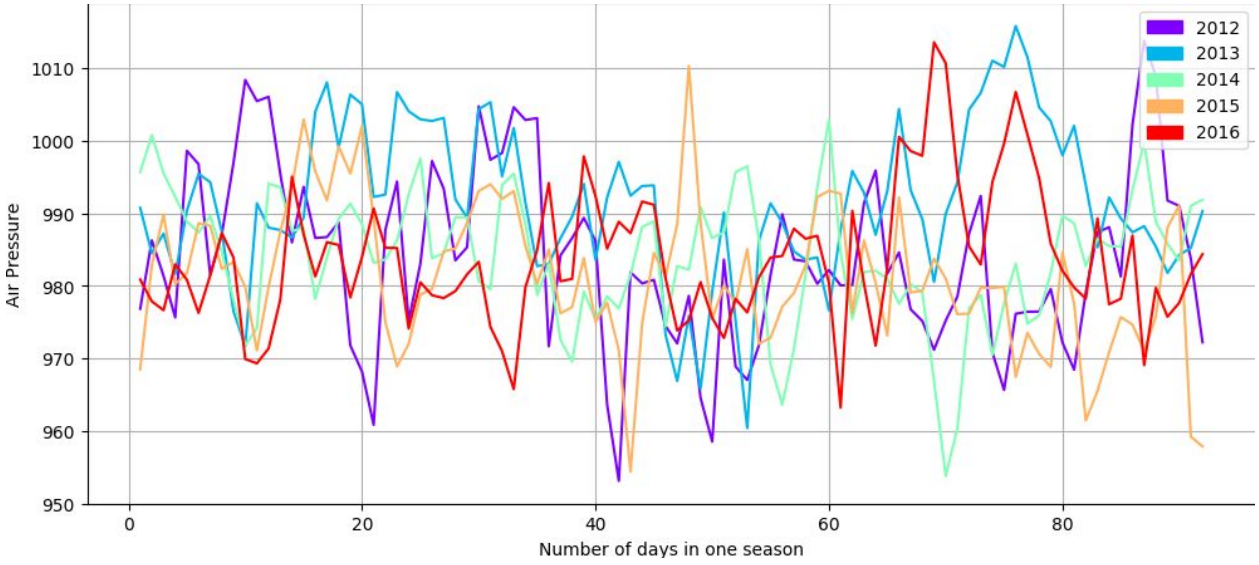


Fig 17- Seasonal comparison of air pressure during winters from 2012-2016

Fast Fourier Transformation

Fourier analysis is the study of the way to approximate a general function into simple trigonometric functions.The decomposition process is called Fourier Transformation. The discrete Fourier transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency.The Discrete Fourier Transformation is calculated with the help of the following formula-

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \qquad \textit{(Eq.1)}$$
$$= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)],$$

Fig 18- Formula for calculating Fast Fourier Transformation

It has many practical applications in digital signal processing , image processing , differential equation solutions , etc.

A Fast Fourier Transform is an algorithm which computes the Discrete Fourier Transformation of a sequence.It has a time complexity of O(nlogn) unlike others which have O(n*n). Cooley Turkey algorithm is the most commonlyhT used FFT algorithm.It re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size N = N 1 N 2 in terms of N 1 smaller DFTs of sizes N 2 , recursively, to reduce the computation time to O(NlogN)

FFT has been used in this project to analyse the data and find patterns. These patterns can be used to identify the periodicity of the data and can help in forecasting. Python and Ferret(Programming language used for weather data analysis) were used to plot the Fourier Transformation of the data. In Python SciPy ,Matplotlib and plotly libraries were used to plot the same.

The code snippet for Python with mathplotlib library is given below-

```
import datetime
import numpy as np
import scipy as sp
import scipy.fftpack
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
data=pd.read_csv('iig_maitri.csv')
data['obstime']=pd.to_datetime(data['obstime'],format='%m/%d/%Y
%H:%M')
data=data.set_index('obstime')
date=data.index
ws = data['ws']
temp=data['tempr']
rh=data['rh']
wd=data['wd']
ap=data['ap']
N = len(temp)
temp_fft = sp.fftpack.fft(temp)
temp_psd = np.abs(temp_fft) ** 2
temp_fftfreq = sp.fftpack.fftfreq(len(temp_psd), 1. / (365*24))
temp_i = temp_fftfreq > 0
ws_fft = sp.fftpack.fft(ws)
ws_psd = np.abs(ws_fft) ** 2
ws_fftfreq = sp.fftpack.fftfreq(len(ws_psd), 1. / (365*24))
ws_i = ws_fftfreq > 0
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
ax.plot(ws_fftfreq[ws_i],(ws_psd[ws_i]))
ax.set_xlim(0, 100)
ax.set_xlabel('Frequency (1/year)')
ax.set_ylabel('Amplitude')
plt.title('Fourier Transform of Wind Speed')
plt.grid(True)
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
ax.plot(temp_fftfreq[temp_i],(temp_psd[temp_i]))
ax.set_xlim(0, 100)
ax.set_xlabel('Frequency (1/year)')
ax.set_ylabel('Amplitude')
plt.grid(True)
plt.title('Fourier Transform of Temperature')
```

Fig 19 - Code Snippet for FFT with matplotlib library

The following results were obtained-

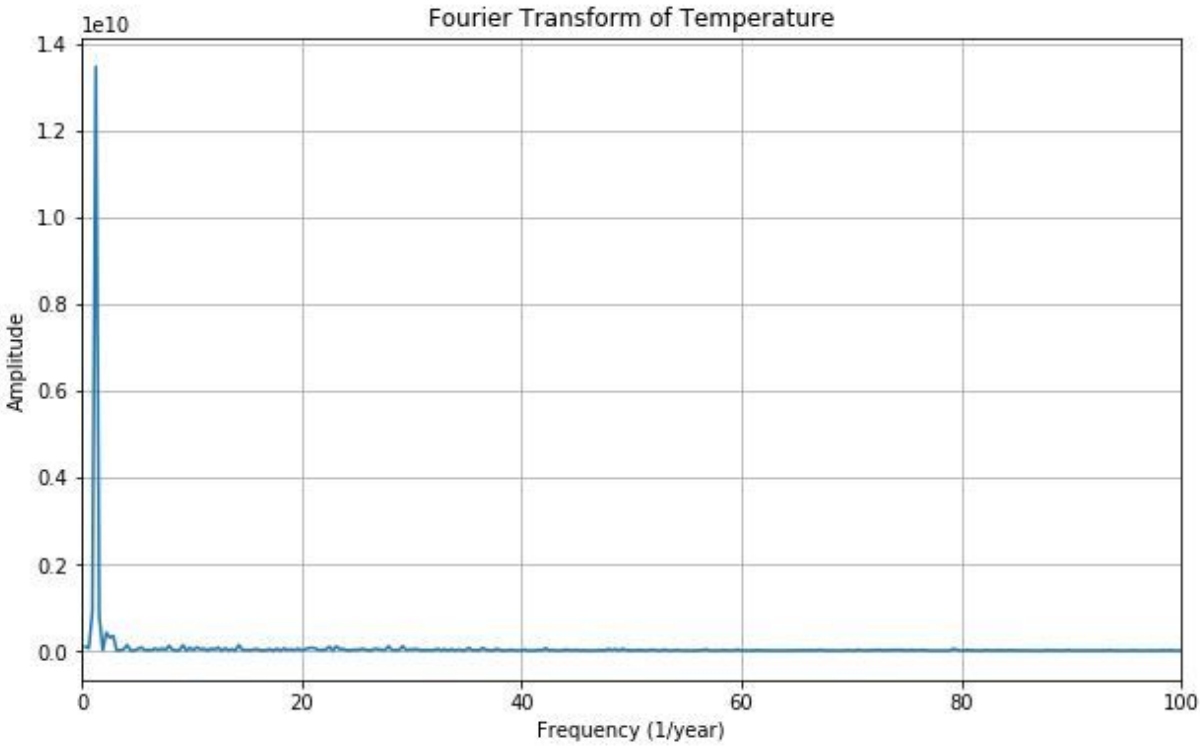


Fig 20 - Fast Fourier Transformation of Temperature

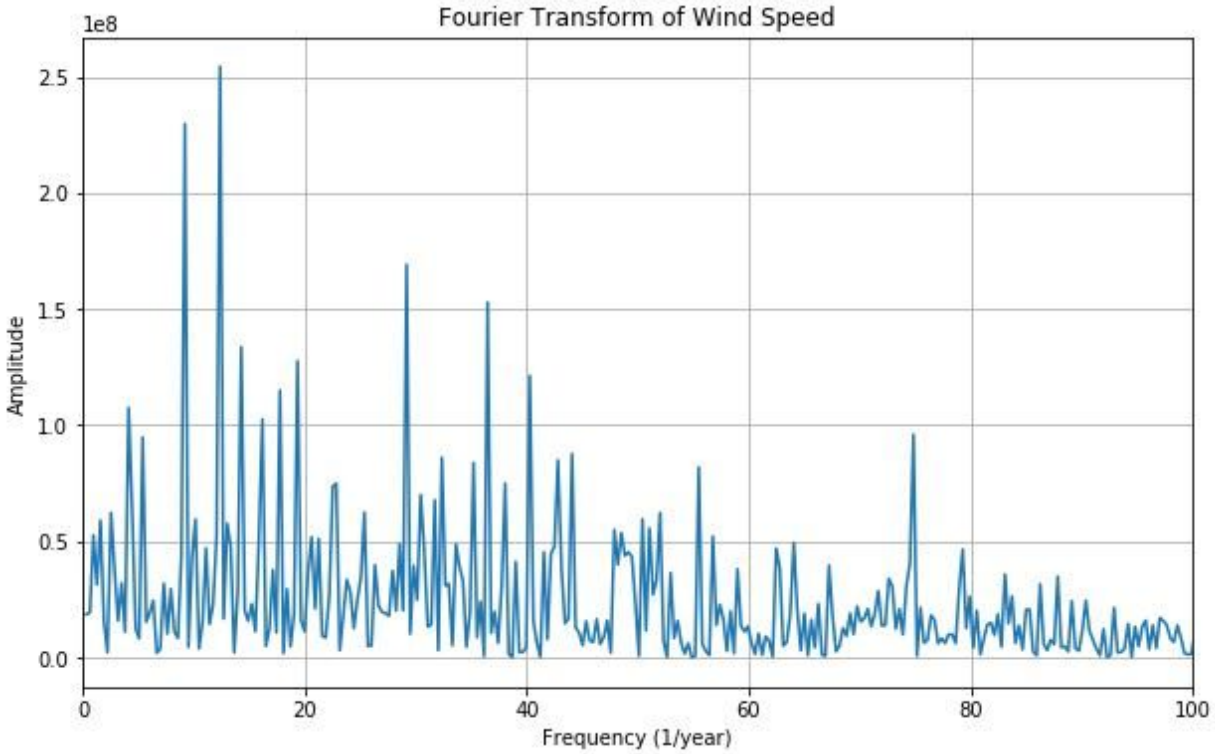


Fig 21 - Fast Fourier Transformation of Wind Speed

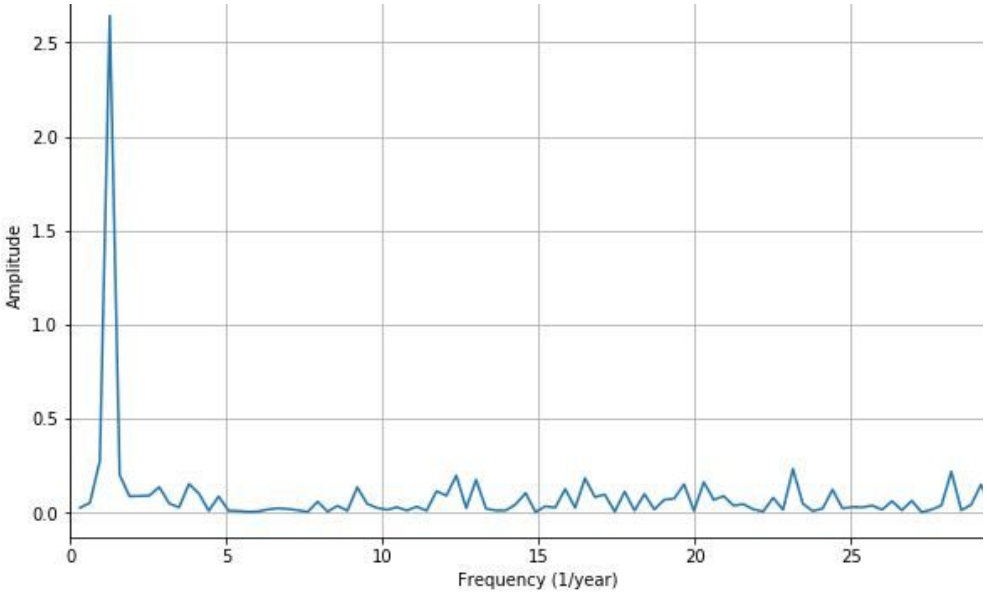


Fig 22- Fast Fourier Transformation of Air Pressure

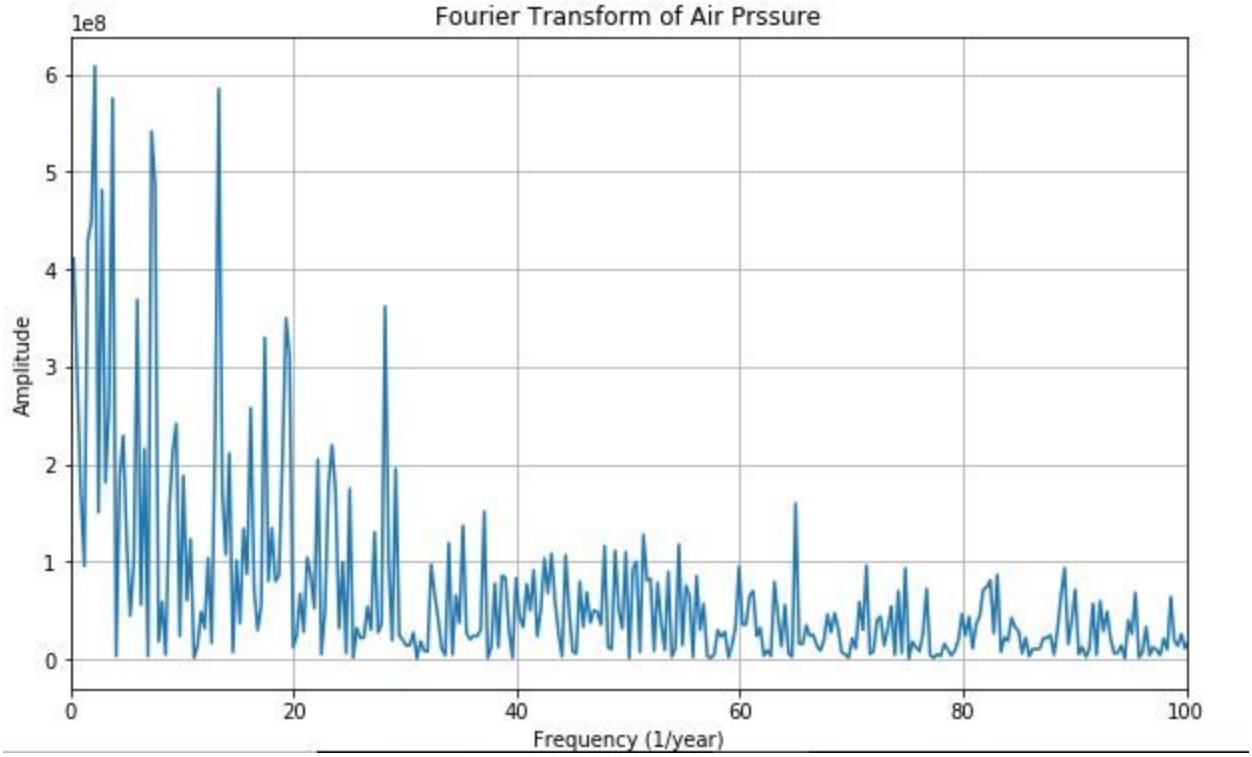


Fig 23 - Fast Fourier Transformation of Wind Direction

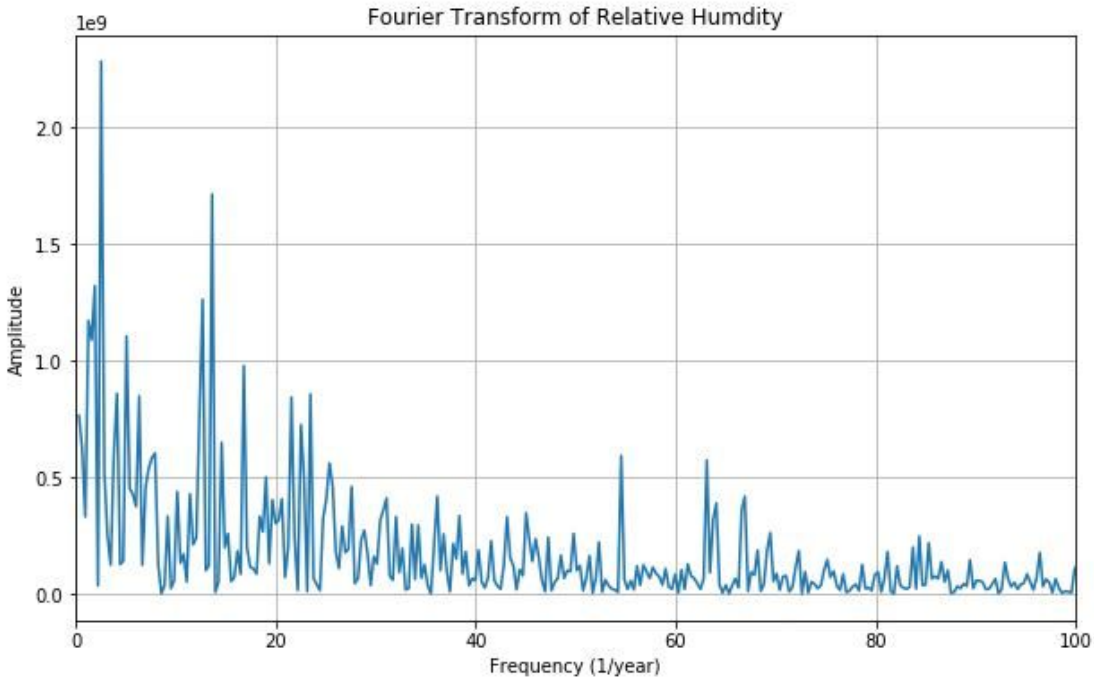


Fig 24 - Fast Fourier Transformation of Relative Humidity

The code snippet for Python with plotly library is given below-

```
ws_fft = sp.fftpack.fft(ws)
ws_psd = np.abs(ws_fft) ** 2
ws_fftfreq = sp.fftpack.fftfreq(len(ws_psd), 1. / (365*24))
ws_i = ws_fftfreq > 0
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
ax.plot(ws_fftfreq[ws_i],(ws_psd[ws_i]))
ax.set_xlim(0, 100)
ax.set_xlabel('Frequency (1/year)')
ax.set_ylabel('Amplitude')
plt.title('Fourier Transform of Wind Speed')
plt.grid(True)
plotly.offline.plot({
      "data": [go.Scatter(x=ws_fftfreq[ws_i],y=ws_psd[ws_i])],
      "layout" : go.Layout(title='Fourier Transform of Windspeed',
xaxis=dict(title='Frequency(1/hour)'),yaxis=dict(title='Amplitude'))
 }, auto_open=True)
```

Fig 25- Code Snippet for Ferret FFT plot

Interactive graphs were plotted and put up on the NCPOR data portal which are as following:

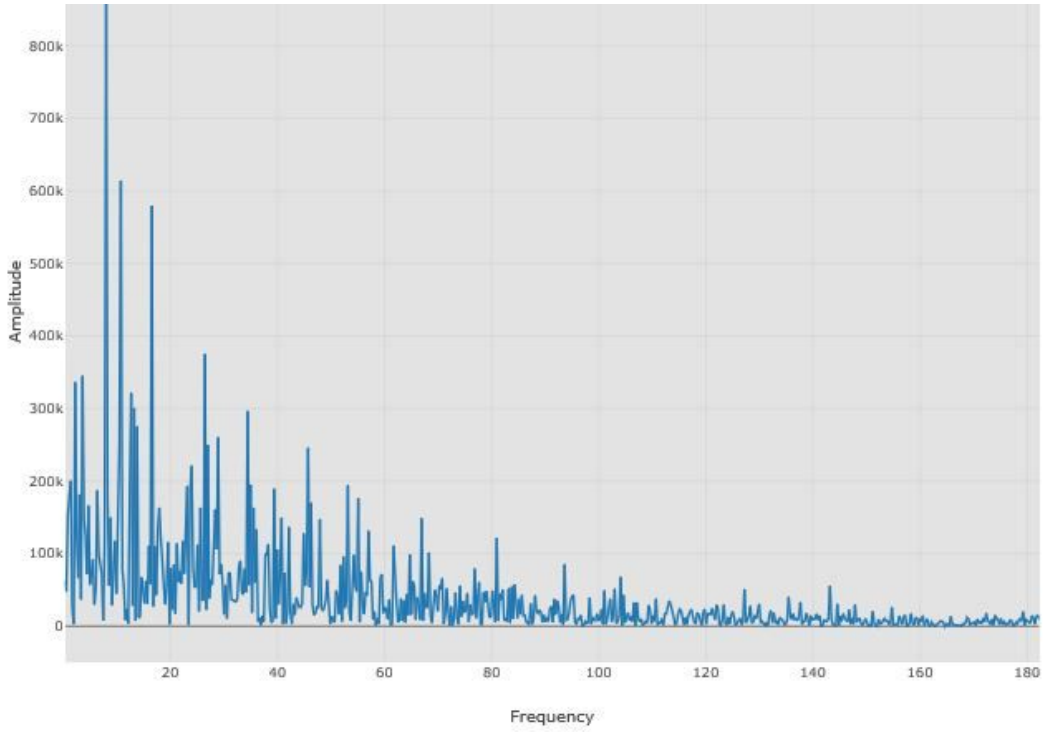


Fig 26: Fourier Transformation plot using Plotly Library

Ferret was also used to plot FFT[6]. It involved converting the data from .csv to a Netcdf file. The code for processing the data is provided in Appendix A.4

After this code was run on the csv file a new processed csv file was created which didn't have data repetition and had data in a monotonic time order. After that the following code was run to convert the data to a netcdf file.

```
ind = np.argsort(np.array(obstime).reshape((len(obstime),)))
print(ind.shape)
#ind = ind[:].astype(int)
time[:]=date2num(np.sort(obstime),units=units, calendar=calendar)
temperature[:]=np.array(tempr)[ind]
relhumid[:]=np.array(rh)[ind]
windspeed[:]=np.array(ws)[ind]
winddirection[:]=np.array(wd)[ind]
airpress[:]=np.array(ap)[ind]
# num.units = units
time.units = units
```

```
ncout.close()
```

Fig 27-Code Snippet

Then a ferret script was run to plot the graph. A snippet has been provided below

```
use iig_maitri.nc
#data that can be used for plotting fft is 31-Jan-2012 : 25-Dec-2015
define axis/t=31-Jan-2012:25-Dec-2015:1/units=hour tax
let input = $1
let wsv = input[gt=tax]
let fftws = ffta(wsv)
let frq_ws = t[gt=fftws]
let per_ws =1/frq_ws/24
plot/vs/line/hlimit=0:10 per_ws, fftws
```

Fig 28-Code Snippet

The following graphs were plotted

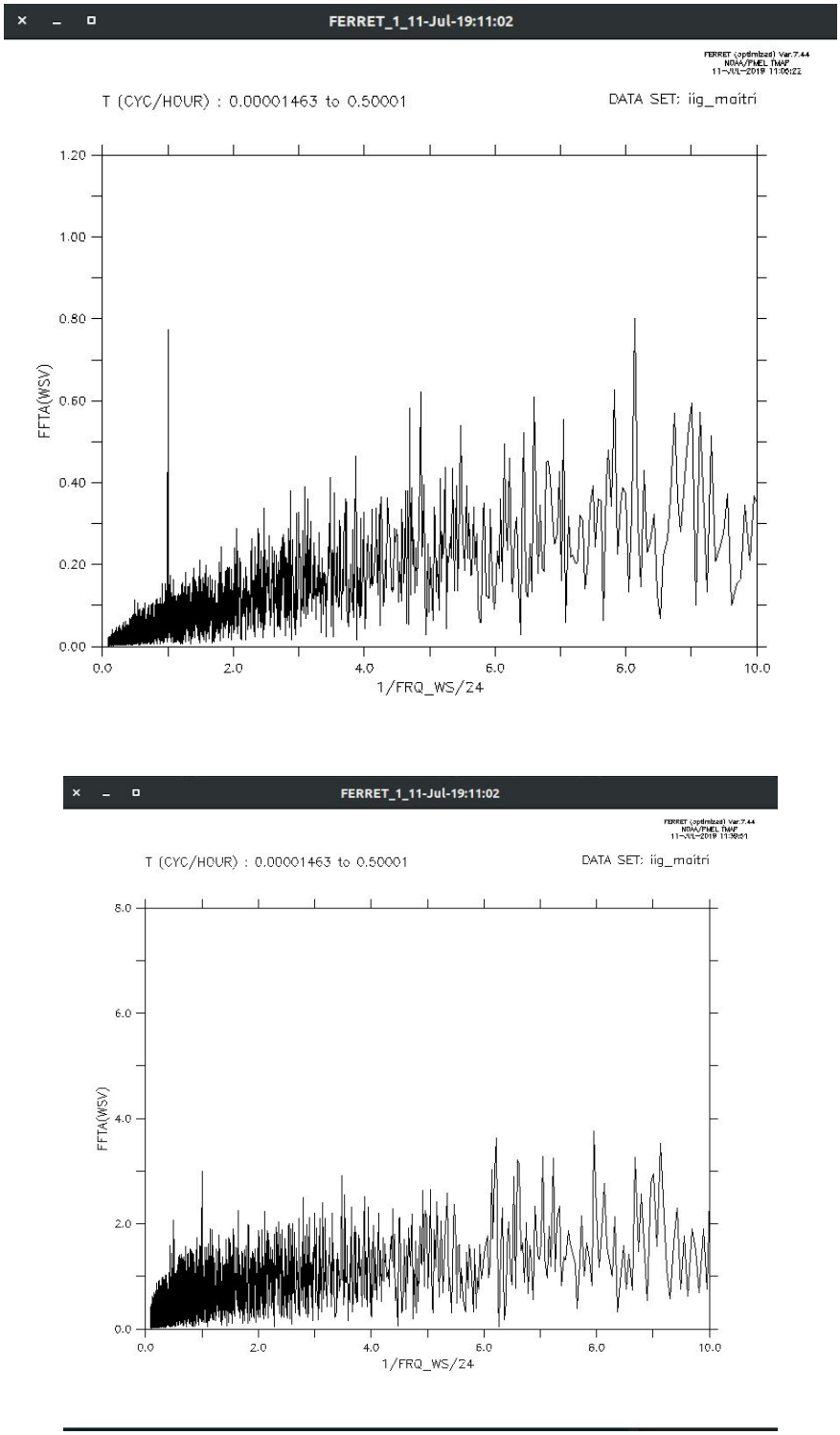
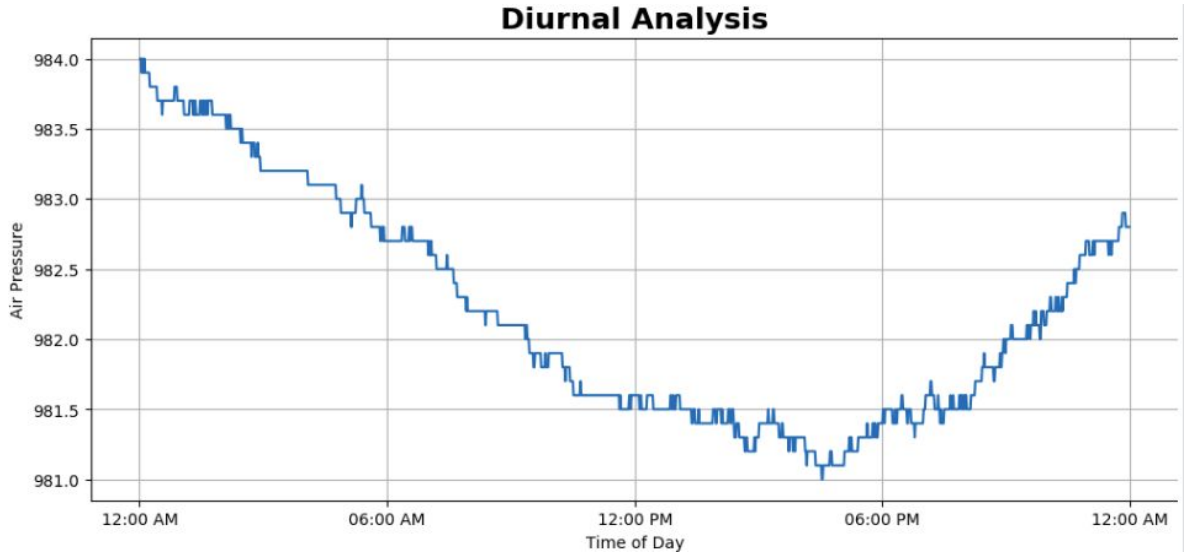
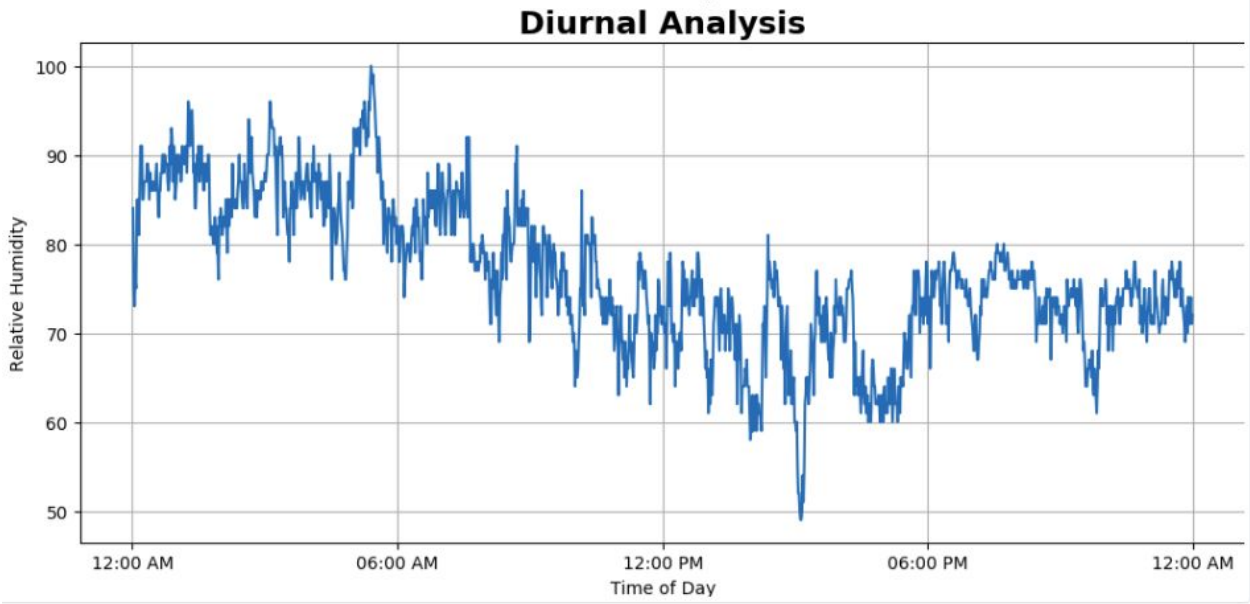


Fig 29: FFT using Ferret

Diurnal Analysis:

Diurnal analysis involves plotting a particular day's weather conditions on a minute to minute basis. It helps in analysing how the weather conditions of a particular place change with time.The graphs of temperature, air pressure , wind speed, wind direction and relative humidity were plotted. Many trends and relations between the above parameters were observed and noted for further research purpose.
The code snippet is given in Appendix A.5
This code was uploaded on the website and the following graphs were plotted:


Diurnal Analysis
Air Pressure
984.0
983.5
983.0
982.5
982.0
981.5
981.0
12:00 AM
06:00 AM
12:00 PM
06:00 PM
12:00 AM
Time of Day


Diurnal Analysis
Relative Humidity
100
90
80
70
60
50
12:00 AM
06:00 AM
12:00 PM
06:00 PM
12:00 AM
Time of Day

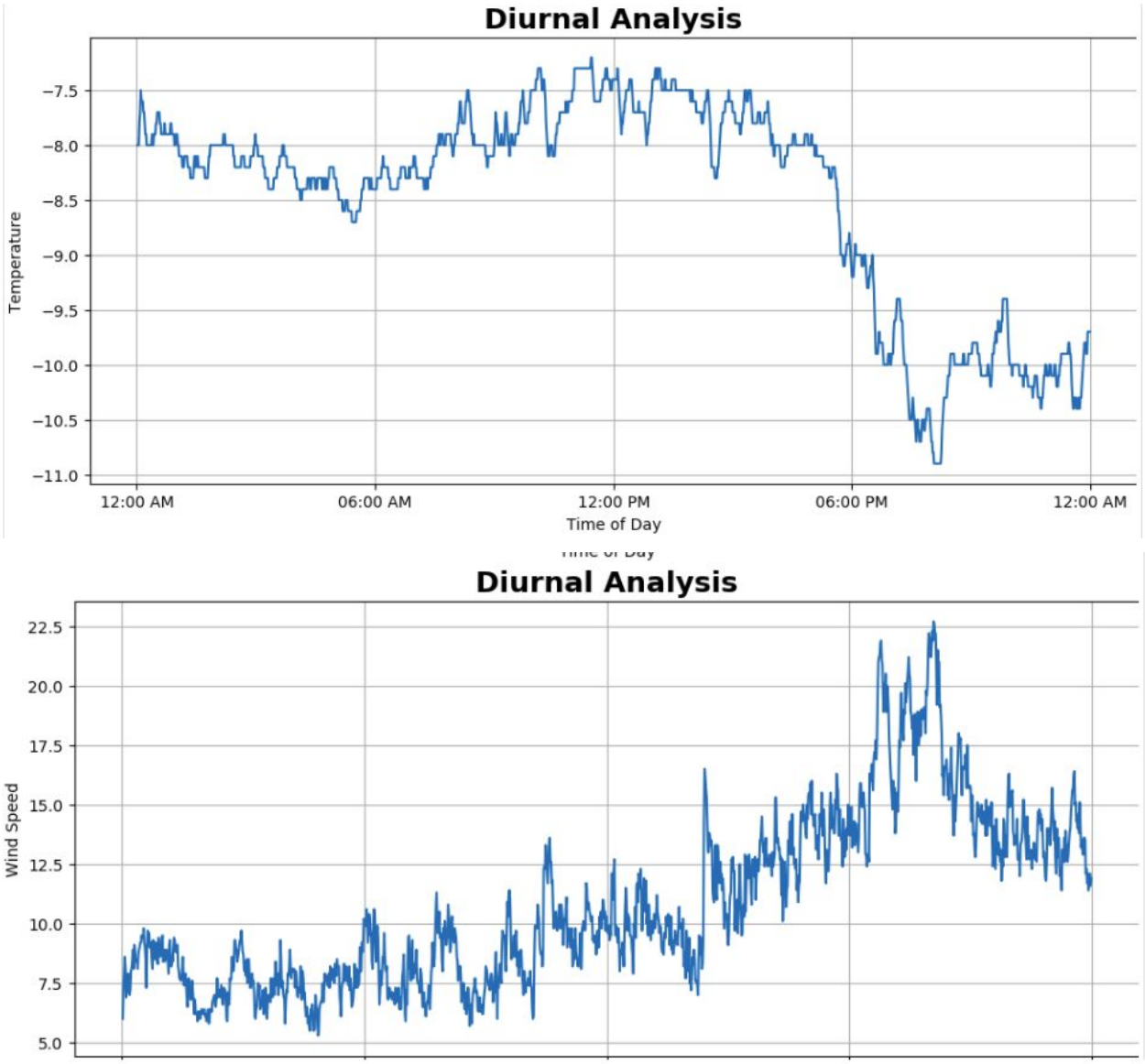


Fig 30 : Diurnal Analysis plots of Temperatire,Wind speed , Air Pressure and Relative Humidity

Scatter (Correlation) Matrix:

Correlation is a tool used in multivariate statistics to find relations between different variables.A Scatter Matrix is an estimation of covariance matrix when calculating covariance is difficult or not possible.A scatter plot which when approximated to a line with a positive slope has a positive correlation ie if one variable increases the other also increases. A scatter plot with a negative slope has a negative correlation ie if one variable increases the other decreases.

The scatter matrix is computed by the following equation:

$$S = \sum_{k=1}^{n} (\boldsymbol{x}_k - \boldsymbol{m})\,(\boldsymbol{x}_k - \boldsymbol{m})^T$$

where $\boldsymbol{m}$ is the mean vector

$$\boldsymbol{m} = \frac{1}{n} \sum_{k=1}^{n} \boldsymbol{x}_k$$

The following code was used to calculate and plot the scatter matrix of the data. Four different plots were made for the four different seasons namely summer, fall, winter and spring. The data had five different parameters- temperature , wind speed , wind direction , relative humidity and air pressure. Relations between them were observed.

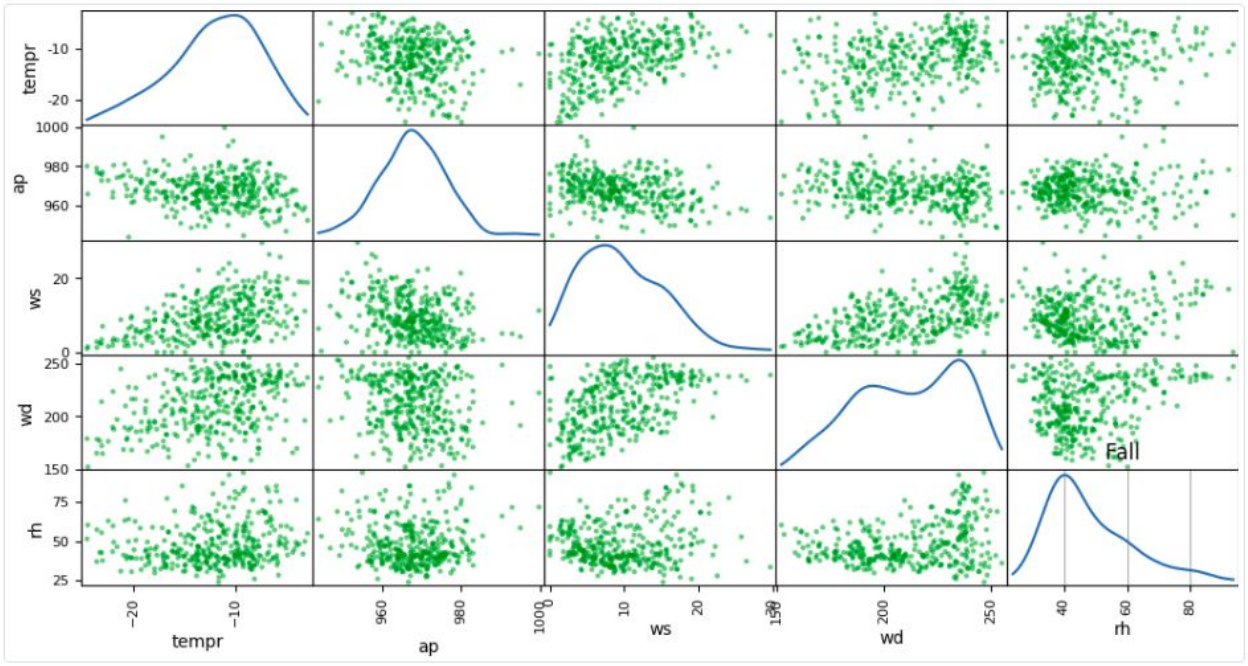
```
def find_correlation(table, sd, ed, frq):
   if table == 'surface_data':
       point = 'IMD'
       station = 'GANGOTRI (1985-1989)<br>MAITRI (1990-2016)'
   else:
       if table =='weatherdata' or table == 'climate_data':
           point = ''
           station = ''
       else:
           point, station = table.split('_')
           station = station.upper()
           point = point.upper()

   lst = ['obstime']

   length = 4
   t = 1
   start_date = sd
   end_date = ed

   global conn
   q = ','.join(lst)
   query = "SELECT * FROM " + table + " WHERE obstime BETWEEN '" +
start_date + "' AND '" + end_date + "'"
   lists = pd.read_sql(query, conn)

   lists['obstime'] = pd.to_datetime(lists['obstime'])
```

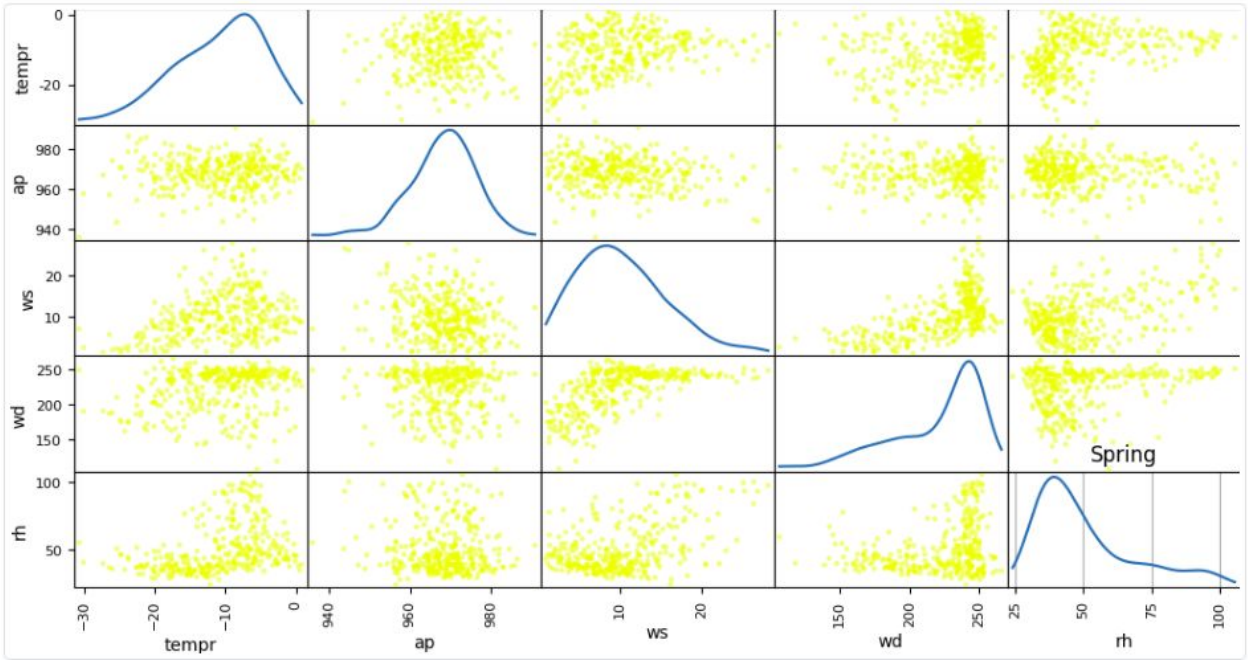
```
    lists = lists.set_index(lists['obstime'])

    info = 'No Data available for this duration - '

    lists = lists.resample(frq.upper()).mean()

    lists['month'] = lists.index.month
    lists.dropna(inplace=True)
    lists_sum=lists.query('month==12 or month==1 or month==2')
    lists_fall=lists.query('month==3 or month==4 or month==5')
    lists_wint=lists.query('month==6 or month==7 or month==8')
    lists_spring=lists.query('month==9 or month==10 or month==11')
    lists_sum=lists_sum.drop(['month'],axis=1)
    lists_fall=lists_fall.drop(['month'],axis=1)
    lists_wint=lists_wint.drop(['month'],axis=1)
    lists_spring=lists_spring.drop(['month'],axis=1)

    fig1 = plt.figure(figsize=(12, 6))
    ax1 = plt.subplot(5, 1, t, label='Summer')
    spm=scatter_matrix(lists_sum,alpha=0.5, diagonal='kde',c='red', ax=ax1)
    plt.title('Summer')
    plt.grid(True)
    plt.plot()
    fn1 = datetime.datetime.now().strftime("%Y_%m_%d_%H_%M_%S_%f") + str(t)
+'.png'
    fig1.savefig("static/" + fn1, bbox_inches='tight', pad_inches=0)
    plt.close(fig1)

    t += 1
    fig2 = plt.figure(figsize=(12, 6))
    ax2 = plt.subplot(5, 1, t, label='Fall')
    spm=scatter_matrix(lists_fall,alpha=0.5, diagonal='kde',c='green',
ax=ax2)
    plt.title('Fall')
    plt.grid(True)
    plt.plot()
    fn2 = datetime.datetime.now().strftime("%Y_%m_%d_%H_%M_%S_%f") + str(t)
+'.png'
    fig2.savefig("static/" + fn2, bbox_inches='tight', pad_inches=0)
    plt.close(fig2)

    t += 1
```

```
   fig3 = plt.figure(figsize=(12, 6))
   ax3 = plt.subplot(5, 1, t, label='Winter')
   spm=scatter_matrix(lists_wint,alpha=0.5, diagonal='kde',c='blue',
ax=ax3)
   plt.title('Winter')
   plt.grid(True)
   plt.plot()
   fn3 = datetime.datetime.now().strftime("%Y_%m_%d_%H_%M_%S_%f") + str(t)
+'.png'
   fig3.savefig("static/" + fn3, bbox_inches='tight', pad_inches=0)
   plt.close(fig3)

   t += 1
   fig4 = plt.figure(figsize=(12, 6))
   ax4 = plt.subplot(5, 1, t, label='Spring')
   spm=scatter_matrix(lists_spring,alpha=0.5, diagonal='kde',c='yellow',
ax=ax4)
   plt.title('Spring')
   plt.grid(True)
   plt.plot()
   fn4 = datetime.datetime.now().strftime("%Y_%m_%d_%H_%M_%S_%f") + str(t)
+'.png'
   fig4.savefig("static/" + fn4, bbox_inches='tight', pad_inches=0)
   plt.close(fig4)
```

Fig 31-Code Snippet

This code was put up on the NCPOR data portal and can be accessed by scientists across the world for research purposes.


tempr
ap
ws
wd
rh
Fall


tempr
ap
ws
wd
rh
Spring

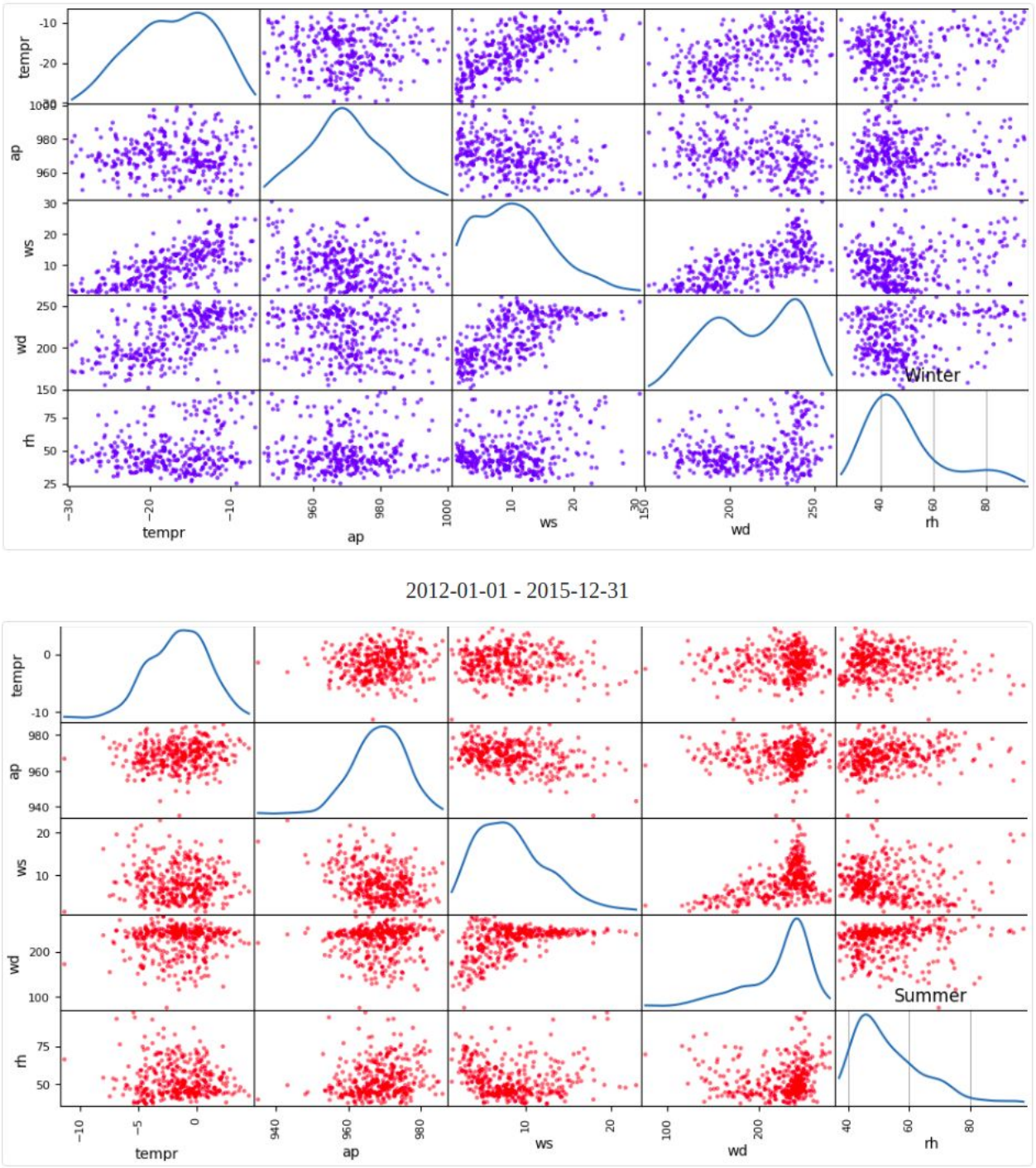


Fig 32 - Scatter Matrix of all four seasons

Data Forecasting:

RNN (in Python) - Recurrent Neural Network for Weather Prediction-

The goal of this part of the project was to write a program for weather forecasting in Antarctica using RNN and it will be used in the NCAOR website for future weather prediction. Recurrent Neural Network (RNN) is a type of Deep Learning Neural Network generally used for time series analysis. It is different from the feedforward neural network in the sense that it also considers the previous state for giving the output which regular feedforward neural network doesn't take into consideration.

We implemented Recurrent Neural Network (RNN) in python for the time series analysis and future prediction for Weather data given by NCAOR.The data had different sub-categories like temperature,humidity,pressure,wind speed,wind direction,rain,dew,etc. The data was predicted using univariate as well as multivariate time series analysis. The data was in CSV format which was extracted using the pandas library of Python. Used Keras library of Python for implementing RNN.The project was implemented using Anaconda distribution which had many pre-downloaded Python libraries.

CNN (in Python) - Convolutional Neural Network for Weather Prediction-

Another machine learning model used for weather forecasting was CNN(Convolution Neural Network). Generally CNN is used for image processing as it is good in finding patterns in the data. So we tried to use CNN for weather forecasting in order to find patterns in the weather parameters such as Temperature, Air Pressure, Wind Speed, Wind Direction and Relative Humidity..

We implemented CNN in python on the time series data given by NCAOR. Various data files were used to test the models iig_maitri, iig_bharati, imd_maitri, imd_bharati, surface_data, sankalp_sase. The data given had different parameters like Temperature, Pressure, Wind Speed, etc. The data was in CSV format and it was extracted as Dataframe using Pandas library of Python.

The CNN model was made using the Conv1D model of the Keras library of Python. The prediction was done using univariate as well as multivariate time series analysis. The program was implemented using Anaconda distribution which had many pre-downloaded python libraries.

This program was finally used for prediction of Temperature, Air Pressure, Relative Humidity, Wind Speed on the NCAOR website. Below I am attaching some snapshot of weather forecasting using CNN :

Finally, a function was used to integrate the code of RNN and CNN on the website of NCAOR. Firstly various libraries were imported that will be used in the code.

```
import numpy as np
import math
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from keras.layers import GRU, Dropout, SimpleRNN
from keras.optimizers import SGD
%matplotlib inline
sns.set()
plt.rcParams.update({'figure.figsize': (18, 8), 'figure.dpi': 100})
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

Fig 33- Code Snippet

The function has four inputs which are model name using which prediction was to be done, the dataset on which the prediction has to be done, the variable on which it has to be done and the duration of the prediction. In the beginning of the function, the datasets were read using pandas library and the parameter for which the prediction to be done was extracted and the non-available values(-999.99) were removed from the data.

```
def pred(m_name,data,var,dur):
      df = pd.read_csv(data+'.csv')
      df.obstime = pd.to_datetime(df.obstime)
      df.set_index('obstime', inplace=True)
      df=df[df[var]>=-900]
      fy=df.index.year[0]
      ly=df.index.year[df.index.size-1]
      avg=[]
      avg=np.array(avg)
      fp=[]
      fp=np.array(avg)
      count=0;
      fm=df.index.month[0]
      lm=13
      for j in range(df.index.year[0],df.index.year[df.index.size-1]+1):
        I   if(j>df.index.year[0]):
                  fm=1
            if(j==df.index.year[df.index.size-1]):
```

```
                lm=df.index.month[df.index.size-1]+1;
            for i in range(fm,lm):
                av=df[np.logical_and(df.index.year == j
,df.index.month==i)][var].mean()
                avg=np.append(avg,av)
                fp=np.append(fp,av)
```

Fig 34- Code Snippet

Then the data was normalised and converted into the supervised format to train the respective model.

```
    m=avg.mean()
    avg-=m
    fp-=m
    ma=np.abs(avg).max()
    avg/=ma
    fp/=ma
    def split_sequence(sequence, n_steps):
        X, y = list(), list()
        for i in range(len(sequence)):
            end_ix = i + n_steps
            if end_ix > len(sequence) - 1:
                break
            seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
            X.append(seq_x)
            y.append(seq_y)
        return np.array(X), np.array(y)


    raw_values =avg
    raw_values2=raw_values[:(avg.size-12)]
    testX=raw_values[(avg.size-12):]
    n_steps = 12
    X, y = split_sequence(raw_values, n_steps)
    n_features = 1
```

```
      X = X.reshape((X.shape[0], X.shape[1], n_features))
```

Fig 34- Code Snippet

After that the models were specified depending on the model given as input.

```
      if(m_name=='CNN'):
         model = Sequential()
         model.add(Conv1D(filters=64, kernel_size=2, activation='relu',
input_shape=(n_steps, n_features)))
         model.add(MaxPooling1D(pool_size=2))
         model.add(Flatten())
         model m=avg.mean()
         model.compile(optimizer='adam', loss='mse')

      if(m_name=='RNN'):
         model = Sequential()
         model.add(SimpleRNN(50, input_shape=(n_steps, n_features),
return_sequences=False,activation='tanh'))
         model.set_weights([np.random.rand(*w.shape)*0.2 - 0.1 for w in
model.get_weights()])
         model.add(Dense(1))
         model.compile(optimizer='adam', loss='mse')
```

Fig 35- Code Snippet

The model was trained till a certain loss was reached and the prediction was done for the specified duration.

```
         i=0
         while((i==0 or hist.history['loss'][0]>=0.0056  or i<80 ) and i
<1000 ):
             hist=model.fit(X, y, epochs=1,verbose=0)
             i=i+1
```

```
temp_values = raw_values
temp_values2=fp
pred=[]
pred=np.array(pred)
temp_values_e=raw_values2
pred_e=[]
pred_e=np.array(pred_e)
j=0
s=0
for i in range(dur):
    x_input = temp_values[-1*n_steps:]
    x_input = x_input.reshape((1, n_steps, n_features))
    yhat = model.predict(x_input, verbose=0)
    pred=np.append(pred,yhat[0])
    temp_values = np.append(temp_values, yhat)
    temp_values2=np.append(temp_values2,yhat)
    x_input_e = temp_values_e[-1*n_steps:]
    x_input_e = x_input_e.reshape((1, n_steps, n_features))
    yhat_e = model.predict(x_input_e, verbose=0)

    pred_e=np.append(pred_e,yhat_e)
    temp_values_e=np.append(temp_values_e,yhat_e[0])
```

Fig 36- Code Snippet

Finally the prediction was shown on a graph using matplotlib library of python.

```
import datetime as dt

dates = []
for year in range(int(fy), int(ly)+1024):
    if(year==int(fy)):
        fm=int(df.index.month[0])
    else:
        fm=1
    lm=13
    for month in range(fm, lm):
        dates.append(dt.datetime(year=year, month=month, day=1))
```

```
        plt.plot(dates[0:temp_values2.size], (temp_values2+m))
        plt.plot(dates[0:fp.size], (fp+m))
        plt.show()
```

Fig 37- Code Snippet

Attached below are some snapshots of the prediction using RNN and CNN :

Fig 38 - Relative Humidity Prediction(CNN) of 3 years on sankalp_sase dataset

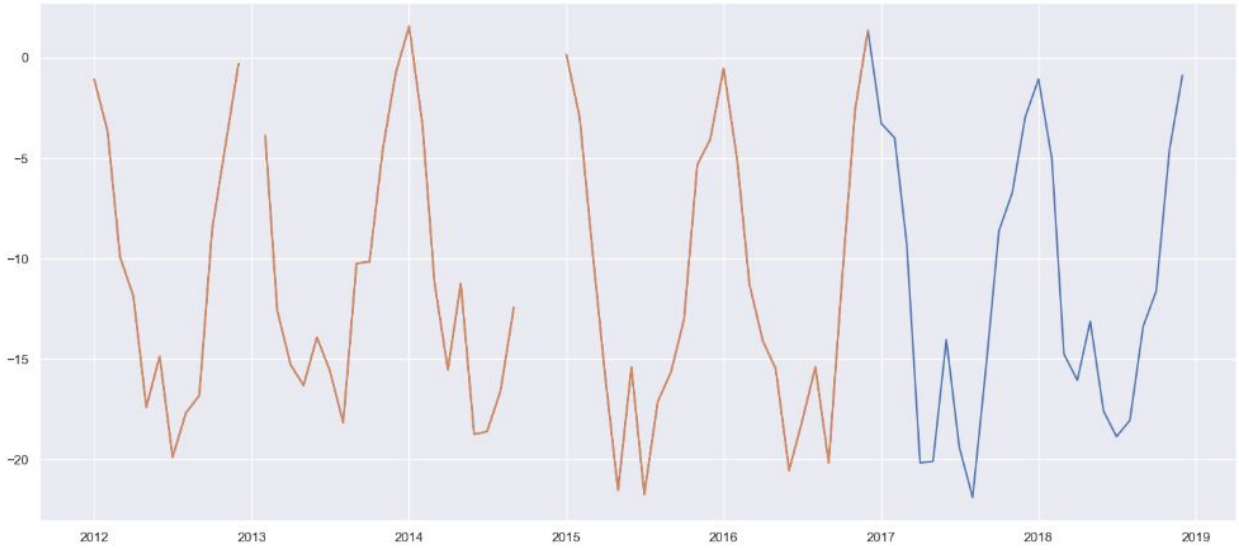


Fig 39 - Temperature Prediction(CNN) of 2 years on the iig_bharati dataset

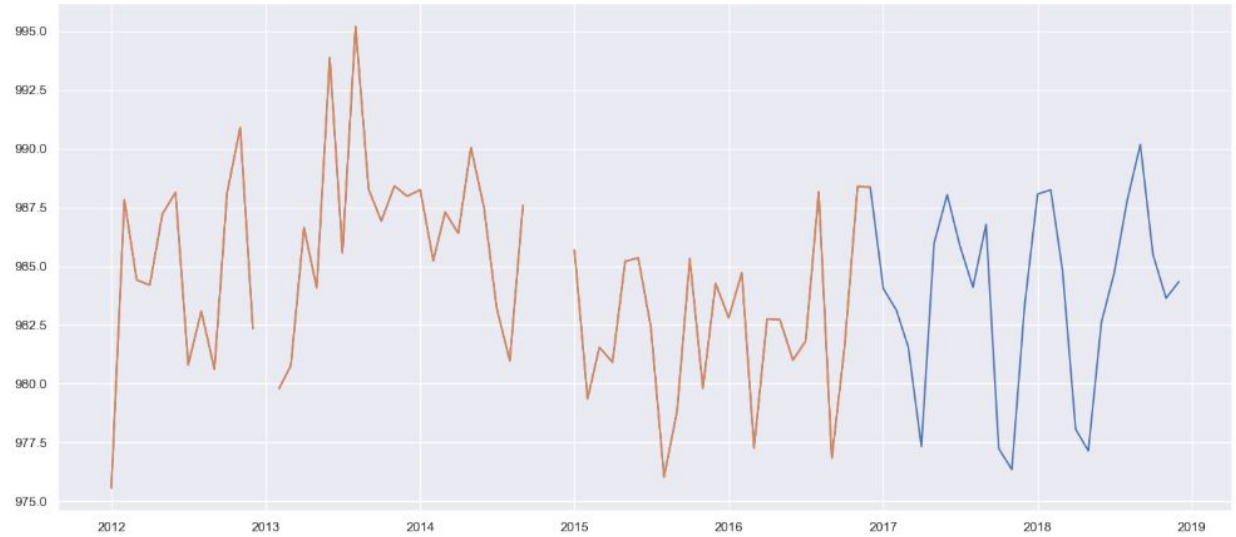


Fig 40 - Air Pressure Prediction (CNN) of 2 years on the iig_bharati dataset

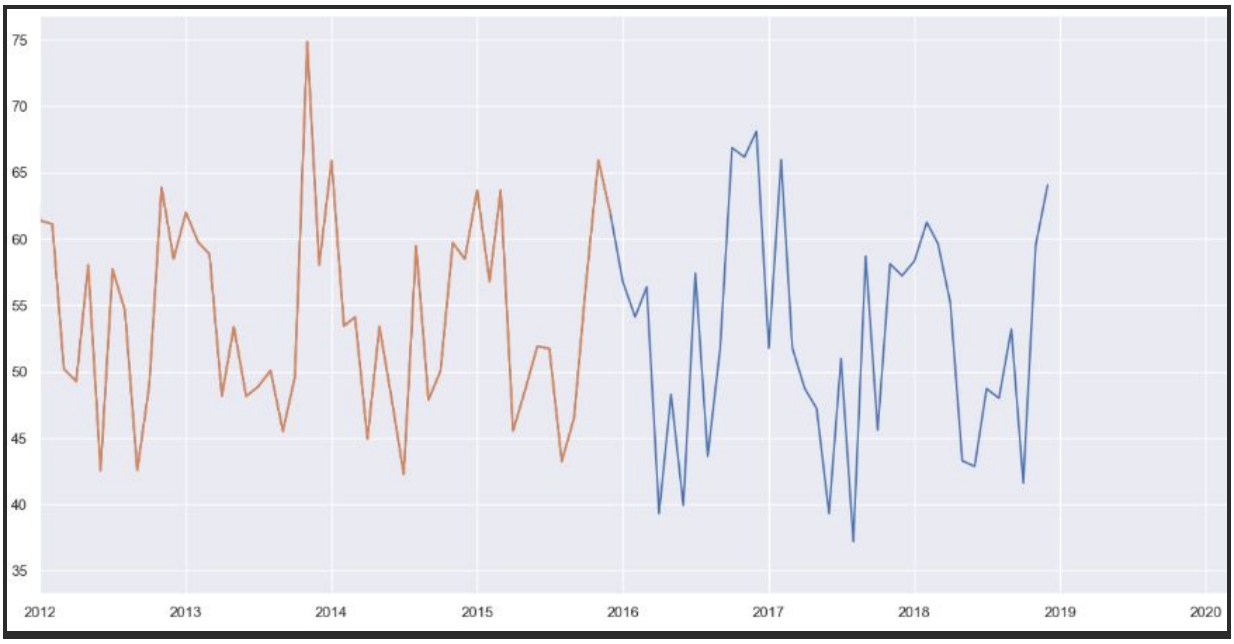


Fig 40 - Relative Humidity Prediction (RNN) of 3 years on Sankalp_Sase dataset

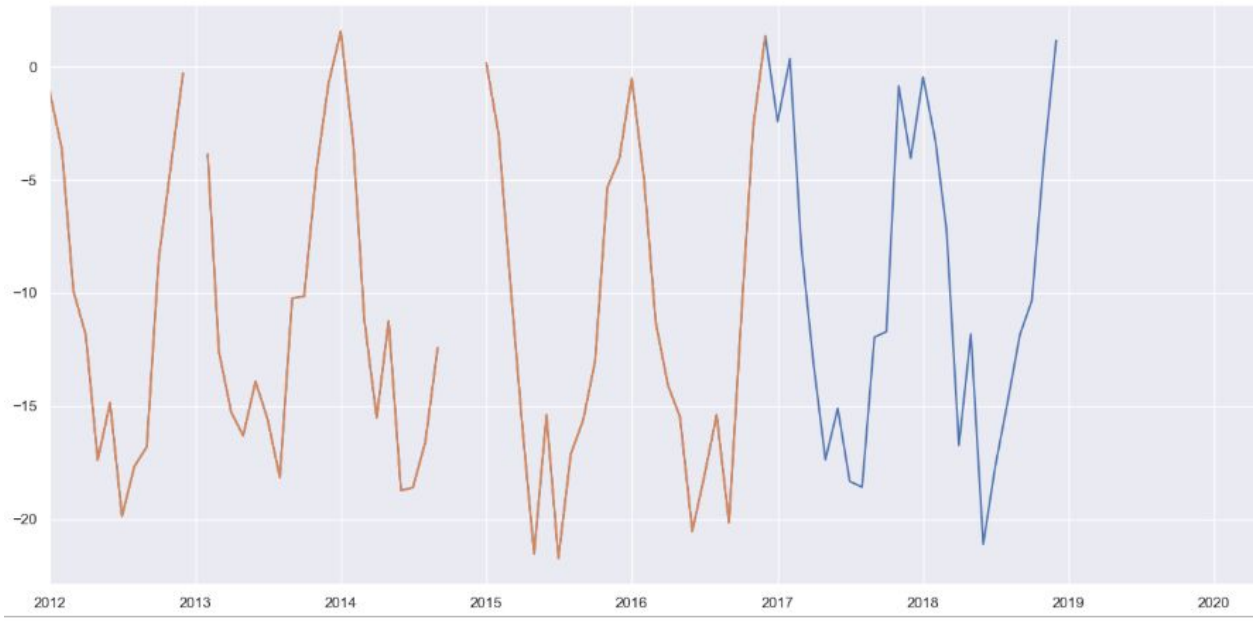


Fig 41 - Temperature Prediction (RNN) of 2 years on the iig_bharati dataset

Fig 42 - Wind Speed Prediction (RNN) of 2 years on the surface_data dataset

ARIMA Model (Auto Regression - Integration - Moving Average)- Univariate Time Series Prediction:

ARIMA (AutoRegressive Integrated Moving Average) is a time series model used to predict future points and to understand the data. It is mainly applied in statistics and econometrics. In this project, it is used as a tool to predict the weather conditions of a specified place with the help of historical data recorded by NCPOR. ARIMA is a generalisation of ARMA(AutoRegressive Moving Average) that is used to forecast for non-stationary data i.e. used to forecast for time series data that contains trend and seasonality.

The ARIMA model has 3 different parameters p, d and q.

- The parameter p is part of the AutoRegressive model which is determined by plotting an AutoCorrelation function. In an AR(p) model the future value of a variable is assumed to be a linear combination of p past observations and a random error together with a constant term.
- The parameter d is the Integration parameter that makes non-stationary data stationary by applying differencing.
- The parameter q is part of the Moving Average model which is determined by plotting a Partial AutoCorrelation function.MA(q) model uses past errors as the explanatory variables to predict the future outcome.

AR and MA model are effectively combined to form ARMA which is further generalised to accommodate non-stationary data forming ARIMA.

In this project data of the following form is taken from a .csv file using python pandas library.

datetime	condensation	dew	fog	hail	ht_index	hum	prec	pressure	rain	snow	temperature	thunder	tornado	vis	wspeed	wdir
1996-11-01 11:00:00	Smoke	9.0	0	0	NaN	27.0	NaN	1010.0	0	0	30.0	0	0	5.0	280.0	West
1996-11-01 12:00:00	Smoke	10.0	0	0	NaN	32.0	NaN	-9999.0	0	0	28.0	0	0	NaN	0.0	North
1996-11-01 13:00:00	Smoke	11.0	0	0	NaN	44.0	NaN	-9999.0	0	0	24.0	0	0	NaN	0.0	North
1996-11-01 14:00:00	Smoke	10.0	0	0	NaN	41.0	NaN	1010.0	0	0	24.0	0	0	2.0	0.0	North
1996-11-01 16:00:00	Smoke	11.0	0	0	NaN	47.0	NaN	1011.0	0	0	23.0	0	0	1.2	0.0	North

Fig 43 - Data read using the Pandas library for prediction using ARIMA

Then it is processed and corrected for the ARIMA model. Also, outliers are removed.

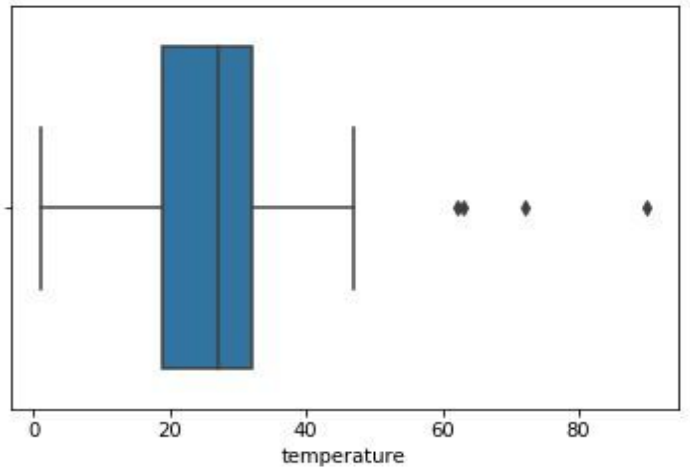


Fig 44 - Visualising the outliers in the temperature data.

After removal, an analysis of the PACF, ACF and data is done to decide the p,d and q parameters.
Techniques like taking a logarithm are used to make the data stationary and its stationarity is checked using the Dickey-Fuller Test. After deciding on the p,d and q parameters ARIMA is applied to forecast the future and also to compare the predictions with the actual data to check the efficiency of the data.

A python library which automates this whole process is *pmdarima*. It runs a grid search on all of the p, d and q parameters to find the best fit.

```
import numpy as np
import pmdarima as pm
from pmdarima import pipeline, preprocessing as ppc, arima
from matplotlib import pyplot as plt
import math
import datetime as dt
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

import pandas as pd
import datetime
get_ipython().run_line_magic('matplotlib', 'inline')
from scipy import stats

def Arima(df, var, months):
#     if dataset=='iig_maitri' or dataset=='iig_bharati':
#         data2=pd.read_csv('datasets/'+dataset+'.csv')
#         # print(df)
#     elif dataset=='dcwis':
#         data2=pd.read_csv('datasets/'+dataset+'.csv', names=['obstime',
'tempr', 'ap', 'ws', 'rh', 'dew'])
    df['obstime']=pd.to_datetime(df['obstime'])
    df=df.set_index('obstime')
    df=df.resample('M').mean()
    data2=df[var]
    # print(data2)
    # train2, test2= data2[:23423], data2[23423:]
    # train =train2.resample('M').mean()
#     ds_temp=df[var].resample('M').mean()
    if var=='rh':
        data2=data2[data2>10]
    if var=='ws':
        data2=data2[data2>=0]
    if var=='ap':
        data2=data2[data2>-10]
    data =data2.resample('M').mean()
    #     datum=data
    data.dropna(inplace=True)
    # print(data.size)
    ind=list()
    for i in range (int(data.size)):
```

```
        ind.append(i)
    # print(data.Date)
    # print(ind)
#     Q1 = data.quantile(0.25)
#     Q3 = data.quantile(0.75)
#     IQR = Q3 - Q1
#     # print(IQR)
#     data = data[~((data < (Q1 - 1.5 * IQR)) |(data > (Q3 + 1.5 * IQR)))]
    pipe = pipeline.Pipeline([
        ("fourier", ppc.FourierFeaturizer(m=12)),
        ("arima", arima.AutoARIMA(stepwise=True, trace=1,
error_action="ignore",
                                  seasonal=False,  # because we use Fourier
                                  transparams=False,
                                  suppress_warnings=True))
    ])

    pipe.fit(data)
```

Fig 45 - Code to fit the ARIMA model to a dataset through a Pipeline

The model is then used to make the required predictions.

```
df = df.asfreq('m')

    dates=(pd.to_datetime(df.index.values))
#     dates=df.index
#     print(dates)
    dates.freq='m'
    for i in range (months+1):
        dates = dates.union([dates[-1] + 1])
    preds, conf_int = pipe.predict(n_periods=months, return_conf_int=True)
    datum=data
    data3=data2.resample('M').mean()
    # print(dates)
    # print(data3[dt()'2012-01-31'])
    # for i, j in zip(data3.index, range(data3.size)):
    #     if math.isnan(data3[i])==True:
    #         del dates[j]
    # print(datum)
    temp=np.append(data3,preds)
```

```
    # dates=np.array(dates)
    # print(np.array(dates).shape)
    # # print
    # print(temp.shape)
    # plt.subplot(211)
    plt.plot(dates[:temp.size], temp)
    plt.plot(datum)
    # print(temp)
    # plt.subplot(212)
```

Fig 45 - Code Snippet

Following are some predictions using ARIMA:

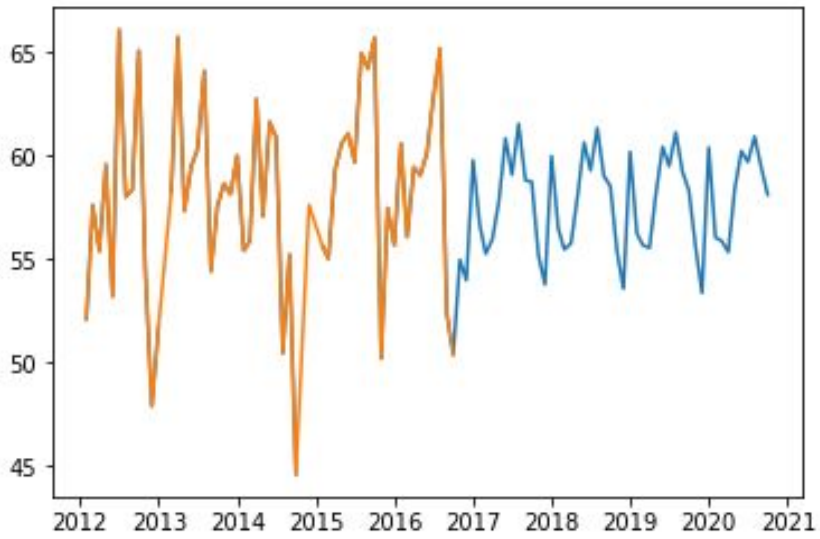


Fig 46 - Prediction of Relative Humidity using the iig_bharati dataset for the next 48 months.

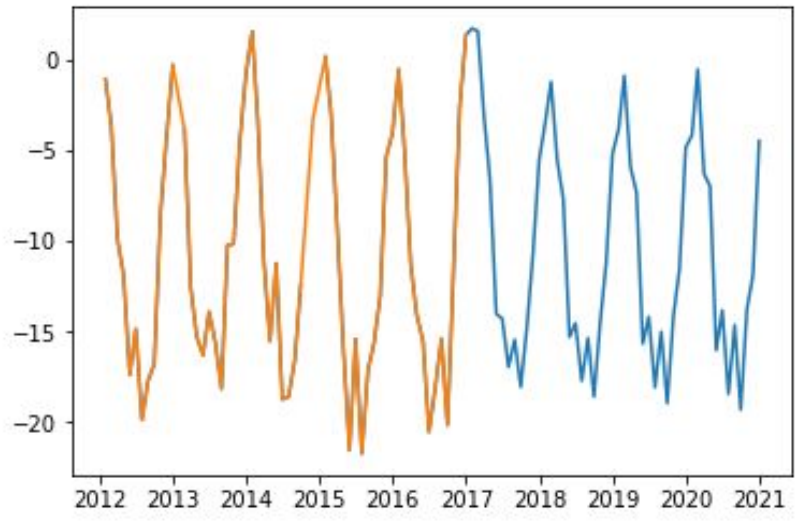


Fig 47 - Prediction of Temperature using the iig_bharati dataset for the next 48 months

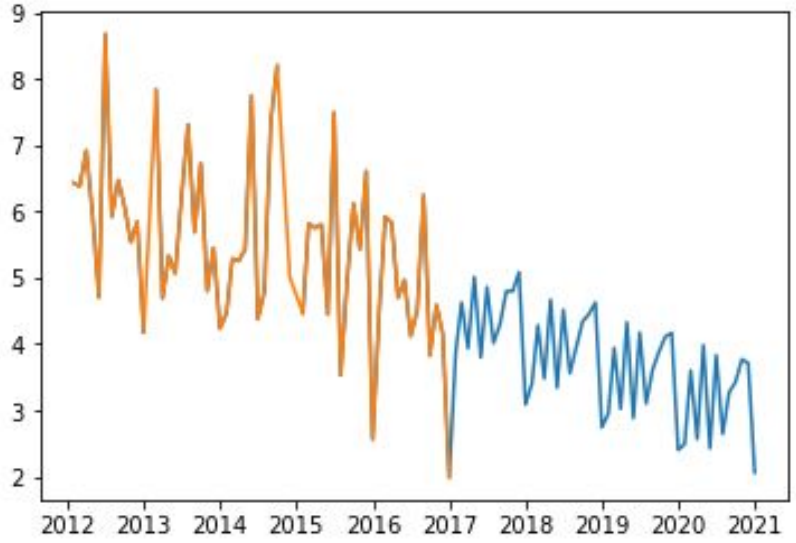


Fig 48 - Prediction of Wind Speed using the iig_bharati dataset for the next 48 months.

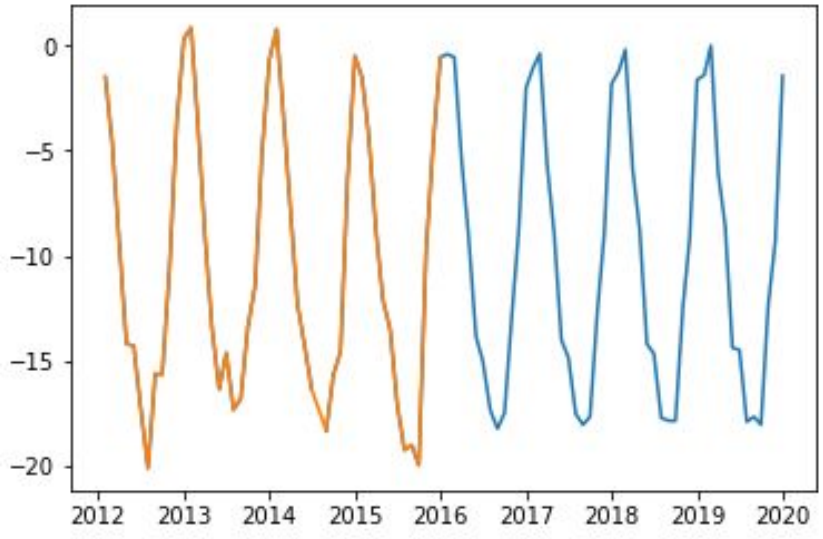


Fig 49 - Prediction of Temperature using the iig_maitri dataset for the next 48 months.

LSTM (in Python) - Long Short Term Memory for weather prediction:

The Model:

LSTM can be thought of as an improvement over RNN Models. Two common problems associated with deep RNN models are: 1. The problem of exploding gradients and 2. The problem of vanishing gradients. Both of these are beyond the purview of this report. The LSTM model deals with these problems by the usage of Gates. Refer to the following diagram for a more detailed explanation.

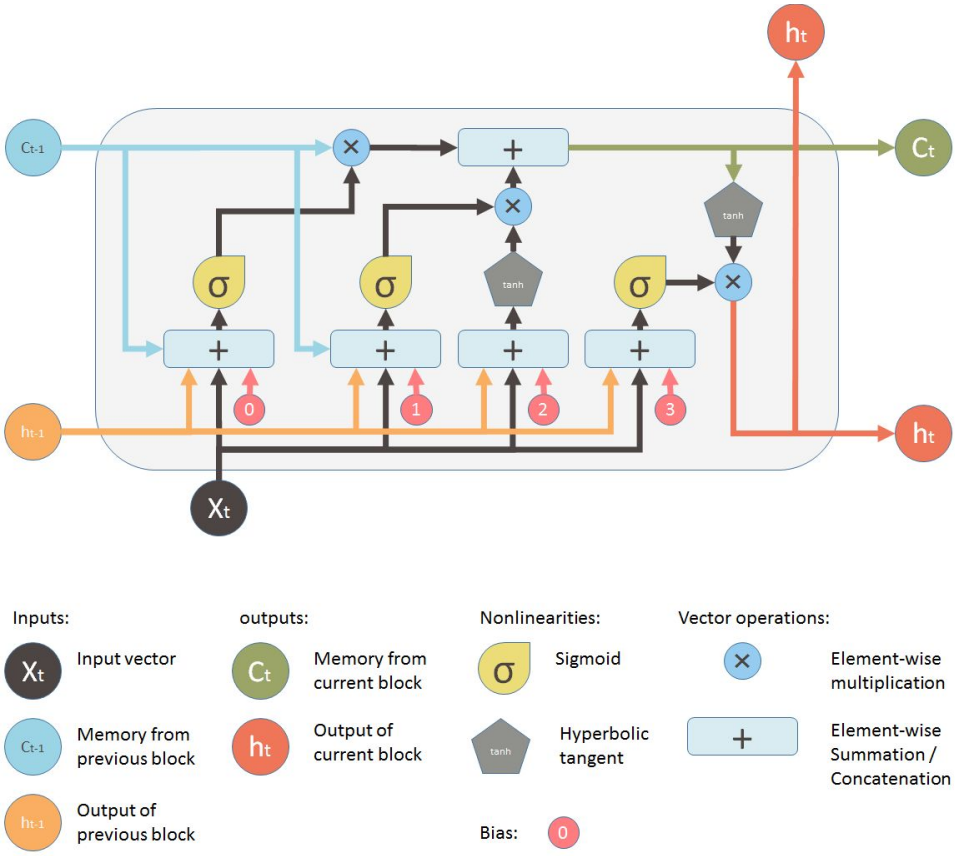


Fig 50 - A Hidden Layer in an LSTM Model (Simplified)

Implementation:

The Keras model is used to implement the LSTM Model.

Firstly, the required libraries are imported.

```
import keras
import pandas as pd
# import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
# import sklearn as sk
# from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import mean_squared_error
from dateutil.parser import parse
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import math
import datetime as dt
from dateutil.relativedelta import *
```

Fig 51- Code Snippet

The learning model is then constructed and then trained using the available data.

```
model=Sequential()
model.add(LSTM(100,input_shape=(12, 1),))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
ds2=np.reshape(ds2, (ds2.shape[0], 1, 1))
i=0
while i==0 or hist.history['loss'][0]>0.02:
    i+=1
    print('Epoch: ',i)
    hist=model.fit(X, Y, epochs=1, verbose=1, batch_size=1)
```

Fig 52- Code Snippet

The trained model is then used to make the forecast.

```
ds3=ds
temp_values=ds2
preds=ds3
for i in range(months_):
    x_input=temp_values[-12:]
    x_input=x_input.reshape(1,12,1)
```

```
    print(i,' out of ',months_-1,':  ')
    yhat=model.predict(x_input, verbose=1)
    temp_values=np.append(temp_values,yhat[0])
    preds=np.append(preds,yhat[0])
plt.rcParams.update({'figure.figsize':(18,8),'figure.dpi':100})
# plt.subplot(211)
preds=preds*maxi
preds=preds+mean
ds3=ds3*maxi
ds3=ds3+mean
```

Fig 53 - Code for a Dynamic LSTM Model.

LSTM predictions are more accurate than their RNN and CNN counterparts if the data is sufficiently large. Some predictions made by the LSTM model are shown here.

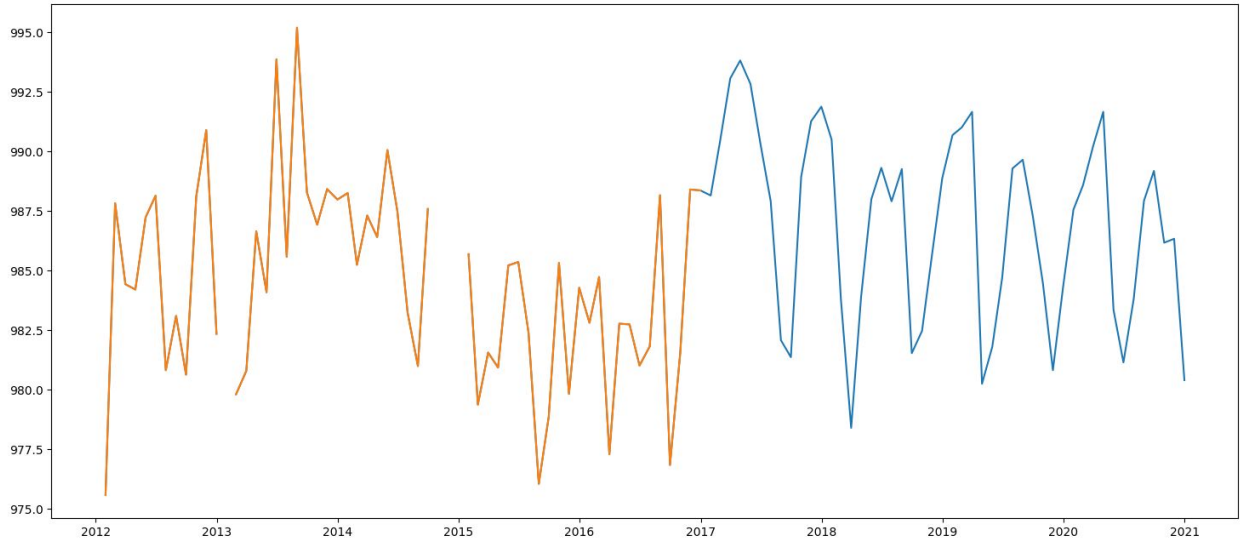


Fig 54 - Prediction of Air Pressure using the iig_bharati dataset for the next 48 months.

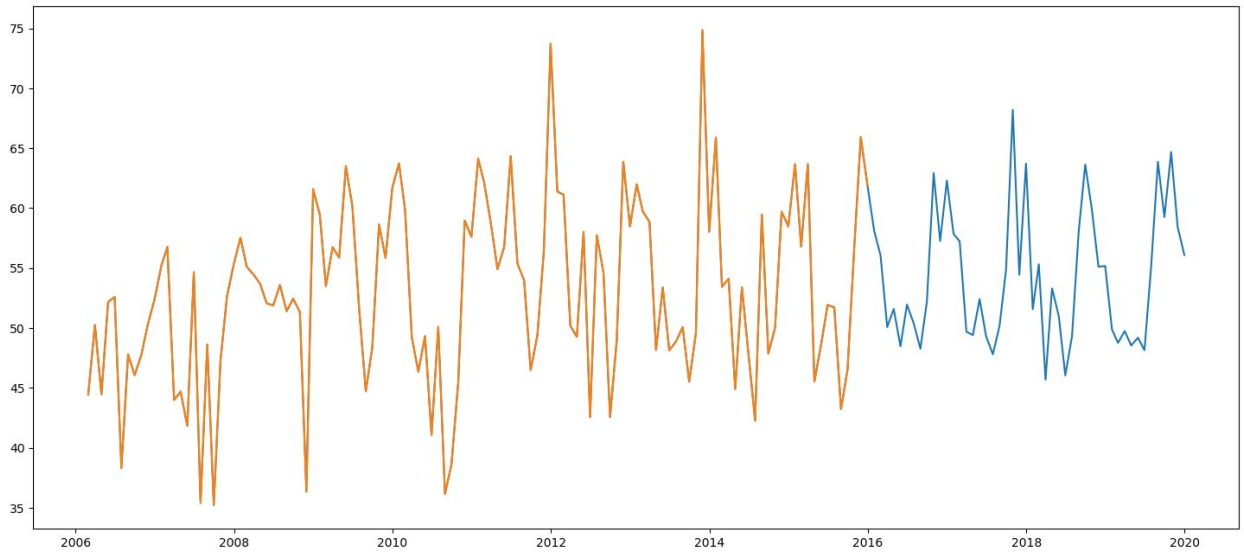


Fig 55 - Prediction of Relative Humidity using the sankalp_sase dataset for the next 48 months.

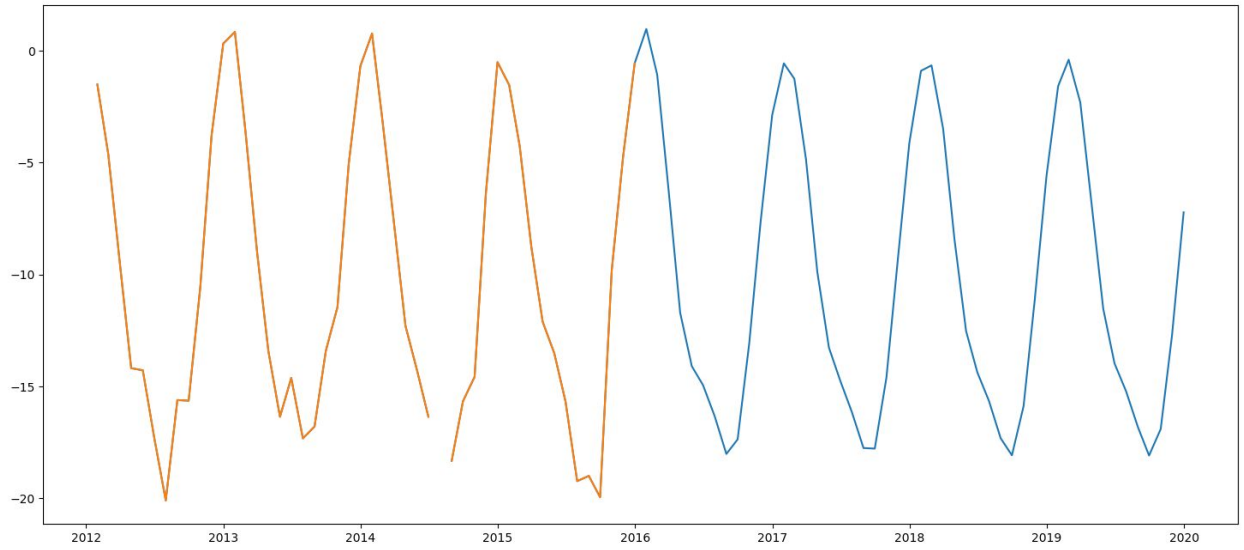


Fig 56 - Prediction of Temperature using the iig_maitri dataset for the next 48 months.

Data Analysis in R:

As part of the PS project, we were required to make a web app, data analysis. The major requirements of the webapp were that it must be quick and convenient to use and that it must respond to a variety of data. To achieve these aims the webapp is made as an interactive GUI and

the functions used are generalized functions which compute values dynamically according to the data requested by the user. To make the app fast, functions are made reactive which prevents the loading of the complete dataset each time a new subset interval is chosen.

A great deal of time was spent for data preprocessing, to bring various data formats under a standardized one to incorporate generality into the code. An important aspect of this was the handling of date formats. The following function converts all the formats under tryFormats to a POSIX format of dates.

The following is the most widely used function. It filters out the requested dates from the dataset and removes the outliers. The output of this function is used as an intermediate input to many other functions.

The app dynamically loads the first and last date ranges and also the parameters of the dataset. These parameters can then be selected by the user as required.

As part of the forecasts, 4 techniques, namely ARIMA,RNN,CNN, and LSTM are used, details of which are provided above. The number of units to forecast in the future is kept dynamic as a numeric input by the user. The forecasted values are shown as by a red line overlapping a blue one which represents the actual data values. To compare between different models, Mean Error is chosen as a metric for the accuracy of the forecasted values.

To plot the data, data is first converted to a time series. The periodicity of time-series is calculated by calculating difference in time period of two consecutive dates.

A feature to plot the correlation matrix, expressing the correlation between the parameters is provided. A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used as a way to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses. The matrix helps analyze large amounts of data by quickly identifying patterns.

- The distribution of each variable is shown on the diagonal.
- On the bottom of the diagonal : the bivariate scatter plots with a fitted line are displayed
- On the top of the diagonal : the value of the correlation plus the significance level as stars
- Each significance level is associated to a symbol : p-values(0, 0.001, 0.01, 0.05, 0.1, 1) <=> symbols(“***”, “**”, “*”, “.”, " “)

The aforementioned models are static statistical models. There is a feature for dynamic forecasting provided as well. Dynamic forecast uses the value of the previous forecasted value of the dependent variable to compute the next one, while static forecast uses the actual value for each subsequent forecast.

To compare between different parameters over a specific time period rather than over the entire dataset an option is provided to plot them simultaneously. The values are first normalized over the time period requested and plotted side by side, clearly showing the trends.

Often, parameter values are required to be plotted season wise for weather analysis. Therefore, the season action button is provided which provides a season by season plot of the requested parameter.

A blizzard is an anomaly whose detection is imperative during expeditions to the polar regions. To filter out anomalies, two conditions are imposed; namely, wind speed should be greater than 23 knotts and relative humidity should be higher than 75%. Prolonged periods(3 hours or more) are taken to be a blizzard. These blizzards are represented as red dots in the ggplot.

Blizzard Prediction:

A blizzard is a severe snowstorm characterised by high wind speeds and long durations. Blizzards are a common occurrence in the Antarctic. Blizzards are characterized by high wind speeds. Wind rose graphs were plotted to understand the correlation between wind speed and other attributes.

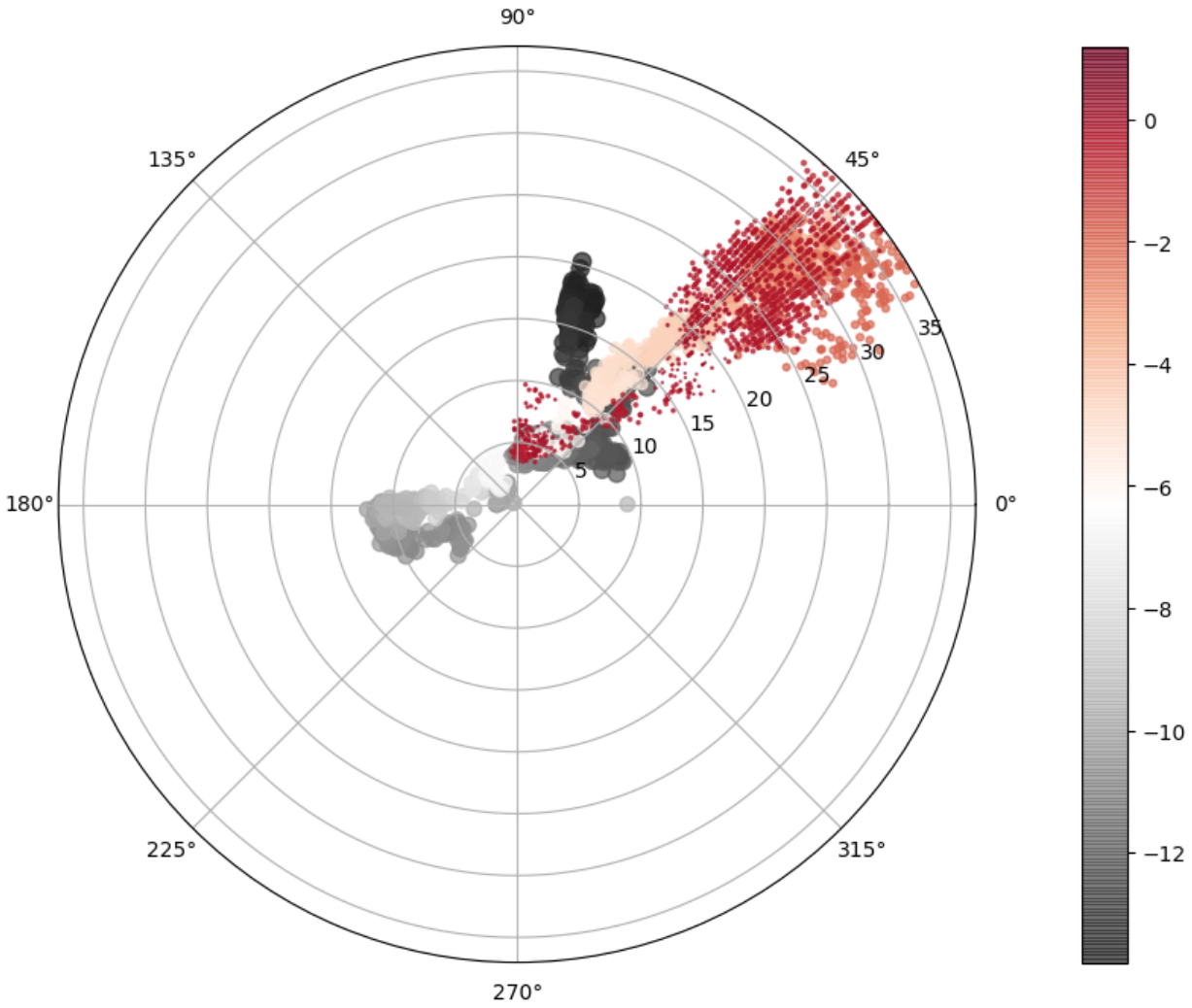


Fig 57 - Wind rose graph between wind speed, wind direction, temperature

In the above figure, the distance of a point from the centre is determined by the wind speed. The colour intensity of a point is determined by the temperature at that point.

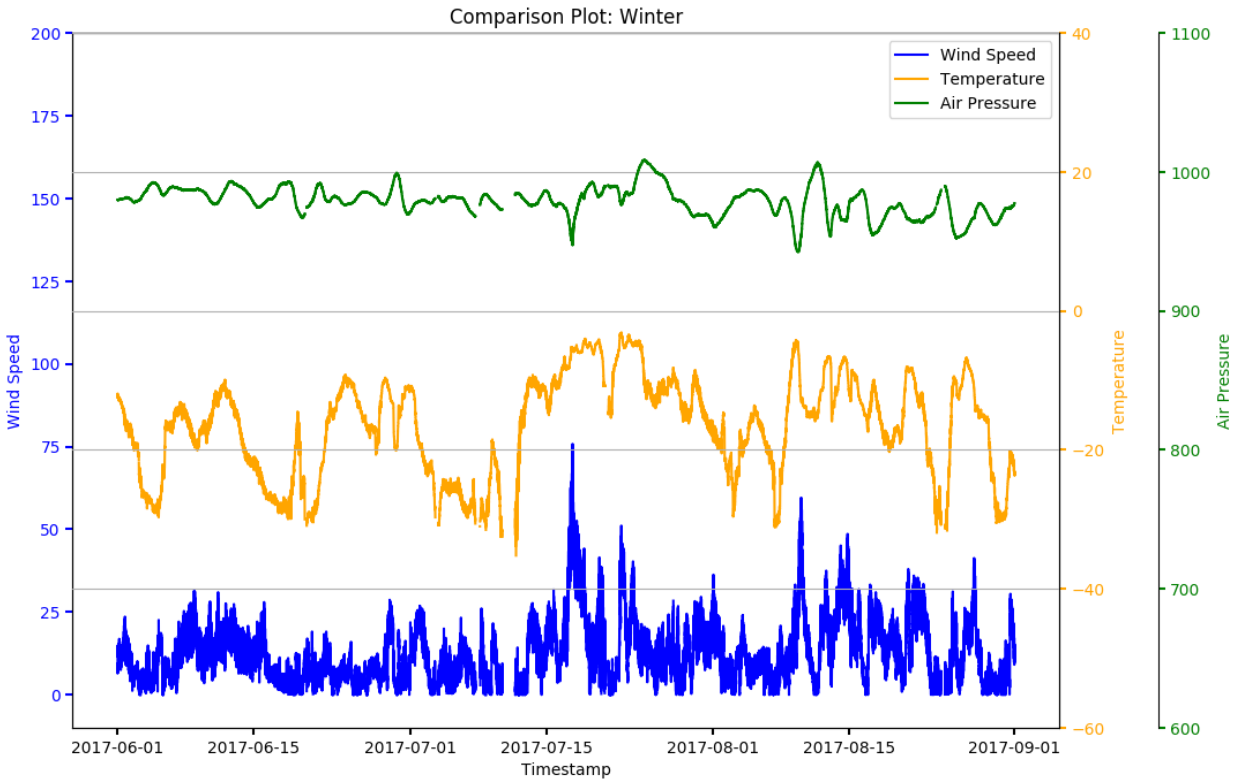


Fig 58 - Comparison plot between wind speed, temperature and pressure

From the above figure, it can be observed that there is no significant correlation between wind speed and temperature. Although, there is an inverse relationship between wind speed and air pressure. A low air pressure zone is created whenever the wind speed goes high.

A statistical model was created with the help of different characteristics of blizzards. A few characteristics which played a crucial role in the statistical model include wind speeds above 23 knots for a duration of at least 3 hours along with a relative humidity above 75% and a low air pressure. The model went through the dataset which contained hourly data and used the above conditions to help determine the date and time at which a blizzard occurred.

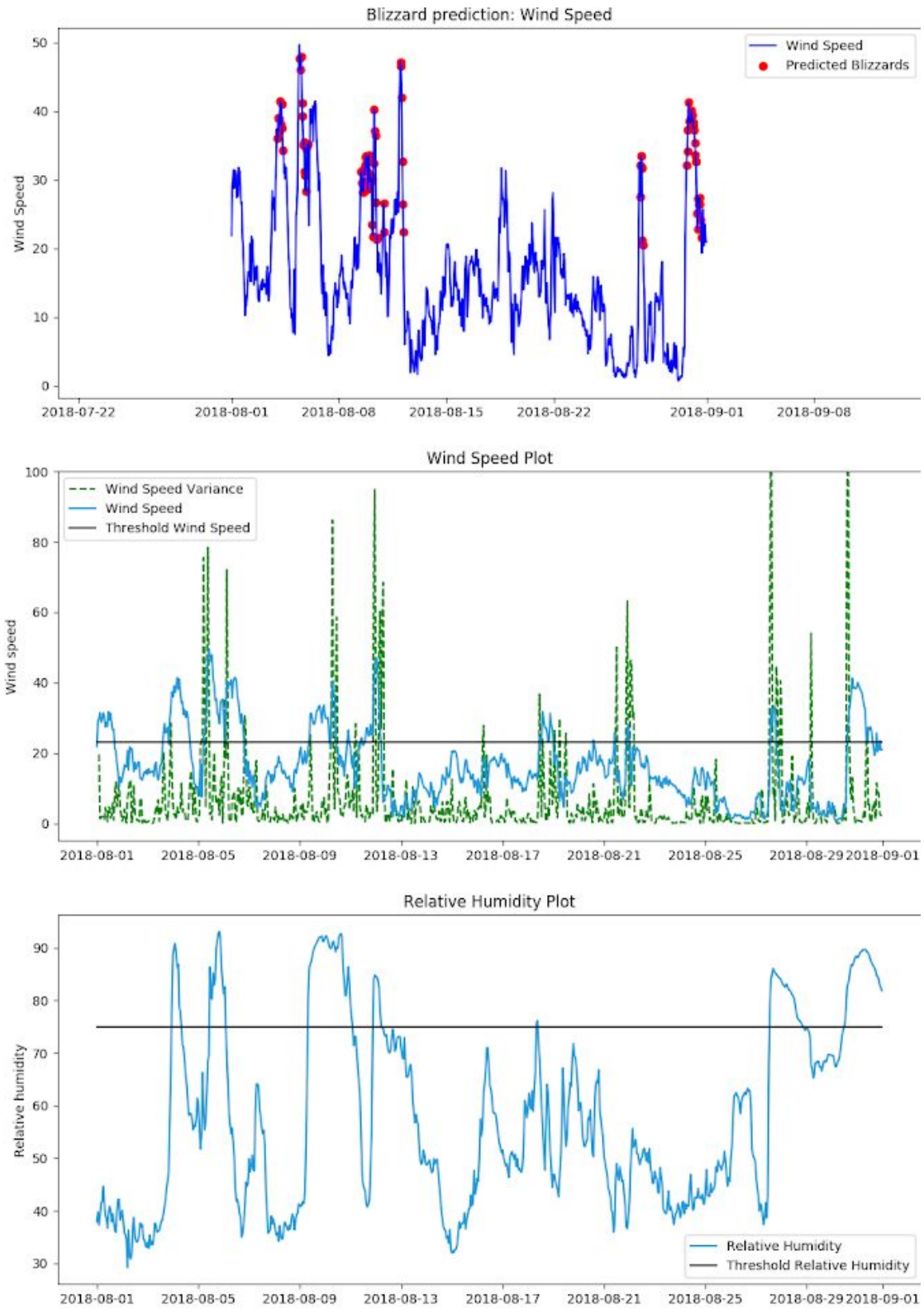


Fig 59 - Plot of predicted blizzards

Results/Conclusions:

The following conclusions can be drawn from the work done at NCPOR:

1. The National Centre for Polar and Ocean Research works for the benefit of the environment. It finds the reasons for the melting and pollution of polar ice caps and glaciers and tackles them at their roots.
2. The current project involves analysis of weather data collected by Indian Weather Stations at the coldest places around the Earth.
3. The various departments at NCPOR provide this data in a readable format.
4. The project involves implementing models like ARIMA, RNN, CNN and LSTM for the prediction of weather.
5. The prediction of weather data helps scientists to plan their exploration to some of the most dangerous places on Earth.
6. Python and R are two programming languages which are very rich in libraries, especially for data analysis.
7. The models for the prediction of time-series data are being improved by the second.
8. A computer system with high-end specifications is needed to handle the actual data, which is very large in size.
9. Time series prediction has great usage in various fields such as weather prediction, stock market analysis etc.

The 3 anomaly detection algorithms were run on the same dataset and their results are compiled below. The data after removing the outliers was fit to a normal distribution.

Table 3- Anomaly detection algorithms comparison

Algorithm	Time Taken	Mean	Standard Deviation
K-Means	60.32 seconds	37.49	44.55
One Class SVM	0.76 seconds	38.28	47.08

Isolation Forest	1.73 seconds	37.52	44.77

In the blizzard prediction model, rolling variance for wind speed remains constant and low after observing a large spike during the time of blizzard. This coupled with high relative humidity and low air pressure zone gives an accurate prediction for blizzards.
On matching the predictions with the actual weather report for those days, the statistical model achieved 90% accuracy.

Future Scope:

The current server prevents multiple users accessing at the same time. The backend of the web server can be configured to handle multiple users accessing together.
The current machine learning algorithms are applied on univariate data. We can achieve better results by using multivariate data models.
Due to lack of resources, current models predict results on a monthly scale. In the future, models can be optimized to forecast minute-by-minute data.

Appendices:

Appendix A: Anomaly Detection Algorithms

1. K-means-

```
df = featureSelection(processed_list, col)
            data = df[[col, 'month', 'day', 'season']]

            min_max_scaler = preprocessing.StandardScaler()
            np_scaled = min_max_scaler.fit_transform(data)
            data = pd.DataFrame(np_scaled)
            pca = PCA(n_components=2)
            data = pca.fit_transform(data)

            min_max_scaler = preprocessing.StandardScaler()
            np_scaler = min_max_scaler.fit_transform(data)
            data = pd.DataFrame(np_scaled)

            n_cluster = range(1,20)
            kmeans = [KMeans(n_clusters=i).fit(data) for i in n_cluster]
            scores = [kmeans[i].score(data) for i in range(len(kmeans))]

            n_cluster = 10
            df['cluster'] = kmeans[n_cluster].predict(data)
            df['pf1'] = np.array(data[0])
            df['pf2'] = np.array(data[1])
            df['cluster'].value_counts()

            distance = getDistanceByPoint(data, kmeans[n_cluster])
            number_of_outliers = int(outliers_fraction*len(distance))
            threshold = distance.nlargest(number_of_outliers).min()
            df['anomaly'] = np.array((distance >= threshold).astype(int))
```

2. Isolation Forest-

```
df = featureSelection(processed_list, col)
            data = df[[col, 'month', 'day', 'season']]
            # print(data.head())

            min_max_scaler = preprocessing.StandardScaler()
            np_scaled = min_max_scaler.fit_transform(data)
            data = pd.DataFrame(np_scaled)

            model =  IsolationForest(contamination = outliers_fraction)
            model.fit(data)

            df['anomaly'] = np.array(model.predict(data))
            df['anomaly'] = df['anomaly'].map( {1: 0, -1: 1} )

            # df[col] = df[col] * result_add.seasonal
            t = anomaly_plot(df, col, length, t, sd, ed, yearly_plot)
```

3. One Class SVM-

```
df = featureSelection(processed_list, col)
            data = df[[col, 'month', 'day', 'season']]
            # print(data.head())
            min_max_scaler = preprocessing.StandardScaler()
            np_scaled = min_max_scaler.fit_transform(data)
            # train one class SVM
            model =  OneClassSVM(nu=0.95 * outliers_fraction) #nu=0.95 *
outliers_fraction  + 0.05
            data = pd.DataFrame(np_scaled)
            model.fit(data)
            # add the data to the main
            df['anomaly'] = np.array(model.predict(data))
            df['anomaly'] = df['anomaly'].map( {1: 0, -1: 1} )

            # df[col] = df[col] * result_add.seasonal
            # print(df.head())
            t = anomaly_plot(df, col, length, t, sd, ed, yearly_plot)
```

4. FFT-

```
import pandas as pd
import sys
from datetime import datetime
import numpy as np
def find_outliers(df, cols):
      df_temp = df
      df_outliers = df
      outliers_dict = {}
      for col in cols:
      data = df_temp[col].tolist()
      #     print(data)
      q25, q75 = np.nanpercentile(data, 25), np.nanpercentile(data, 75)
      iqr = q75 - q25
      print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75,
iqr))
      # calculate the outlier cutoff
      cut_off = iqr * 1.5
      lower, upper = q25 - cut_off, q75 + cut_off
      # identify outliers
      outliers = [x for x in data if x < lower or x > upper]
      outliers_dict[col]  = outliers
      print('Identified outliers: %d' % len(outliers))
      # remove outliers
      outliers_removed = [x for x in data if x >= lower and x <= upper]
      print('Non-outlier observations: %d' % len(outliers_removed))
      df_temp = df_temp[~df_temp[col].isin(outliers)]
      # df_outliers = df_outliers[df[col].isin(outliers)]
      return df_temp
df=pd.read_csv('dcwis2.csv')
# df['obstime']=pd.to_datetime(df['obstime'],format='%d-%m-%Y %H:%M')
df['obstime'] = [datetime.strptime(x, '%m-%d-%Y %H:%M') for x in
df['obstime']]
df.replace(-999, np.nan)
df=df.set_index('obstime')
#df = df.resample('H').mean()
# df = df.dropna()
df=df.resample('H').mean()
```

```
#df1=find_outliers(df,df.columns.values)
# df1.head()


df.to_csv(r'File Name.csv')
```

5. Diurnal Analysis-

```
def diurnal_analysis(table, plot_date, tp, ws, wd, ap, rh):
   if table == 'surface_data':
       point = 'IMD'
       station = 'GANGOTRI (1985-1989)<br>MAITRI (1990-2016)'
   else:
       if table =='weatherdata' or table == 'climate_data':
           point = ''
           station = ''
       else:
           point, station = table.split('_')
           station = station.upper()
           point = point.upper()

   lst = ['obstime']
   para = ''
   if tp == 'tempr':
       para += '| Temperature |'
       lst.append('tempr')
   if ws == 'ws':
       para += '| Wind Speed |'
       lst.append('ws')
   if wd == 'wd':
       para += '| Wind Direction |'
       lst.append('wd')
   if ap == 'ap':
       para += '| Air Pressure |'
       lst.append('ap')
   if rh == 'rh':
       para += '| Relative Humidity |'
       lst.append('rh')
```

```
   length = 5
   t = 1

   global conn
   q = ','.join(lst)
   query = "SELECT " + q + " FROM " + table + " WHERE CAST(obstime AS DATE)
= '" + plot_date + "'"
   lists = pd.read_sql(query, conn)

   lists['obstime'] = pd.to_datetime(lists['obstime'])
   lists = lists.set_index(lists['obstime'])
   fig = plt.figure(figsize=(12, 30))

   info = 'No Data available for this duration - '

   # arr stores selected parameters
   arr = []
   if(tp == 'tempr'):
       arr.append(tp)
   if(ws == 'ws'):
       arr.append(ws)
   if(wd == 'wd'):
       arr.append(wd)
   if(ap == 'ap'):
       arr.append(ap)
   if(rh == 'rh'):
       arr.append(rh)

   for col in arr:
       processed_list = lists[lists[col] != -999]
       processed_list = processed_list.sort_index()
       ax = plt.subplot(length, 1, t)
       ax.set_title('Diurnal Analysis',fontsize=18,weight='bold')
       ax.set_ylabel(attr_names[col])
       ax.set_xlabel('Time of Day')
       ax.plot(processed_list.index,processed_list[col])
       ticks=ax.get_xticks()
       plt.grid(True)
       ax.set

ax.set_xticks(np.linspace(ticks[0],mdates.date2num(mdates.num2date(ticks[-1
])+datetime.timedelta(hours=0)),5))
```

```
ax.set_xticks(np.linspace(ticks[0],mdates.date2num(mdates.num2date(ticks[-1
])+datetime.timedelta(hours=0)),5), minor=True)
       ax.xaxis.set_major_formatter(mdates.DateFormatter('%I:%M %p'))
       t = t+1

   fn = datetime.datetime.now().strftime("%Y_%m_%d_%H_%M_%S_%f") + '.png'
   fig.savefig("static/" + fn, bbox_inches='tight', pad_inches=0)
   if info == 'No Data available for this duration - ':
       info = ''

   plot_date = datetime.datetime.strptime(plot_date, '%Y-%m-%d')
   plot_date = plot_date.strftime('%m/%d/%Y')

   return '<h4 align="center">Seasonal Analysis</h4><h5 align="center"><b>'
+ station + '</b></h5><h5 align="center"><b>' \
           + point + '</b></h5><h5><p align="center">' + plot_date +
'</p><p align="center"><b>Parameters</b> ' \
           + para + '</h5></p><p align="center">' + info + '</p><img
id="graphImage" class="img img-thumbnail" src="\\static\\' \
           + fn + '">'
```

```
lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")

  use_python("/usr/local/bin/python")
  Sys.setenv(TENSORFLOW_PYTHON="/usr/local/bin/python")
  Sys.setenv(RETICULATE_PYTHON =
"/home/anirudh/anaconda3/bin/python")

  library("data.table")
  library("devtools")
  library("dplyr")
  library("forecast")
  library("shiny")
  library("stringi")
  library("keras")
  library("tensorflow")
```

```
  library("lubridate")
  library("zoo")
  library("xts")
  library("ggplot2")
  library("reticulate")
  library("ggfortify")
  library("broom")
  library("PerformanceAnalytics")
  library("timetk")
  library("shinyBS")
  library("ggthemes")
  library("openair")
  library("gridExtra")



forecasting_techniques=c("ARIMA"="arima","LSTM"="lstm","RNN"="rnn","C
NN"="cnn")

  date_to_posix=function(arg){
      arg=(as.POSIXct(arg,format="%d-%m-%Y %H:%M",tryFormats =
c("%m/%d/%Y %H:%M",
                                          "%Y-%m-%d",

"%Y/%m/%d"),origin='1970-01-01'))
      return(arg)
  }


  get_frequency=function(time_recorded){
      int_time_rec=time_recorded
      first_obs=as.numeric(int_time_rec[1])
      second_obs=as.numeric(int_time_rec[2])
      diff=second_obs-first_obs

      if(diff==60){
      return(60*24)
```

```
      }

      if(diff==3600){
      return(24)
      }

      if(diff==86400){
      return(365)
      }

      if(diff==2592000){
      return(12)
      }

  }




desired_ts_vector=function(requested_data,time_recorded,start_date,en
d_date,forecast_num){


start_time=as.numeric(as.POSIXct(start_date,format="%Y-%m-%d"),origin
= lubridate::origin)

end_time=as.numeric(as.POSIXct(end_date,format="%Y-%m-%d"),origin =
lubridate::origin)
      time_recorded=as.numeric(date_to_posix(time_recorded))
      requested_data_time=data.frame(time_recorded,requested_data)

      colnames(requested_data_time)=c("time_recorded","value")
      desired_range.df=requested_data_time %>%
filter(time_recorded>=start_time & time_recorded<=end_time)

      index_num=which(requested_data_time$time_recorded>end_time)
      req_index=index_num[1]
```

```
      to_be_fore=requested_data_time %>%
subset(as.numeric(rownames(requested_data_time))>=req_index &
as.numeric(rownames(requested_data_time))<req_index+forecast_num)
      desired_range.df=rbind(desired_range.df,to_be_fore)

      desired_range.df[,1]=desired_range.df[,1]+19800

      new_col=as_datetime(desired_range.df[,1],origin =
lubridate::origin, tz = "IST")

      desired_range.df=desired_range.df[-c(1)]
      desired_range.df["time_recorded"]=new_col
      desired_range.df[,c(1,2)]=desired_range.df[,c(2,1)]
      colnames(desired_range.df)=c("time_recorded","value")

      outlier_rows=which(desired_range.df$value< (-998)&
desired_range.df$value> (-1000))

      if(length(outlier_rows)){
      desired_range.df=desired_range.df[-c(outlier_rows),]
      }

      return(desired_range.df)
  }



  #check if timeseries has to be adjusted as well
  desired_timeseries=function(input_dataframe){
      time_recorded=input_dataframe[,"time_recorded"]
      freq=get_frequency(time_recorded)

temp.zoo=zoo(input_dataframe[,"value"],input_dataframe[,"time_recorde
d"])
      temp.timeseries=ts(temp.zoo,frequency = freq)
      return(temp.timeseries)
  }
```

```
  #check for all col if needs to be revised

desired_all_col_df=function(input_df_wo_time,time_recorded,start_date
,end_date){


start_time=as.numeric(as.POSIXct(start_date,format="%Y-%m-%d"),origin
= lubridate::origin)

end_time=as.numeric(as.POSIXct(end_date,format="%Y-%m-%d"),origin =
lubridate::origin)
     time_recorded=as.numeric(date_to_posix(time_recorded))

     requested_data_time=data.frame(time_recorded,input_df_wo_time)
     desired_range.df=requested_data_time %>%
filter(time_recorded>=start_time & time_recorded<=end_time)

     new_col=as_datetime(desired_range.df[,1],origin =
lubridate::origin, tz = "IST")
     desired_range.df=desired_range.df[-c(1)]
     desired_range.df["time_recorded"]=new_col

     outlier_rows=which(desired_range.df[,2]< (-998)&
desired_range.df[,2]> (-1000))

     if(length(outlier_rows)){
     desired_range.df=desired_range.df[-c(outlier_rows),]
     }

     return(desired_range.df)
  }


  #add the forecast_num to all predict_arima functions
  predict_arima=function(temp.timeseries,forecast_num){

     N = length(temp.timeseries)
```

```
      n=N-forecast_num
      train = temp.timeseries[1:n]
      test  = temp.timeseries[(n+1):N]

      decomp = stl(temp.timeseries, s.window="periodic")
      seasonal_comp=decomp$time.series[,1]
      to_be_added=seasonal_comp[n+1:N]
      deseasonal_cnt <- seasadj(decomp)
      fit=auto.arima(deseasonal_cnt, seasonal=FALSE)

      pred=predict(fit,n.ahead=length(test))

      for (i in 1:length(pred$pred)) {
      pred$pred[i]=pred$pred[i]+to_be_added[i]
      }

      return((pred$pred))
  }

  #check if CNN has the same requirements for n or 3 should be added

tidy_df_forecast=function(temp.timeseries,forecast_vec_df,forecast_ty
pe,forecast_num){
      N = length(temp.timeseries)
      n=N-forecast_num

      if(stri_cmp(forecast_type,"cnn")==0){
      n=n+3
      }

      train = temp.timeseries[1:n]
      test  = temp.timeseries[(n+1):N]

      tidy_timeseries=tidy(temp.timeseries)
      zeros_vec=numeric(n)
      test_arima<- c(zeros_vec,as.vector(forecast_vec_df))
      tidy_timeseries["forecast"]=test_arima
      tidy_timeseries=as.data.frame(tidy_timeseries)
```

```
      return(tidy_timeseries)
  }

  wind_dial_df=function(inp_df,time_col,start_date,end_date){
      vector_ws=find_type(inp_df,'ws')
      vector_wd=find_type(inp_df,'wd')


two_col_ws=desired_ts_vector(vector_ws,time_col,start_date,end_date,0
)

two_col_wd=desired_ts_vector(vector_wd,time_col,start_date,end_date,0
)

      colnames(two_col_ws)=c("date","ws")
      two_col_ws["wd"]=two_col_wd["value"]
      return(two_col_ws)
  }

  plot_multiple=function(inp_all_col_df,selected_column_vector){

      int_df=as.data.frame(inp_all_col_df[,selected_column_vector])
      normalized=scale(int_df)

      time_recorded=inp_all_col_df[,"time_recorded"]
      to_return=data.frame(time_recorded,normalized)
      return(to_return)
  }


  find_date_column=function(input.df){
      for(i in 1:ncol(input.df))
      {
      if(is.Date(input.df[1,i]) | is.POSIXct(input.df[1,i]) |
is.factor((input.df[1,i])))
      return(i)
      }
  }
```

```
  find_type=function(input_df,clicked_string){

      for(column in colnames(input_df))
      {
      if(stri_cmp(clicked_string,column)==0)
      return(input_df[,column])
      }

  }


desired_season_dataframe=function(inp_vector,time_recorded,year_inp){

      int_vector=inp_vector
      int_time_recorded=time_recorded
      summer_start=paste((as.numeric(year_inp)-1),'-12-01',sep = "")
      summer_end=paste(year_inp,'-02-28',sep = "")

      autumn_start=paste(year_inp,'-03-01',sep = "")
      autumn_end=paste(year_inp,'-05-31',sep = "")

      winter_start=paste(year_inp,'-06-01',sep = "")
      winter_end=paste(year_inp,'-08-31',sep = "")

      spring_start=paste(year_inp,'-09-01',sep = "")
      spring_end=paste(year_inp,'-11-30',sep = "")


summer_two_col=desired_ts_vector(int_vector,int_time_recorded,summer_
start,summer_end,0)

autumn_two_col=desired_ts_vector(int_vector,int_time_recorded,autumn_
start,autumn_end,0)

winter_two_col=desired_ts_vector(int_vector,int_time_recorded,winter_
start,winter_end,0)
```

```
spring_two_col=desired_ts_vector(int_vector,int_time_recorded,spring_
start,spring_end,0)


to_return=list(summer=summer_two_col,autumn=autumn_two_col,winter=win
ter_two_col,spring=spring_two_col)

     return(to_return)
  }


  blizzard_indices=function(inp_all_col_df){

     int_indices=which(inp_all_col_df$ws>23 & inp_all_col_df$rh>75)
     int_start=1
     iter=length(int_indices)
     start_vec=c()
     end_vec=c()

     num=0

     while (int_start+2<iter) {
     beg=int_start
     while (1) {
     mid_count=int_indices[int_start]
     final_count=int_indices[int_start+1]

     int_start=int_start+1
     num=num+1

     if((final_count-mid_count)!=1){
          break
     }

     }

     if(num>2){
```

```
      start_vec=c(start_vec,beg)
      end_vec=c(end_vec,int_start-1)
      }

      num=0
      }

      to_ret_df=inp_all_col_df[c(),]
      for (i in 1:length(start_vec)) {

to_ret_df=rbind(to_ret_df,inp_all_col_df[int_indices[start_vec[i]]:in
t_indices[end_vec[i]],])
      }
      return(to_ret_df)
  }



  ui=fluidPage(
      titlePanel("Interactive forecasting"),

      sidebarLayout(
      sidebarPanel(
      fileInput("file1", "Choose CSV File",
                accept = c(
                "text/csv",
                "text/comma-separated-values,text/plain",
                ".csv")
      ),
      uiOutput("data_type_dyn"),
      uiOutput("date_range_dyn"),

      numericInput("forecast_num", "Enter the number of values to
forecast:", 10, min = 1, max = 100),

      actionButton("forecast","forecast"),

      actionButton("plot","plot"),
```

```
      actionButton("zoom","zoom"),

      radioButtons("forecast_type","choose the type of forecasting
technique",forecasting_techniques),

      actionButton("correlation_matrix","Correlation Matrix"),

      actionButton("wind_dial","Wind Dial"),

      actionButton("dynamic_forecast","Dynamic Forecast"),

      numericInput("season_year_num", "Enter the Year for data season
wise: ", 2013, min = 2001, max = 2100),

      actionButton("season_plot","Season Plot"),

      uiOutput("multiple_data_type_dyn"),

      actionButton("plot_multiple", "Plot the selected parameters"),

      actionButton("plot_blizzard","Plot Blizzards")
      ),
      mainPanel(
      plotOutput("selected_data_type"),
      bsModal("modalExample", "Your plot", "zoom", size =
"large",plotOutput("zoom_plot"),downloadButton('downloadPlot',
'Download')),
      textOutput("accuracy_text")
      )

      )
  )


  server=function(input,output){

      reactive_df_wo_time=reactive({
```

```
      inFile <- input$file1
      filename=inFile$datapath
      data_read<-read.csv(file=filename, header=TRUE,
sep=",",stringsAsFactors = TRUE)
      date_col_num=find_date_column(data_read)
      data_read=data_read[-c(date_col_num)]
      return(data_read)
      })


      reactive_colnames=reactive({
      int_df=reactive_df_wo_time()
      return(colnames(int_df))
      })


      reactive_time_recorded=reactive({
      inFile <- input$file1
      filename=inFile$datapath
      data_read<-read.csv(file=filename, header=TRUE,
sep=",",stringsAsFactors = TRUE)
      date_col_num=find_date_column(data_read)
      time_recorded=data_read[,date_col_num]
      return(time_recorded)
      })


      reactive_wind_dial=reactive({
      int_df=reactive_df_wo_time()
      int_time_rec=reactive_time_recorded()

int_wind_dial=wind_dial_df(int_df,int_time_rec,input$daterange[1],inp
ut$daterange[2])
      return(int_wind_dial)
      })


      reactive_two_col_df=reactive({
```

```
      int_df=reactive_df_wo_time()
      int_time_recorded=reactive_time_recorded()
      desired_type=(find_type(int_df,input$data_type))

desired_range.df=(desired_ts_vector(desired_type,int_time_recorded,in
put$daterange[1],input$daterange[2],input$forecast_num))
      return(desired_range.df)
      })

      reactive_all_col_df=reactive({
      int_df=reactive_df_wo_time()
      int_time_recorded=reactive_time_recorded()

desired_all_col.df=desired_all_col_df(int_df,int_time_recorded,input$
daterange[1],input$daterange[2])
      return(desired_all_col.df)
      }))

      reactive_blizzard_df=reactive({
      int_all_col_df=reactive_all_col_df()
      int_two_col_df=reactive_two_col_df()
      int_blizzard=blizzard_indices(int_all_col_df)

      col_to_select=c(input$data_type,"time_recorded")
      int_blizz=int_blizzard[,col_to_select]
      colnames(int_blizz)=c("value","time_recorded")

      return_list=list(original=int_two_col_df,blizzard=int_blizz)
      return(return_list)
      })

      reactive_timeseries=reactive({
      int_two_col_df=reactive_two_col_df()
      temp.timeseries=desired_timeseries(int_two_col_df)
      return(temp.timeseries)
      })

      reactive_tidy_arima=reactive({
```

```
      int_two_col_df=reactive_two_col_df()
      temp.timeseries=desired_timeseries(int_two_col_df)
      predicted=(predict_arima(temp.timeseries,input$forecast_num))

int_tidy_arima_dataframe=tidy_df_forecast(temp.timeseries,predicted,"
arima",input$forecast_num)
      return(int_tidy_arima_dataframe)
      })

      reactive_tidy_lstm=reactive({
      int_two_col_df=reactive_two_col_df()
      temp.timeseries=desired_timeseries(int_two_col_df)
      predicted=(predict_lstm(temp.timeseries,input$forecast_num))

int_tidy_lstm_dataframe=tidy_df_forecast(temp.timeseries,predicted,"l
stm",input$forecast_num)
      return(int_tidy_lstm_dataframe)

      })

      reactive_tidy_rnn=reactive({
      int_two_col_df=reactive_two_col_df()
      temp.timeseries=desired_timeseries(int_two_col_df)
      predicted=(predict_rnn(temp.timeseries,input$forecast_num))

int_tidy_rnn_dataframe=tidy_df_forecast(temp.timeseries,predicted,"rn
n",input$forecast_num)
      return(int_tidy_rnn_dataframe)

      })

      reactive_tidy_cnn=reactive({
      int_two_col_df=reactive_two_col_df()
      temp.timeseries=desired_timeseries(int_two_col_df)
      predicted=(predict_cnn(temp.timeseries,input$forecast_num))

int_tidy_cnn_dataframe=tidy_df_forecast(temp.timeseries,predicted,"cn
n",input$forecast_num)
```

```
      return(int_tidy_cnn_dataframe)

      })

      reactive_arima_accuracy=reactive({
      int_two_col_df=reactive_two_col_df()
      int_timeseries=desired_timeseries(int_two_col_df)
      N = length(int_timeseries)
      n = N-input$forecast_num
      test  = int_timeseries[(n+1):N]

forecasted_arima=(predict_arima(int_timeseries,input$forecast_num))
      accu=accuracy(forecasted_arima,test)
      text=(paste("The ME value of the model is:  " , accu[1,"ME"]))
      return(text)
      })

      reactive_lstm_accuracy=reactive({
      int_two_col_df=reactive_two_col_df()
      int_timeseries=desired_timeseries(int_two_col_df)
      N = length(int_timeseries)
      n = N-input$forecast_num
      test  = int_timeseries[(n+1):N]

forecasted_lstm=(predict_lstm(int_timeseries,input$forecast_num))

      accu=accuracy(forecasted_lstm,test)
      text=(paste("The ME value of the model is:  " , accu[1,"ME"]))
      return(text)
      })

      reactive_rnn_accuracy=reactive({
      int_two_col_df=reactive_two_col_df()
      int_timeseries=desired_timeseries(int_two_col_df)
      N = length(int_timeseries)
      n = N-input$forecast_num
      test  = int_timeseries[(n+1):N]
      forecasted_rnn=(predict_rnn(int_timeseries,input$forecast_num))
```

```
      accu=accuracy(forecasted_rnn,test)
      text=(paste("The ME value of the model is:  " , accu[1,"ME"]))
      return(text)
      })

      reactive_cnn_accuracy=reactive({
      int_two_col_df=reactive_two_col_df()
      int_timeseries=desired_timeseries(int_two_col_df)

      N = length(int_timeseries)
      n = N-input$forecast_num+3
      test  = int_timeseries[(n+1):N]
      forecasted_cnn=(predict_cnn(int_timeseries,input$forecast_num))
      forecasted_cnn=as.ts(forecasted_cnn)
      accu=accuracy(forecasted_cnn,test)
      text=(paste("The ME value of the model is:  " , accu[1,"ME"]))
      return(text)
      })

      reactive_dynamic_forecast=reactive({
      int_two_col_df=reactive_two_col_df()

forecasted_dynamic=(dynamic_arima(int_two_col_df,input$forecast_num))
      })

      reactive_seasons_df=reactive({
      int_df=reactive_df_wo_time()
      int_time_recorded=reactive_time_recorded()
      desired_type=(find_type(int_df,input$data_type))

int_desired_season_df=desired_season_dataframe(desired_type,int_time_
recorded,input$season_year_num)
      return(int_desired_season_df)
      })

      reactive_multiple_var=reactive({
      int_all_col_df=reactive_all_col_df()
```

```
      int_mult=plot_multiple(int_all_col_df,input$param_choice)
      return(int_mult)
      })

      output$data_type_dyn=renderUI({

      column_names=reactive_colnames()
      inFile <- input$file1
      filename=inFile$datapath
      conditionalPanel(
      condition = "filename!=NULL",
      selectInput("data_type", label="Choose the type of
information",
                  choices = column_names)
      )

      })

      output$date_range_dyn=renderUI({

      time_recorded=reactive_time_recorded()
      time_recorded=date_to_posix(time_recorded)

start_date=(as.Date(as.numeric(time_recorded[1])/86400,format="%Y-%m-
%d"))

end_date=(as.Date(as.numeric(time_recorded[length(time_recorded)])/86
400,format="%Y-%m-%d"))

      dateRangeInput("daterange", "Date range:",
                     start  = start_date,
                     end   = end_date,
                     min   = start_date,
                     max   = end_date,
                     format = "yyyy/mm/dd",
                     separator = " - ")

      })
```

```
      output$multiple_data_type_dyn=renderUI({

      column_names=reactive_colnames()
      checkboxGroupInput("param_choice","Choose the parameters to be
plotted: ",choices = column_names)

      })

      interim=reactiveValues(x=NULL)

      observeEvent(input$plot,{
      output$selected_data_type=renderPlot({
      p=ggplot(data =
reactive_two_col_df(),aes(x=time_recorded,y=value)) +
            geom_line(aes(color = "Line joining data"), size = 1)
      p=p+ggtitle("Plot of requested data")+
            theme(panel.background = element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                              breaks = c("Line joining data"),
                              values = c("blue"))
      interim$x=p+theme(aspect.ratio = 1/4)
      print(p)
      })

      })

      observeEvent(input$zoom,{
      output$zoom_plot=renderPlot({
      (interim$x)
      })

      output$downloadPlot <- downloadHandler(
      filename = "Shinyplot.png",
      content = function(file) {
            png(file)
            interim$x
```

```
            dev.off()
      })

      })


      observeEvent(input$forecast,{
      output$selected_data_type=renderPlot({

      if(stri_cmp(input$forecast_type,"arima")==0)
      {
            int_tidy_arima=reactive_tidy_arima()
            total_rows=nrow(int_tidy_arima)
            partition_num=total_rows-input$forecast_num+1

int_tidy_arima_forecast=int_tidy_arima[partition_num:total_rows,]
            p=ggplot(data=int_tidy_arima)+
            geom_line(aes(x=index,y=value,color="Actual values"))+

geom_line(data=int_tidy_arima_forecast,aes(x=index,y=forecast,color="
Forecasted values"))+
            ggtitle("Arima Forecast")+  theme(panel.background =
element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                                breaks = c("Actual
values","Forecasted values"),
                                values = c("blue","red"))

            interim$x=p+theme(aspect.ratio = 1/4)
            print(p)
      }

      if(stri_cmp(input$forecast_type,"lstm")==0)
      {
            int_tidy_lstm=reactive_tidy_lstm()
            total_rows=nrow(int_tidy_lstm)
            partition_num=total_rows-input$forecast_num+1
```

```
int_tidy_lstm_forecast=int_tidy_lstm[partition_num:total_rows,]
          p=ggplot() +
          geom_line(data =
int_tidy_lstm,aes(x=index,y=value,color="Actual values")) +
          geom_line(data =
int_tidy_lstm_forecast,aes(x=index,y=forecast,color="Forecasted
Values"))+
          ggtitle("LSTM Forecast")+  theme(panel.background =
element_rect(fill = '#FFFFCC'))+
          scale_colour_manual("",
                              breaks = c("Actual
values","Forecasted values"),
                              values = c("blue","red"))
          interim$x=p+theme(aspect.ratio = 1/4)
          print(p)
     }

     if(stri_cmp(input$forecast_type,"rnn")==0)
     {
          int_tidy_rnn=reactive_tidy_rnn()
          total_rows=nrow(int_tidy_rnn)
          partition_num=total_rows-input$forecast_num+1

int_tidy_rnn_forecast=int_tidy_rnn[partition_num:total_rows,]
          p=ggplot() +
          geom_line(data =
int_tidy_rnn,aes(x=index,y=value,color="Actual Values")) +
          geom_line(data =
int_tidy_rnn_forecast,aes(x=index,y=forecast,color="Forecasted
Values"))+
          ggtitle("RNN Forecast")+  theme(panel.background =
element_rect(fill = '#FFFFCC'))+
          scale_colour_manual("",
                              breaks = c("Actual
values","Forecasted values"),
                              values = c("blue","red"))
          interim$x=p+theme(aspect.ratio = 1/4)
          print(p)
```

```
      }

      if(stri_cmp(input$forecast_type,"cnn")==0)
      {
            int_tidy_cnn=reactive_tidy_cnn()
            total_rows=nrow(int_tidy_cnn)
            partition_num=total_rows-input$forecast_num+1

int_tidy_cnn_forecast=int_tidy_cnn[partition_num:total_rows,]
            p=ggplot() +
            geom_line(data =
int_tidy_cnn,aes(x=index,y=value,color="Actual Values")) +
            geom_line(data =
int_tidy_cnn_forecast,aes(x=index,y=forecast,color="Forecasted
Values"))+
            ggtitle("CNN Forecast")+  theme(panel.background =
element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                                breaks = c("Actual
values","Forecasted values"),
                                values = c("blue","red"))
            interim$x=p+theme(aspect.ratio = 1/4)
            print(p)
      }

      })

      accuracy_text=reactive({
      if(stri_cmp(input$forecast_type,"arima")==0)
            {p=reactive_arima_accuracy()}

      else if(stri_cmp(input$forecast_type,"lstm")==0)
            {p=reactive_lstm_accuracy()}

      else if(stri_cmp(input$forecast_type,"rnn")==0)
            {p=reactive_rnn_accuracy()}

      else if(stri_cmp(input$forecast_type,"cnn")==0)
```

```
            {p=reactive_cnn_accuracy()}

      return(p)
      })

      output$accuracy_text=renderText({accuracy_text()})

      })

      observeEvent(input$correlation_matrix,{
      int_all_col_df=reactive_all_col_df()
      int_all_col_df=int_all_col_df[,-which(names(int_all_col_df)
%in% c("time_recorded"))]

output$selected_data_type=renderPlot({chart.Correlation(int_all_col_d
f, histogram=TRUE, pch=19)})
      })

      observeEvent(input$dynamic_forecast,{

      int_dynamic_forecast=reactive_dynamic_forecast()
      output$selected_data_type=renderPlot({
      p=ggplot() +
            geom_line(data =
int_dynamic_forecast$prev,aes(x=Date,y=.actual),color="blue") +
            geom_line(data =
int_dynamic_forecast$prev,aes(x=Date,y=.fitted),color="green")+
            geom_line(data =
int_dynamic_forecast$forecast,aes(x=Date,y=Point.Forecast),color="red
")

      print(p)
      })

      })

      observeEvent(input$wind_dial,{
      int_wind_dial=reactive_wind_dial()
```

```
      output$selected_data_type=renderPlot({
      polarFreq(int_wind_dial,type="season",hemisphere =
"southern",cols = "heat")
      })
      })

      observeEvent(input$season_plot,{

      int_season_df=reactive_seasons_df()
      output$selected_data_type=renderPlot({

      p1=ggplot(data =
int_season_df$summer,aes(x=time_recorded,y=value)) +
            geom_line(aes(color = "Line joining data"), size = 1)
      p1=p1+ggtitle("Summer Data")+
            theme(panel.background = element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                              breaks = c("Line joining data"),
                              values = c("blue"))

      p2=ggplot(data =
int_season_df$winter,aes(x=time_recorded,y=value)) +
            geom_line(aes(color = "Line joining data"), size = 1)
      p2=p2+ggtitle("Winter Data")+
            theme(panel.background = element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                              breaks = c("Line joining data"),
                              values = c("blue"))

      p3=ggplot(data =
int_season_df$autumn,aes(x=time_recorded,y=value)) +
            geom_line(aes(color = "Line joining data"), size = 1)
      p3=p3+ggtitle("Autumn Data")+
            theme(panel.background = element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                              breaks = c("Line joining data"),
                              values = c("blue"))
```

```
      p4=ggplot(data =
int_season_df$spring,aes(x=time_recorded,y=value)) +
            geom_line(aes(color = "Line joining data"), size = 1)
      p4=p4+ggtitle("Spring Data")+
            theme(panel.background = element_rect(fill = '#FFFFCC'))+
            scale_colour_manual("",
                                breaks = c("Line joining data"),
                                values = c("blue"))

      p_all=grid.arrange(p1,p2,p3,p4, ncol=2)
      interim$x=p_all+theme(aspect.ratio = 1/4)
      print(p_all)

      })
      })

      observeEvent(input$plot_multiple,{

      int_mult=reactive_multiple_var()
      int_mult <- melt(int_mult, id.vars="time_recorded")
      output$selected_data_type=renderPlot({
      ggplot(int_mult, aes(time_recorded,value, col=variable)) +
            geom_point() +
            geom_smooth()
      })
      })

      observeEvent(input$plot_blizzard,{
      total_blizzard=reactive_blizzard_df()
      original=total_blizzard$original
      blizzard=total_blizzard$blizzard

      print(original)
      print(blizzard)

      output$selected_data_type=renderPlot({
      p=ggplot() +
            geom_line(data=original ,aes(x=time_recorded,y=value),
```

```
color="blue",size = 1)+
          geom_point(data=blizzard ,aes(x=time_recorded,y=value),
color="red",size = 1)


     p=p+ggtitle("Blizzard data")+
          theme(panel.background = element_rect(fill = '#FFFFCC'))
     # scale_colour_manual("",
     #                     breaks = c("Line joining data"),
     #                     values = c("blue"))
     interim$x=p+theme(aspect.ratio = 1/4)
     print(p)


     })
     })

  }


  predict_lstm=function(temp.timeseries,forecast_num){

     diffed=diff(temp.timeseries,differences = 1)
     supervised = lag_transform(diffed, 1)

     N = nrow(supervised)
     n = N-forecast_num
     train = supervised[1:n, ]
     test  = supervised[(n+1):N,  ]


     Scaled = scale_data(train, test, c(-1, 1))

     y_train = Scaled$scaled_train[, 2]
     x_train = Scaled$scaled_train[, 1]

     y_test = Scaled$scaled_test[, 2]
     x_test = Scaled$scaled_test[, 1]
```

```
      dim(x_train) <- c(length(x_train), 1, 1)

      X_shape2 = dim(x_train)[2]
      X_shape3 = dim(x_train)[3]

      batch_size = 1
      units = 1


      model <- keras_model_sequential()
      model%>%
      layer_lstm(units, batch_input_shape = c(batch_size, X_shape2,
X_shape3), stateful= TRUE)%>%
      layer_dense(units = 1)

      model %>% compile(
      loss = 'mean_squared_error',
      optimizer = optimizer_adam( lr= 0.02, decay = 1e-6 ),
      metrics = c('accuracy')
      )

      Epochs = 50
      for(i in 1:Epochs ){
      model %>% fit(x_train, y_train, epochs=1,
batch_size=batch_size, verbose=1, shuffle=FALSE)
      model %>% reset_states()
      }

      L = length(x_test)
      scaler = Scaled$scaler
      predictions = numeric(L)

      for(i in 1:L){
      X = x_test[i]
      dim(X) = c(1,1,1)
      yhat = model %>% predict(X, batch_size=batch_size)
      # invert scaling
      yhat = invert_scaling(yhat, scaler,  c(-1, 1))
```

```
    # invert differencing
    yhat  = yhat + temp.timeseries[(n+i)]
    # store
    predictions[i] <- yhat
    }

    return(predictions)
  }

  lag_transform <- function(x, k= 1){

    lagged =  c(rep(NA, k), x[1:(length(x)-k)])
    DF = as.data.frame(cbind(lagged, x))
    colnames(DF) <- c( paste0('x-', k), 'x')
    DF[is.na(DF)] <- 0
    return(DF)
  }


  scale_data = function(train, test, feature_range = c(0, 1)) {
    x = train
    fr_min = feature_range[1]
    fr_max = feature_range[2]
    std_train = ((x - min(x) ) / (max(x) - min(x)  ))
    std_test  = ((test - min(x) ) / (max(x) - min(x)  ))

    scaled_train = std_train *(fr_max -fr_min) + fr_min
    scaled_test = std_test *(fr_max -fr_min) + fr_min

    return( list(scaled_train = as.vector(scaled_train),
scaled_test = as.vector(scaled_test) ,scaler= c(min =min(x), max =
max(x))) )

  }


  invert_scaling = function(scaled, scaler, feature_range = c(0, 1)){
    min = scaler[1]
```

```
    max = scaler[2]
    t = length(scaled)
    mins = feature_range[1]
    maxs = feature_range[2]
    inverted_dfs = numeric(t)

    for( i in 1:t){
    X = (scaled[i]- mins)/(maxs - mins)
    rawValues = X *(max - min) + min
    inverted_dfs[i] <- rawValues
    }
    return(inverted_dfs)
}



predict_rnn=function(temp.timeseries,forecast_num){

    diffed=diff(temp.timeseries,differences = 1)
    supervised = lag_transform(diffed, 1)

    N = nrow(supervised)
    n = N-forecast_num
    train = supervised[1:n, ]
    test  = supervised[(n+1):N,  ]


    Scaled = scale_data(train, test, c(-1, 1))

    y_train = Scaled$scaled_train[, 2]
    x_train = Scaled$scaled_train[, 1]

    y_test = Scaled$scaled_test[, 2]
    x_test = Scaled$scaled_test[, 1]
    dim(x_train) <- c(length(x_train), 1, 1)

    X_shape2 = dim(x_train)[2]
    X_shape3 = dim(x_train)[3]
```

```
      batch_size = 1
      units = 1


      model <- keras_model_sequential() %>%
      layer_gru(units = batch_size, input_shape = c( X_shape2,
X_shape3)) %>%
      layer_dense(units = 1)

      model %>% compile(
      optimizer = optimizer_rmsprop(),
      loss = "mae",
      metrics = c('accuracy')
      )


      Epochs = 50
      for(i in 1:Epochs ){
      model %>% fit(x_train, y_train, epochs=1,
batch_size=batch_size, verbose=1, shuffle=FALSE)
      model %>% reset_states()
      }

      L = length(x_test)
      scaler = Scaled$scaler
      predictions = numeric(L)

      for(i in 1:L){
      X = x_test[i]
      dim(X) = c(1,1,1)
      yhat = model %>% predict(X, batch_size=batch_size)
      # invert scaling
      yhat = invert_scaling(yhat, scaler,  c(-1, 1))
      # invert differencing
      yhat  = yhat + temp.timeseries[(n+i)]
      # store
      predictions[i] <- yhat
```

```
    }

    return(predictions)
  }



  predict_cnn=function(temp.timeseries,forecast_num){

    n_steps = 3

    N = length(temp.timeseries)
    n = N-forecast_num
    train = temp.timeseries[1:n]
    test  = temp.timeseries[(n+1):N]

    int_results=split_sequence(train,n_steps)
    X_train=int_results$X
    y_train=int_results$y

    int_results=split_sequence(test,n_steps)
    X_test=int_results$X
    y_test=int_results$y

    sample_num_train=length(X_train)/n_steps
    sample_num_test=length(X_test)/n_steps

    n_features=1

    dim(X_train)=c(sample_num_train,n_steps,n_features)
    dim(X_test)=c(sample_num_test,n_steps,n_features)


    model<-keras_model_sequential()

    model %>%
    layer_conv_1d(filters =
64,kernel_size=2,input_shape=c(n_steps,n_features) ) %>%
```

```
    layer_activation("relu") %>%
    layer_max_pooling_1d((pool_size=2)) %>%
    layer_flatten() %>%
    layer_dense(50) %>%
    layer_activation("relu") %>%
    layer_dense(1)


    model %>%
    compile(loss="mse",
          optimizer="adam",metrics = "accuracy")

    model %>% fit(X_train, y_train, epochs=100, verbose=0)

    predictions=model %>% predict(X_test)

    return(predictions)
}

split_sequence=function(temp.timeseries,n_steps){
    X=c()
    y=c()
    for (i in 1:(length(temp.timeseries)-n_steps)) {
    end_ix = i + n_steps-1
    X=c(X,temp.timeseries[i:end_ix])
    y=c(y,temp.timeseries[end_ix+1])
    }
    result=list(X=X,y=y)
    return(result)
}



dynamic_arima=function(inp_two_col,forecast_num){

    int_two_col=inp_two_col

    N=nrow(int_two_col)
```

```
      n=N-forecast_num

      train=int_two_col[1:n,]
      test=int_two_col[n+1:N,]

      time_recorded=train[,"time_recorded"]
      freq=get_frequency(time_recorded)
      train.zoo=zoo(train[,"value"],train[,"time_recorded"])
      train.ts=ts(train.zoo,frequency = freq)
      train.ts=train.ts[1:length(train.ts)]

      test.zoo=zoo(test[,"value"],test[,"time_recorded"])
      test.ts=ts(test.zoo,frequency = freq)

      fit_arima3 <- auto.arima(train.ts,d=0 )
      (fit_stochastic<- auto.arima(y=train.ts, d=1))

      fitdat=cbind.data.frame(Date=train$time_recorded[1:n]
,sweep::sw_augment(fit_arima3))

      test_augmented <- test %>%
      tk_augment_timeseries_signature()

      fcast3 <- forecast(fit_arima3, h = N-n)%>%data.frame()


dfuture2=data.frame(Date=test[1:length(fcast3),"time_recorded"],fcast
3)

      return_list=list(prev=fitdat,forecast=dfuture2)
      return(return_list)

  }

  shinyApp(ui, server)
```

Appendix B:

1. Formatting the CSV.

```
csvfile = csvname + '.csv'
file_path = process_data_path + csvfile

# to replace missing values with 999

# opens csv
text = open(file_path, "r")
text = ''.join([i for i in text]).replace("nan", "-999")
text = ''.join([i for i in text]).replace("--.-", "-999")
text = ''.join([i for i in text]).replace("--.--", "-999")
text = ''.join([i for i in text]).replace("----", "-999")
text = ''.join([i for i in text]).replace("--", "-999")
text = ''.join([i for i in text]).replace("-,", "-999,")
text = ''.join([i for i in text]).replace("-999-999,", "-999,")
text = ''.join([i for i in text]).replace(" -999,", " 00:00,")
# write back to csv
x = open(file_path, "w+")
x.writelines(text)
x.close()
```

2. Storing data from csv to database

```
try:
   conn = psycopg2.connect("dbname='data_analysis' user='username'
host='localhost' port=5432 password='password'")
except:
   print("Connection Error")

db_table = 'iig_bharati'  # database table name where the data should add

cur = conn.cursor()

for f in files:
```

```
    file_name = os.path.basename(f)
    csvname = file_name[0:-5]

    # avoid xlsx files
    if file_name[-4:] != "xlsx":
        continue
    # if file is csv, then append data to db_table
    with open(process_data_path + csvname + '.csv', 'r') as f:
        cur.copy_from(f, db_table, sep=',')

    conn.commit()
    f.close()
```

Appendix C:

1. Quarterly plot code

```
lists = lists[lists.index.year == int(year)]
            lists = lists.resample('D').mean()
            lists.dropna(inplace=True)
            lists['month'] = [d.strftime('%b') for d in lists.index]
            plot_handles=[]
            n = 4
            color=iter(cm.rainbow(np.linspace(0,1,n)))
            for sea in seasons:
                c=next(color)
                plt.subplot(length, 1, t)
                series = lists.loc[ lists['month'].isin(seasons[sea])]
                series['day'] = np.arange(series.shape[0]) + 1
                plt.plot(series['day'], series[col], color=c)
                plot_handles.append(mpatches.Patch(color=c,
label=season_names[sea]))
                plt.grid(True)
            plt.xlabel('Number of days in one season (' +year+ ')' )
            plt.ylabel(attr_names[col])
            plt.legend(loc=1, handles=plot_handles)
```

2. Year Wise Comparison Plot

```
      lists = lists.resample('M').mean()
           lists['year'] = [d.year for d in lists.index]
           lists['month'] = [d.strftime('%b') for d in lists.index]
           years = lists['year'].unique()
           lists.dropna(inplace=True)
           # print(years)
           plot_handles = []
           n = years.size
           color=iter(cm.rainbow(np.linspace(0,1,n)))
           for i, y in enumerate(years):
               c = next(color)
               plt.subplot(length, 1, t)
               plt.plot('month', col, data=lists.loc[(lists['year']==y), :],
label=y, color=c)
               # plt.text(lists.loc[(lists['year']==y), :].shape[0]-.9,
lists.loc[lists['year']==y, col][-1:].values[0], y, fontsize=12)
               plot_handles.append( mpatches.Patch(color=c, label=y))
               plt.grid(True)
           plt.xlabel('Months')
           plt.ylabel(attr_names[col])
           plt.legend(loc=1, handles=plot_handles)
```

3. Seasonal Comparison Plot

```
      lists = lists.resample('D').mean()
           lists['year'] = [d.year for d in lists.index]
           lists['month'] = [d.strftime("%b") for d in lists.index]
           years = lists['year'].unique()
           months = seasons[seas]
           lists = lists.loc[ lists['month'].isin(months)]
           plot_handles=[]
           n = years.size
           color=iter(cm.rainbow(np.linspace(0,1,n)))
           for i, y in enumerate(years):
               c=next(color)
               plt.subplot(length, 1, t)
               # plt.plot('month', col, data=lists.loc[(lists['year']==y), :],
label=y)
```

```
                # plt.text(lists.loc[(lists['year']==y), :].shape[0]-.9,
lists.loc[lists['year']==y, col].values[0], y, fontsize=12)
                series = (lists.loc[ lists['year']==y])
                series['day'] = np.arange(series.shape[0]) + 1
                # print(series)
                plt.plot(series['day'], series[col], color=c)
                plot_handles.append(mpatches.Patch(color=c, label=y))
                plt.grid(True)
            plt.xlabel('Number of days in one season')
            plt.ylabel(attr_names[col])
            plt.legend(loc=1, handles=plot_handles)
```

References

1. http://data.ncaor.gov.in
2. Choudhary, P. (2019). *Introduction to Anomaly Detection*. [online] Datascience.com. Available at: https://www.datascience.com/blog/python-anomaly-detection [Accessed 11 Jul. 2019].
3. Medium. (2019). *Time Series of Price Anomaly Detection - Towards Data Science*. [online] Available at: https://towardsdatascience.com/time-series-of-price-anomaly-detection-13586cd5ff46 [Accessed 11 Jul. 2019].
4. "Time Series of Price Anomaly Detection - Towards Data Science." *Medium*, Towards Data Science, 24 Jan. 2019, towardsdatascience.com/time-series-of-price-anomaly-detection-13586cd5ff46.
5. pandas.plotting.scatter_matrix — pandas 0.24.2 documentation. (2019). Retrieved from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.scatter_matrix.html
6. PLOT | Science Data Integration Group - Ferret Support. (2019). Retrieved from https://ferret.pmel.noaa.gov/Ferret/documentation/users-guide/commands-reference/PL
7. (2015, February 10) RAD - Outlier Detection on Big Data, [Online]. Available: https://medium.com/netflix-techblog/rad-outlier-detection-on-big-data-d 6b0494371.
8. Keras.io. (2019). *About Keras models - Keras Documentation*. [online] Available at: https://keras.io/models/about-keras-models/ .
9. Matplotlib.org. (2019). *Matplotlib: Python plotting — Matplotlib 3.1.1 documentation*. [online] Available at: https://matplotlib.org/ [Accessed 12 Jul. 2019].
10. "How to Identify and Remove Seasonality from Time Series Data with Python." *Machine Learning Mastery*, 26 Apr. 2019, machinelearningmastery.com/time-series-seasonality-with-python/.