A REPORT

ON

WEATHER ANALYSIS, VISULIZATION AND PREDICTION USING R AND PYTHON

BY

Bhargav Nutalapati (ID No: 2021B3AA0815H)

Prepared in partial fulfillment of the

practice school – 1 Course Nos.

BITS C221/BITS C231/BITS C241

AT

(National Center for Ocean and Polar Research)

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(July 2023)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)
Practice School Division

Station: National center for Ocean and Polar Research

Duration: 54 days

Date of Start: 30-05-23.

Date of Submission 18-07-23

Title of the Project: WEATHER ANALYSIS, VISULIZATION AND PREDICTION USING R AND PYTHON

ID No./Name(s)/ Discipline(s)/of the student(s): Bhargav Nutalapati (20210815)

Name(s) and designation(s) of the expert(s): Dr. Sakthivel Samy

Name(s) of the PS Faculty: Apurba Chakraborty

Project Areas: Data visualization and prediction of weather

ACKNOWLEDGEMENTS

I would like to thank BITS Pilani for giving me this once in a lifetime opportunity of working with such a prestigious institute, NCPOR. I would like to thank the PS division for giving me this chance to experience in the field of research.

Thanks to our PS mentor Mr. Apurba Chakraborty for being so supportive and thanks to Dr Sakthivel Samy for giving me the opportunity to work under his expert guidance and taking time out of his busy schedule for helping me.

TABLE OF CONTENTS

Sl.no	Content	Page no.
1	Introduction	5
2	Plotting color coded seasonal polar plots using temperature wind direction and wind speed.	6
3	Plotting wind speed and air pressure from Bharathi station.	8
4	Plotting wind speed, wind direction and temperature from Bharathi station.	10
5	Plotting wind speed, wind direction and air pressure from Bharathi station.	11
6	Prediction of temperature using Deep learning	13
7	Predicting temperature using ANN.	16
8	Predicting temperature using CNN.	17
9	Predicting temperature using RNN.	19
10	predicting temperature using LSTM.	20
11	Using K – Fold cross validation to validate deep learning models.	22
12	Classification of blizzards using Deep learning algorithms.	23
13	Prediction of blizzards	27
14	Conclusion	28
15	References	29

Introduction:

As the only institute in India that researches on the polar regions it is important to know about the weather parameters and their trends there, and so it is natural to have graphs and plots that show the trends and visualize these weather parameters (air pressure, wind direction, temperature). Here we used R programming to visualize these parameters plotting them in R Studio. We used R as it is powerful statistical software which can be used to plot and visualize diverse types of data.

Apart from analyzing and visualizing existing data it is also important to identify the trends and to predict the future weather data, for this we used Machine Learning algorithm (artificial neural networks) to predict the data for the next few time points in the data. This is done by using different packages in python including packages such as 'TensorFlow,' 'matplotlib,' 'seaborn' etc.… we predicted the data and plotted the future values using the 'matplotlib' in python and have done classification of blizzard dates to predict them.

ANALYSIS AND VISUALIZATION OF WEATHER IN R

PLOTTING COLOUR CODED SEASONAL POLAR PLOTS USING TEMPERATURE, WIND DIRECTION AND WIND SPEED.

The data is from an Antarctic weather station, and it contains the data on temperature(C), wind direction(degrees), wind speed(m/s) and time of the observation taken. The observation is taken at an interval of three hours each from 1985-02-26 00:08:50 up to 2010-12-31 21:08:50, time is given in the format of number of hours lapsed from the initial time, so it must be converted into normal (yyyy-mm-dd hh:mm: ss) format for easier working.

We used openair package available in R which was primarily used by environmentalists to plot wind direction wind speed and the pollutants the wind carries. There is a function polarFreq () that divides the wind into bins of a range of wind direction and wind speed and within each bin it calculates the mean amount of pollutants and color codes them. Using pollutant as temperature we were able to obtain the required polar graph, and as the package can be used to divide the years into seasons, we can even do it based on each season.

The data is divided into 26 years and each year into four seasons giving 104 polar plots.

Each year is divided into.

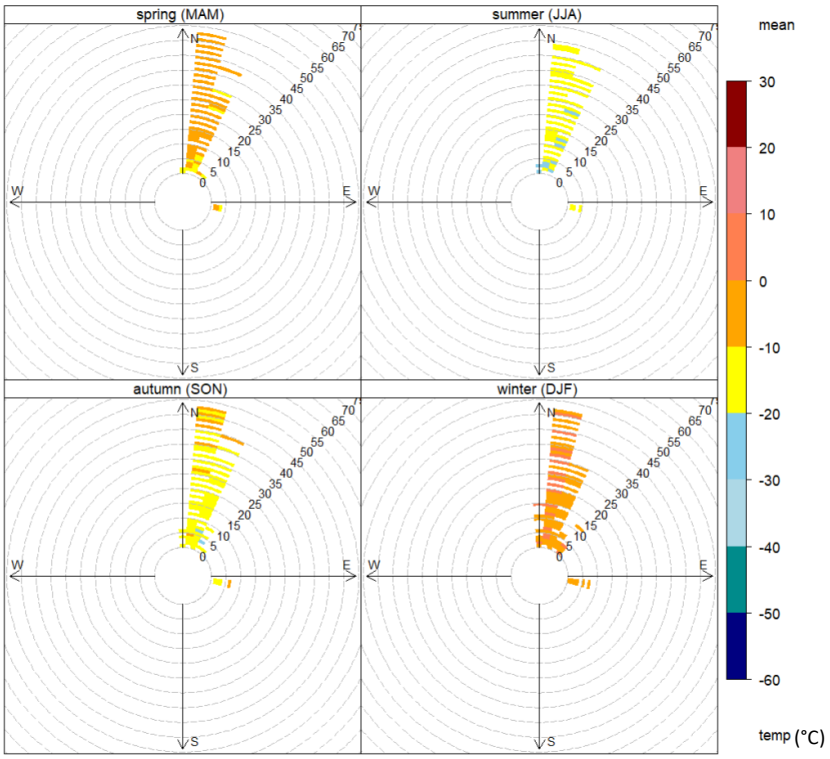
- Summer
- Spring
- Autumn
- Winter

Code for the graphs:

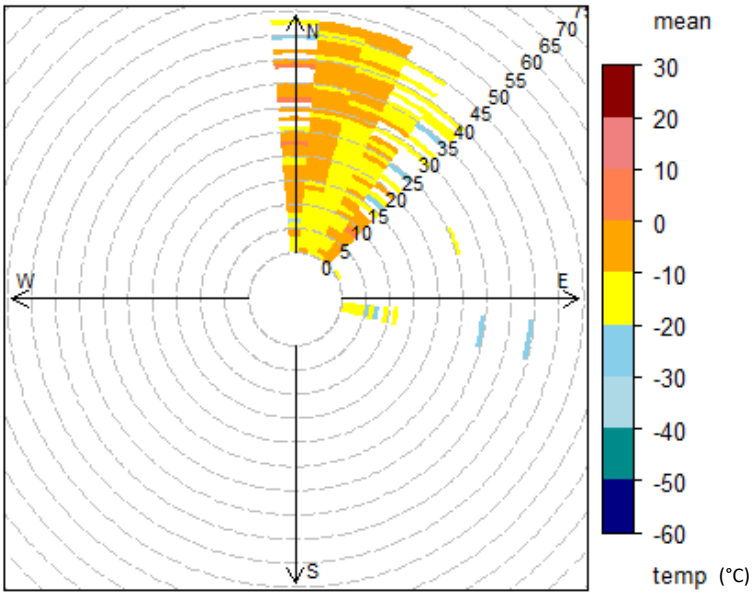
```
for(year in 1985:2010)
{
  year_data <- n_data[n_data$year == year, ]
  polarFreq(year_data,pollutant = "temp",
            statistic = "mean",
            type = "season",
            breaks = seq(-60,30,5),
            cols = "turbo",
            fontsize = 30 )

}
```

Here “n_data” contains the data on date, temperature, wind speed and wind direction.

Below is a graph for one year.



Below is for entire data across the years.



From the above graph divided by seasons we can infer that the temperature in Antarctica is warmer during the winter and colder during the summer as it lies in the southern hemisphere.

PLOTTING WIND SPEED AND AIR PRESSURE FROM BHARATHI STATION.

This other data is collected from Bharathi station, which is a research center of India in antarctica, this contains date temperature, air pressure, wind speed, and wind direction. There are two subsets of graphs from this data, the first is the graph that has sir pressure and wind speed plotted along time on the x – axis.

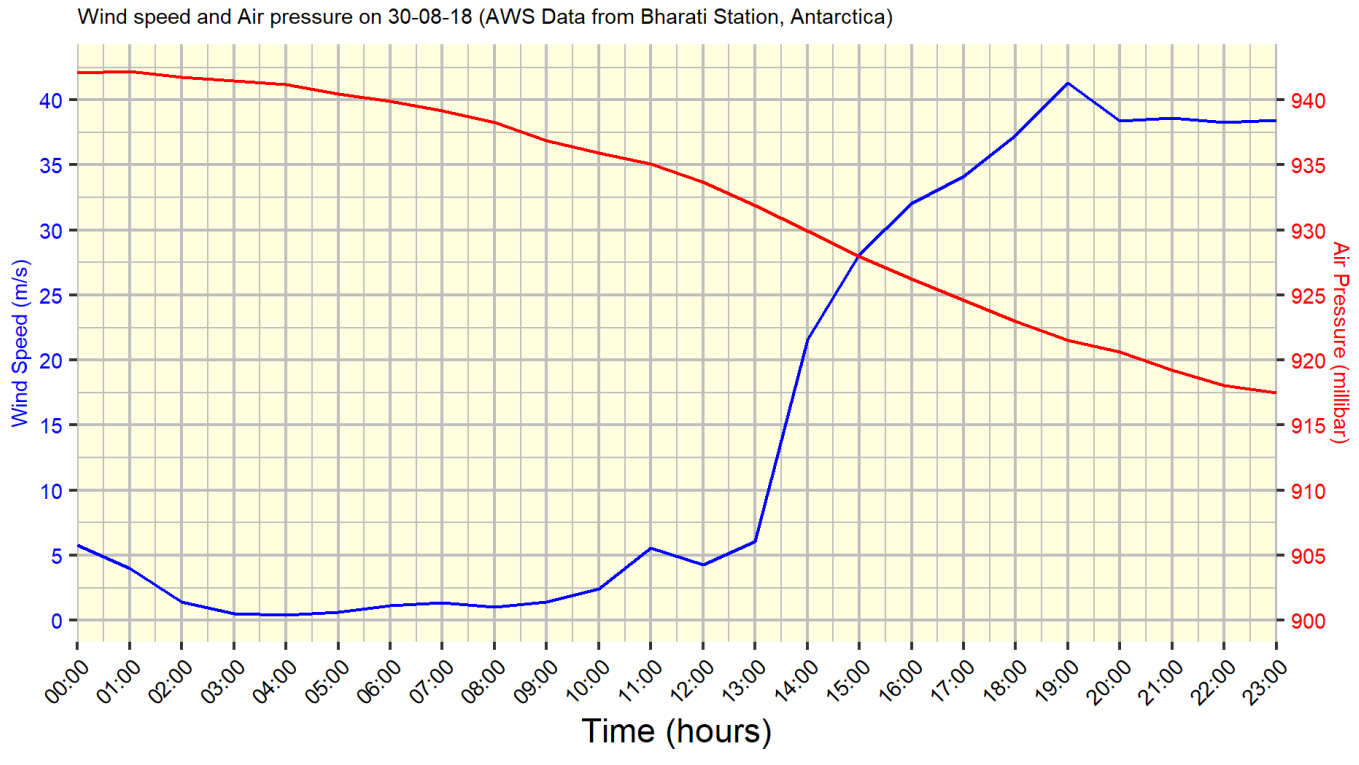
We go into plotting the graphs with the hypothesis that higher the air pressure, lower the wind speed, here we used ‘ggplot2’ package for easier plotting of the graphs.

There were ten dates where weather conditions were blizzard, and we were to plot the graphs for those ten dates. Blizzard dates: 20-07-2016, 26-05-2017, 27-05-2017, 28-05-2017, 30-08-2018, 31-08-2018, 9-06-2019, 10-06-2019, 11-05-2020, 12-05-2020.

Code for the wind speed and wind air pressure graphs:

```
ggplot(data_30_08_18)+
    geom_line(aes(date,ws),colour = "blue")+
    geom_line(aes(date,ap-950),colour = "red")+
    scale_y_continuous(name = "Wind Speed (m/s)",
        sec.axis = sec_axis(~ .+900, name = "Air Pressure (millibar)",breaks = seq(900,950,5)),
        breaks = seq(0,50,5))+
    scale_x_datetime(name = "Time (hours)",
        breaks = seq(as.POSIXct("2018-08-30 00:00:00"),
        as.POSIXct("2018-08-30 23:00:00"),"1 hour"),
        labels = function(x) format(x, "%H:%M"),
        expand = c(0,0))+
    theme(axis.text.y = element_text(color = "blue",size = 7),
        axis.text.y.right = element_text(color = "red"),
        axis.title.y = element_text(color = "blue",size = 7),
        axis.title.y.right = element_text(color = "red"),
    panel.background = element_rect(fill = "lightyellow"),
    panel.grid = element_line(color = "grey"),
    axis.text.x = element_text(angle = 45, hjust = 1),
    axis.text.x.bottom = element_text(size = 7, colour = "black"),
    plot.title = element_text(color = "black",size = 7))+
    labs(title = "Wind speed and Air pressure on 30-08-18 (AWS Data from Bharati Station, Antarctica)")
```

The graph below is for 30-08-18.



PLOTTING WIND SPEED, WIND DIRECTION AND TEMPERATURE FROM BHARATHI DATA

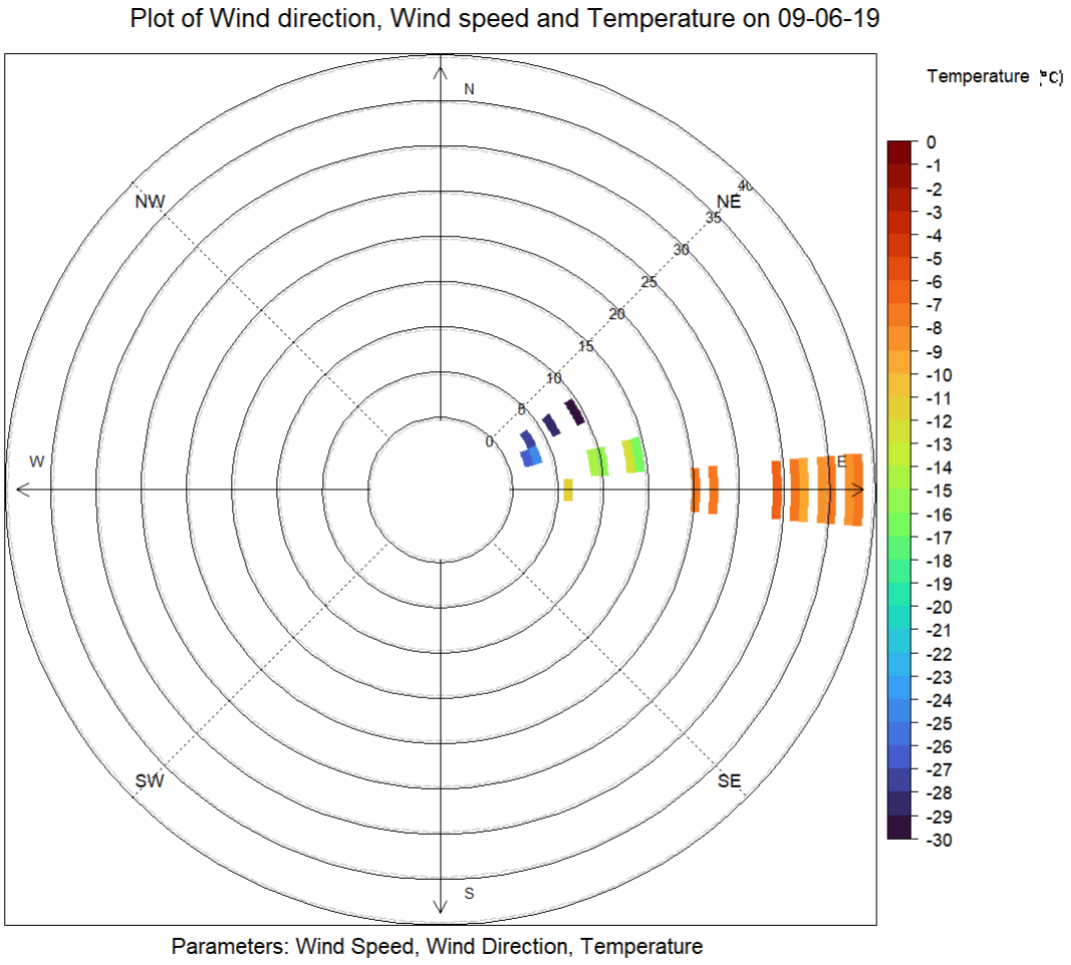
For those ten-blizzard weather conditioned dates we plotted the graphs by dividing the polar plot into bin in range of wind speeds and wind directions and took mean temperature of the bin and color coded it.

The graphs were not visually pleasing as the circular plots extend way over the maximum wind speed limit, so we used the package 'lattice' from which we used 'trellis object' and added a layer of white sheet over the outer part of the graph.

Code for polar graph of wind speed, temperature, and wind direction:

```
radii <- seq(8,60,5)
arc_df <- arc(radii)
polarFreq(data_09_06_19,pollutant = "temp",
          statistic = "mean",
          breaks = seq(-30,0,1),
          cols = "turbo",
          key.header = "Temperature",
          key.footer = NULL,
          xlab = "Parameters: Wind Speed, Wind Direction, Temperature",
          main = "Plot of Wind direction, Wind speed and Temperature on 09-06-19",
          fontsize = 35,
          lty = "dotted")
trellis.last.object() +
    layer(lpolygon(x = arc_df$x,y=arc_df$y,lty = "solid",lwd = 1))
```

We made ten graphs that show temperature on wind direction and windspeed for which one of the graphs is given below.



PLOTTING WIND SPEED, WIND DIRECTION AND AIR PRESSURE FROM BHARATHI DATA

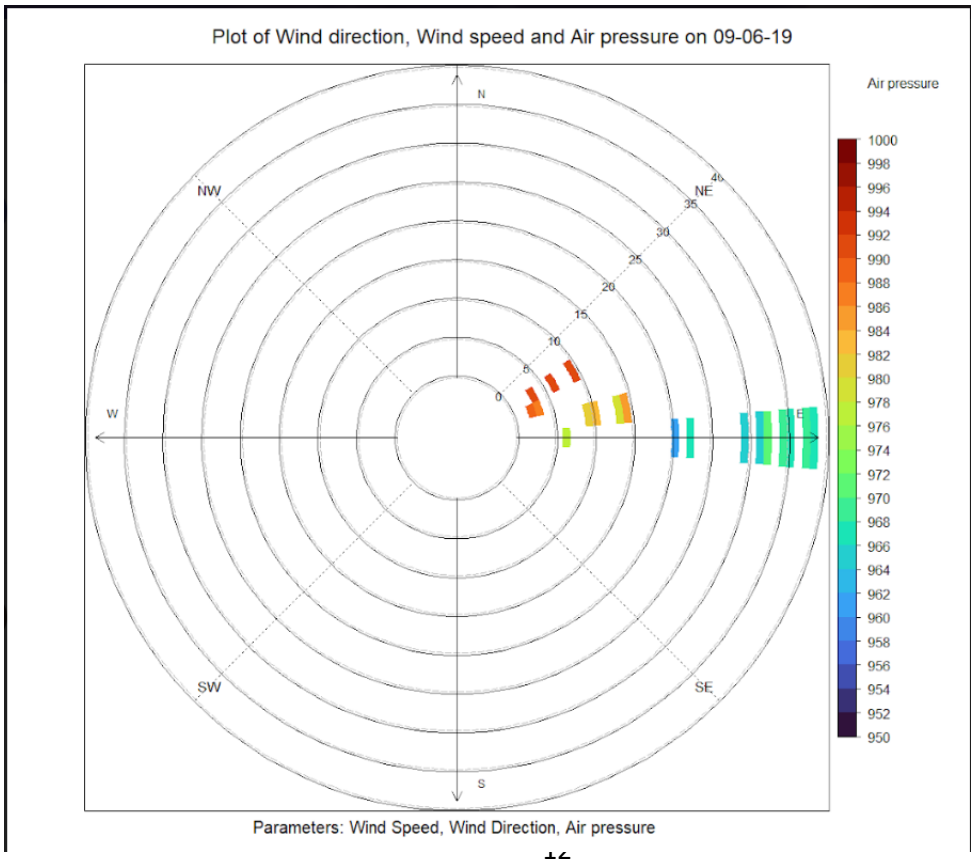
For those ten-blizzard weather conditioned dates we plotted the graphs by dividing the polar plot into bin in range of wind speeds and wind directions and took mean air pressure of the bin and color coded it.

We again applied a layer on top of the graph to remove the outer part as we did above.

Code for polar graph of wind speed, air pressure, and wind direction:

```
radii <- seq(8,60,5)
arc_df <- arc(radii)
polarFreq(data_09_06_19,
          pollutant = "ap",
          statistic = "mean",
          breaks = seq(950,1000,2),
          cols = "turbo",
          key.header = "Air Pressure",
          key.footer = NULL,
          xlab = "Parameters: Wind Speed, Wind Direction, Air Pressure",
          main = "Plot of Wind direction, Wind speed and  Air Pressure on 09-06-19",
          fontsize = 35,
          lty = "dotted")
trellis.last.object() +
    layer(lpolygon(x = arc_df$x,y=arc_df$y,lty = "solid",lwd = 1))
```

We made ten graphs that show air pressure on wind direction and windspeed for which one of the graphs is given below.



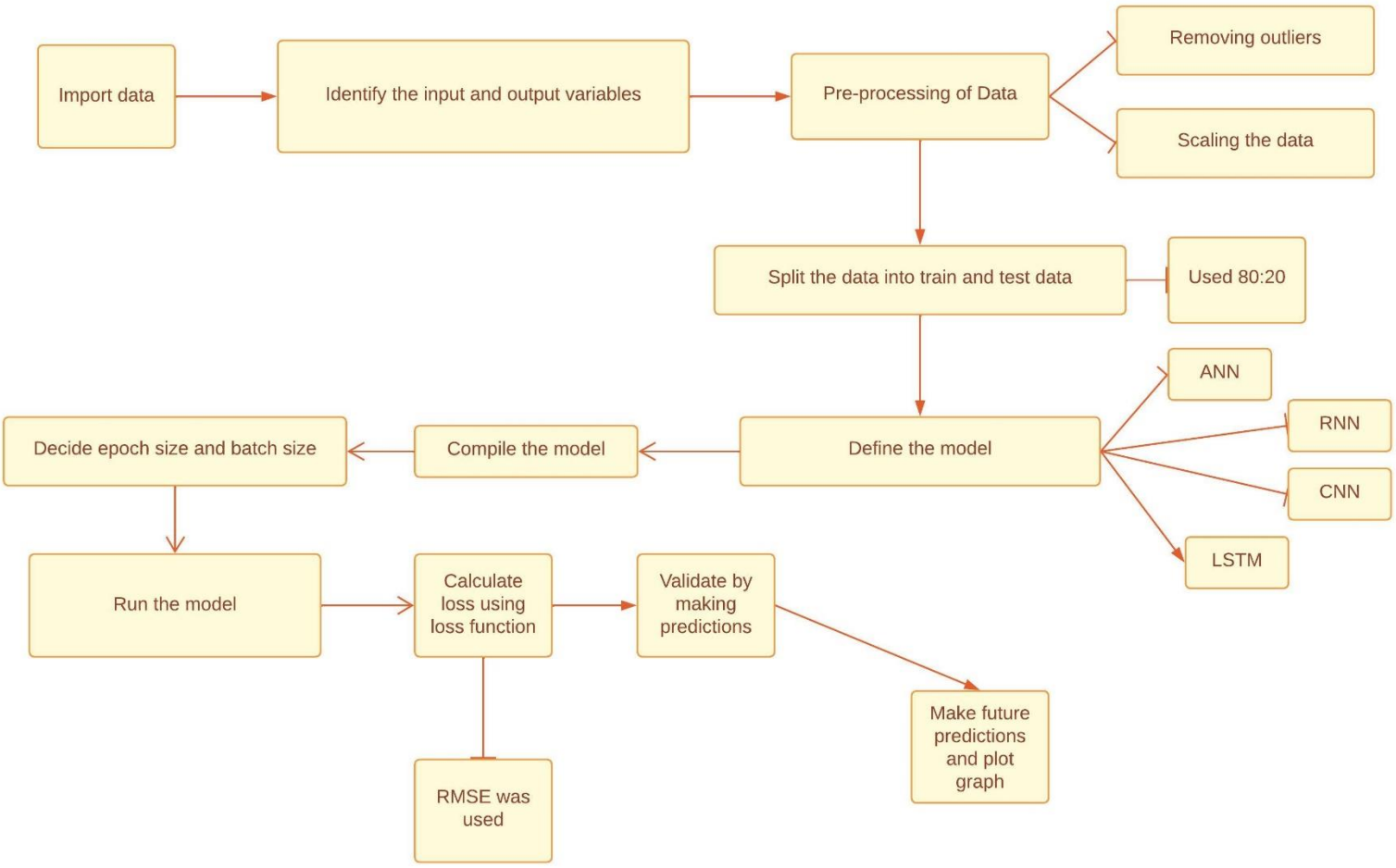
PREDICTING TEMPERATURE USING DEEP LEARNING ALGORITHMS

Given Bharathi data we predicted the temperature for the next three days (seventy-two values for three days) using the previous two weeks temperatures as inputs. For this we used 'TensorFlow,' 'keras,' 'sklearn,' 'numpy', 'pandas' packages in python to create algorithms and used packages such as 'matplotlib' to plot the predicted temperatures.

For predicting the temperature, we use four different algorithms.

- ANN (Artificial neural networks)
- CNN (convolutional neural networks)
- RNN (Recurrent neural network)
- LSTM (Long short-term memory algorithm)

For each of the algorithms we follow same flow of code except for the part in which we define the model we use; the common part is shown in the below flow chart.



The code for importing and preprocessing data:

```
weather_data = pd.read_csv(r"##PATH")
weather_data['date'] = pd.to_datetime(weather_data['date'], format='%Y-%m-%d %H:%M:%S')
weather_data['month'] = weather_data['date'].dt.month
weather_data['hour'] = weather_data['date'].dt.hour
weather_data['year'] = weather_data['date'].dt.year
weather_data.loc[weather_data['temp']<-70, 'temp'] = -70
weather_data = weather_data.fillna(0)
weather_data
```

Data is stored in a data frame called “weather_data.” And all the values that are not recorded are filled with zero.

Code for splitting the data into test and train:

```
window_size = (7*24*2)
X, y = [], []
for i in range(len(weather_data) - window_size - (3*24)):
    X.append(weather_data['temp'][i:i+window_size])
    y.append(weather_data['temp'][i+window_size])
X = np.array(X)
y = np.array(y)

train = int(len(X)*0.8)
X_train = X[:train]
y_train = y[:train]
X_test = X[train:]
y_test = y[train:]
```

Here input is taken as the previous two weeks and output as next three days, the test train data is split into 80:20.

The window size (number of temperature points given to predict the future temperature) is 336 (which is 7*24*2, two weeks). With this the future seventy-two values (three days) are predicted.

Code for validation:

```
predict = []
for i in range(62500, len(weather_data) - window_size):
    asdf = weather_data['temp'][i: i + window_size]
    asdf = np.array([asdf])
    aaa = model.predict(asdf)
    pre_temp = aaa[0][0]
    predict.append(pre_temp)
x = 0
weather_data['predict'] = 0
for i in range(62500+window_size,63648):
    weather_data['predict'][i] = predict[x]
    x = x+1
weather_data
```

Code for prediction:

```
future = []
asdf = weather_data['temp'][-(window_size): ]
for i in range(3*24):
    asdf = np.array([asdf])
    aaa = model.predict(asdf)
    pre_temp = aaa[0][0]
    future.append(pre_temp)
    asdf = np.append(asdf[0],pre_temp)[-(window_size): ]
date_index = pd.date_range(start='2015-02-06 00:00:00', end='2022-05-14 23:00:00', freq='1H')
date_list = date_index.to_list()
temperatures =  weather_data['temp'].to_list()
temperatures = temperatures + future
df = pd.DataFrame(date_list)
df['date'] = date_list
df['temperatures'] = temperatures
df['future'] = temperatures
df['temperatures'][ 63647: ] = 0
df['future'][ :63647] = 0
df
```

The above codes are common for all the four algorithms.

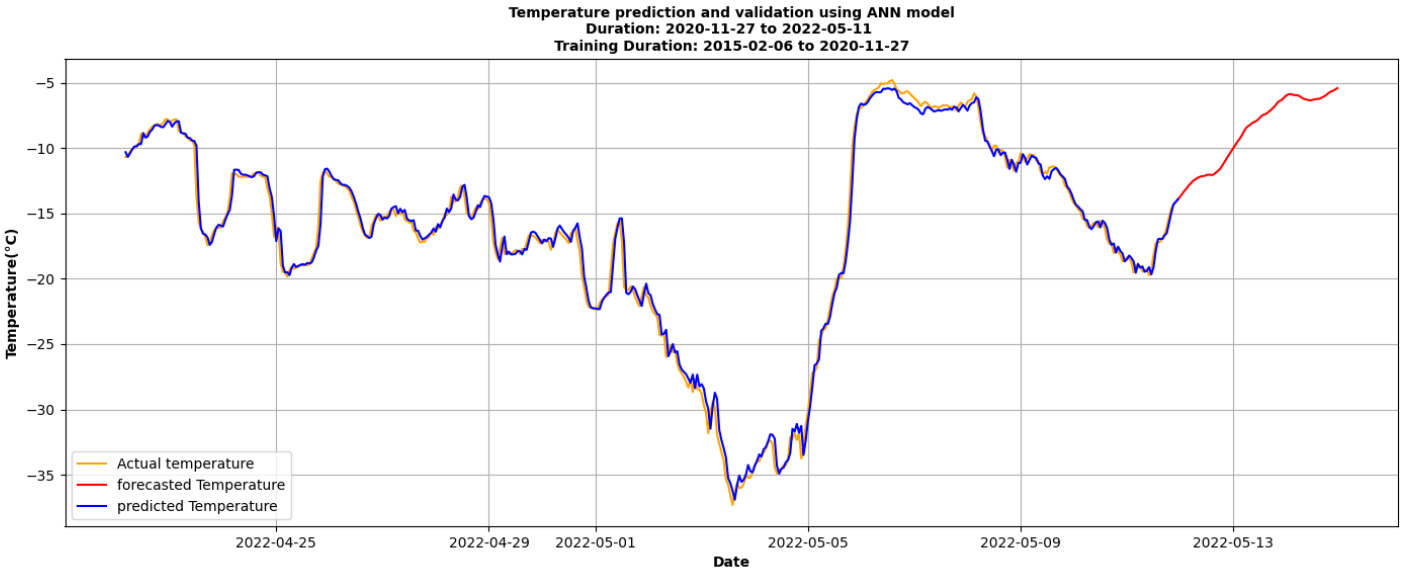
PREDICTION OF TEMPERATURE USING ARTIFICIAL NEURAL NETWORKS

ANN involves passing data through the network, where each neuron performs a computation on the received input and produces an output. The output of one neuron serves as the input for the neurons in the next layer. Computations involve multiplying the inputs by their corresponding weights, summing them up, and applying an activation function to introduce non-linearity. This is used in the model where there is one hidden layer of sixty-four nodes and the activation function is “softplus.”

Code for ANN model:

```
model = Sequential()
model.add(Dense(64, activation='softplus', input_shape=(window_size,)))
model.add(Dense(64, activation='softplus'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=200, batch_size=100, verbose=1)
loss = model.evaluate(X_test, y_test)
print('Mean Squared:', loss)
```

The prediction using the above model is shown below:



The loss for the model came out to be 0.41.

Merits:

ANN is a fast model with the power to do parallel processing and can learn complex patterns and can change internal patterns with new data.

Demerits:

ANN is sensitive to noise; it is important to preprocess the data. It is also difficult to understand the working of how the model made a conclusion.

PREDICTION OF TEMPERATURE USING CONVOLUTION NEURAL NETWORKS

CNN is particularly effective in handling grid-like data structures, one of the key components of CNNs is the convolutional layer. This layer applies a set of learnable filters (also known as kernels) to the input data, performing convolutions across the grid. These convolutions effectively detect local patterns and spatial relationships, enabling the network to identify features such as temperature gradients, spatial correlations, and other relevant patterns.

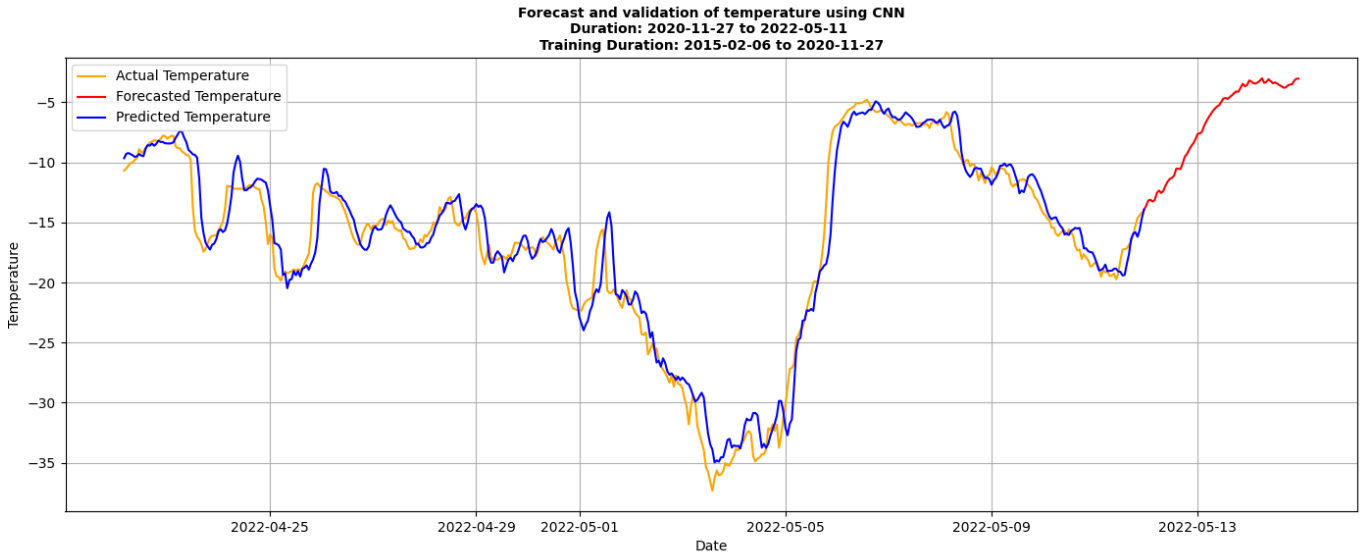
The combination of convolutional and pooling layers enables CNNs to progressively learn and extract spatial features at different scales. These features are then fed into fully connected layers, which perform classification or regression tasks, in this case, temperature prediction.

Code for CNN model:

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='sigmoid', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32, kernel_size=3, activation='sigmoid'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=1)
loss = model.evaluate(X_test, y_test)
print('Mean Squared:', loss)
```

We first added a 1D convolutional layer with sixty-four filters and set the size of each filter to three. The activation function used was sigmoid, which squashes the output values between 0 and 1. We further add a 1D Max pooling layer to the model. The steps are repeated with a convolutional layer with thirty-two filters. To finish it off, the Flatten layer reshapes the multi-dimensional output from the previous layer into a one-dimensional vector.

Below are the predictions using CNN:



The loss came out to be 2.34.

Merits:

CNN could manage substantial amounts of data by exploiting local connectivity and shared weights.

Demerits:

CNN is unable to oversee sequential data unlike RNN and LSTM and it needs high computational power.

PREDICTION OF TEMPERATURE USING RECURRENT NEURAL NETWORK

Recurrent Neural Networks (RNNs) are a type of artificial neural network specifically designed to manage sequential data. Unlike feedforward neural networks, RNNs introduce feedback connections that allow information to be passed from one step in the sequence to the next. This recurrent nature enables RNNs to retain memory of past inputs, making them suitable for modeling and predicting time-dependent data, including temperature. In temperature prediction, RNNs can consider the sequential nature of temperature values.

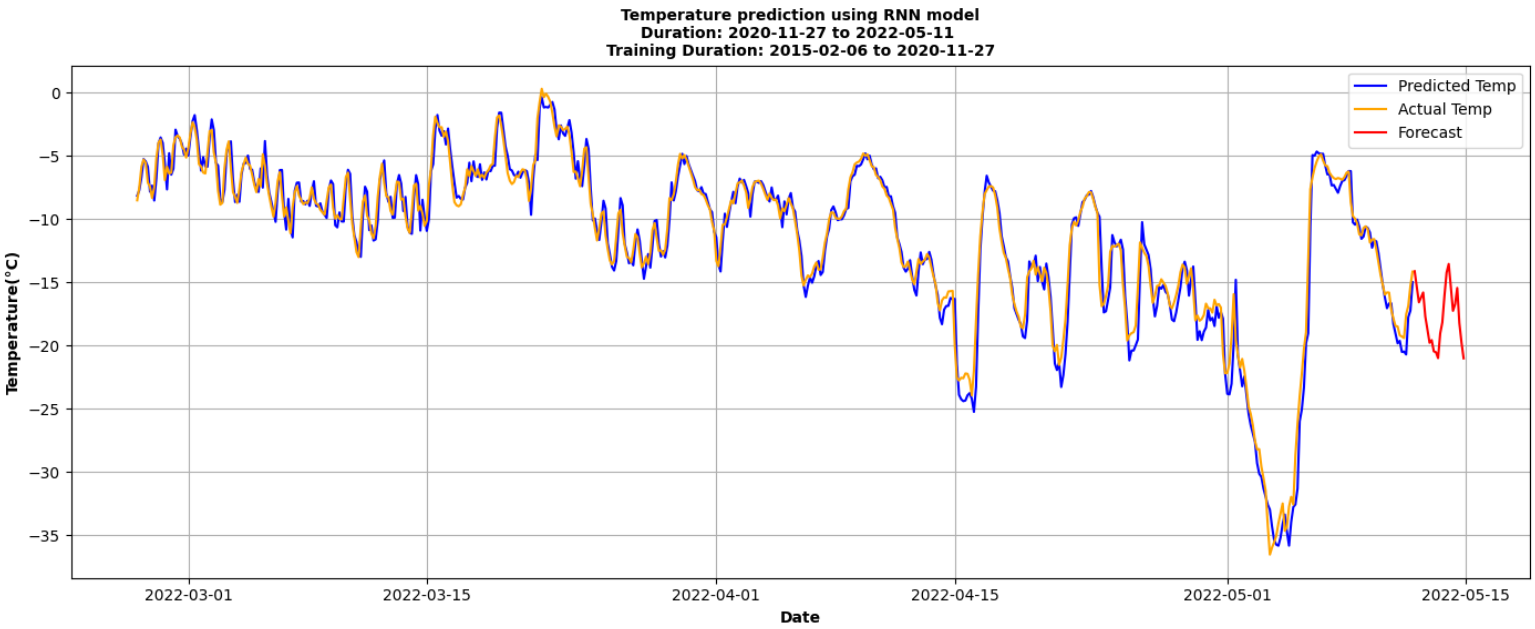
Code for RNN model:

```
model = Sequential()
model.add(SimpleRNN(32, input_shape=(168, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=200, batch_size=32)
```

Here we took one week's temperature as input.

Loss here is 0.304.

Below are the predictions using RNN:



Merits:

RNNs are well designed to manage sequential data, making them very dependable to use for time series data.

Demerits:

One of the problems with RNN is the vanishing gradient and the exploding gradient. Where weights to the network are either exceedingly small or large.

PREDICTION OF TEMPERATURE USING LONG SHORT-TERM MEMORY

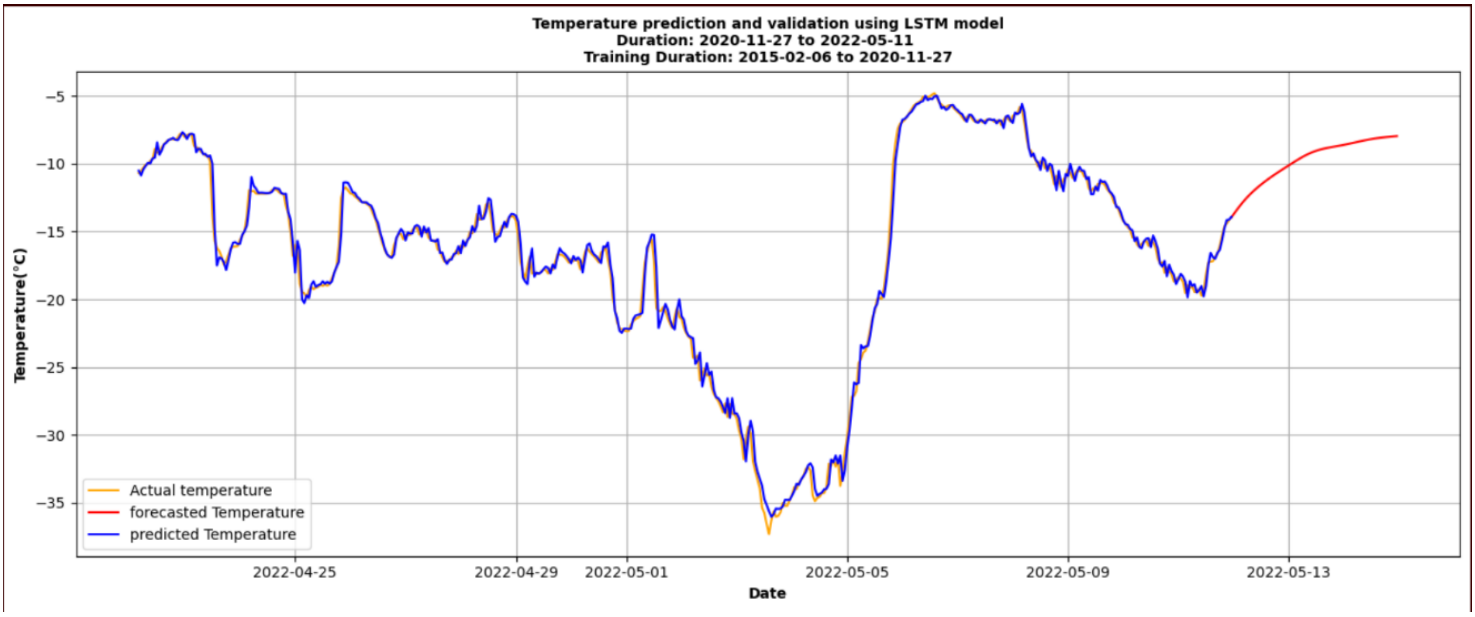
LSTM, like RNN is designed to manage sequential data, but their main distinction lies in their ability to capture long term dependencies. RNN suffers from the vanishing gradient problem, whereas LSTM was designed to overcome this by incorporating memory cells and additional gating mechanisms. Hence, LSTM are much better at handling long term dependencies than RNN.
Code for LSTM:

```
model = Sequential()
model.add(LSTM(units=64, input_shape=(X_train.shape[1], 1)))
model.add(Dense(units=1))
model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=200, batch_size=32)
```

Here, we used an LSTM layer with sixty-four memory units, and one output layer for prediction. The model training was time consuming.

Loss for LSTM is 0.23.

Below is the prediction for LSTM:



Merits:

It solves the vanishing gradient problem and can capture long term dependencies.

Demerits:

It needs a lot of computational power and is harder to interpret.

Results:

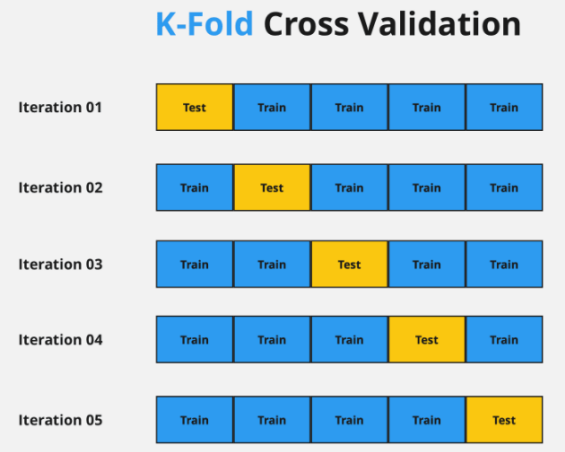
S.no	Model	Loss(RMSE)
1	ANN	0.41
2	RNN	0.304
3	CNN	2.34
4	LSTM	0.23

K FOLD CROSS VALIDATION

Oftentimes, our model might work well just on the specific dataset that we gave it and the specific training and test data. This might not be ideal as the model must be able to manage all types of datasets for it to be efficient. This is where K-Fold cross validation can help us as it measures how well the model can work in different datasets.

In k – fold cross validation we divide the data into k equal parts and train the model on one part and train it on the other k-1 parts. With this we can get k values of metrics (in this case RSME loss), which can be compared to find out the more consistent model.

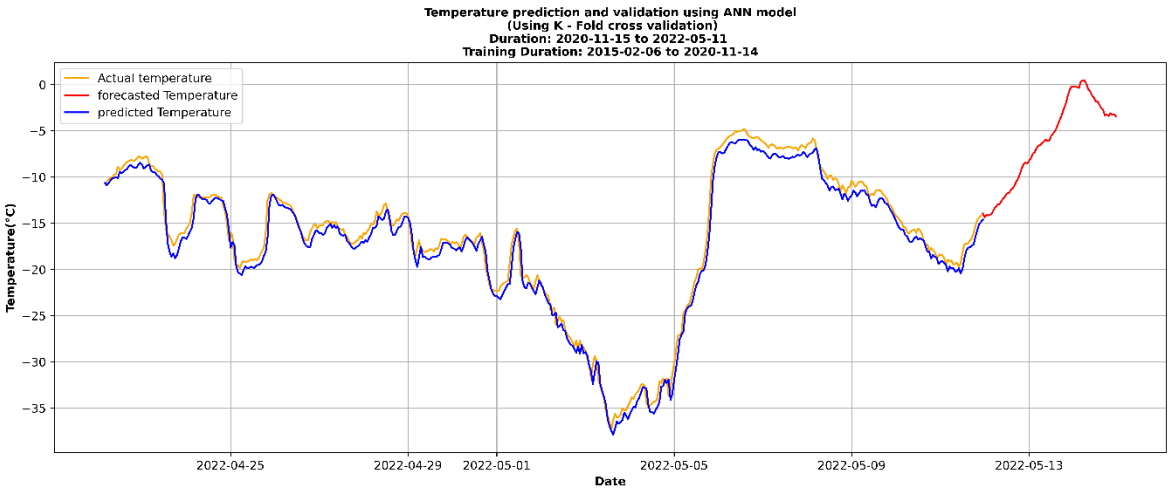
Here we took K as five.



Code for k – fold cross validation:

```
k = 5
kf = KFold(n_splits=k)
train_i = []
test_i = []
for train_index, test_index in kf.split(X):
train_i.append(train_index)
test_i.append(test_index)
loss = []
for i in range(k):
X_train = X[train_i[i]]
y_train = y[train_i[i]]
X_test = X[test_i[i]]
y_test = y[test_i[i]]
```

Using k fold cross validation, we can use the data for which there is minimum loss to train and get the prediction. Below is the prediction using best k-fold validation for ANN.



CLASSIFICATION AND PREDICTION OF BLIZZARD DATES

Given data on blizzard dates we used the parameters (Wind speed, Air pressure, Temperature, Humidity) to classify if the given data is a blizzard or not with the use of neural networks.

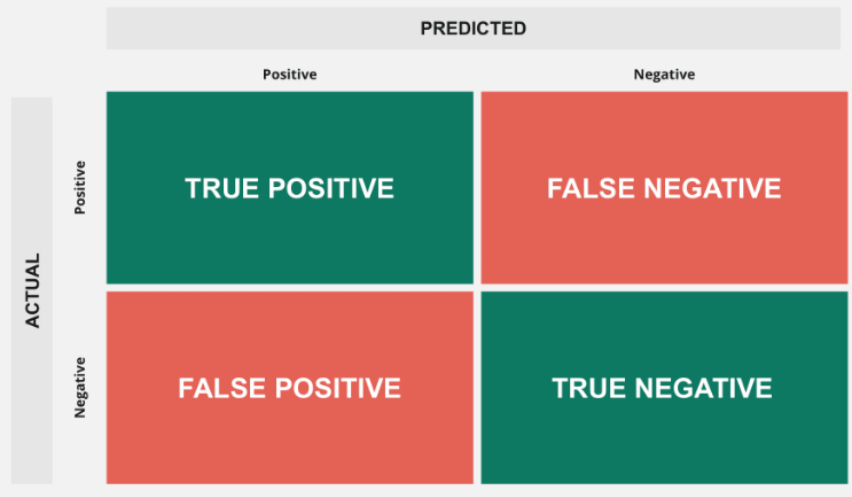
For analysis we used parameters such as accuracy and F1 score.

Accuracy measures the proportion of correctly classified instances made by the classification model. However, accuracy is not a good measure here since a sizable proportion of the data consist of non-blizzard cases and only a few are balanced. Hence, accuracy cannot be a reliable metric in such a hugely unbalanced data.

F1 score makes use of confusion matrix, which is given below, F1 score uses both precision and recall:

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)
Precision = TP / (TP + FP)

Recall = TP / (TP + FN)



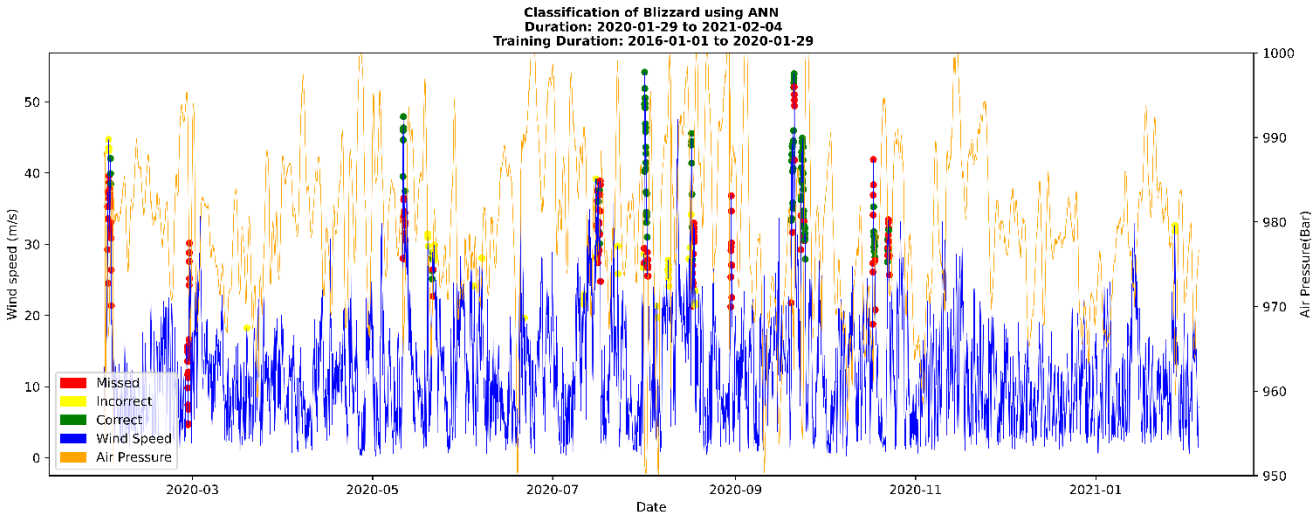
Using ANN:

Code for ANN classification:

```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train_scaled, y_train, epochs=200, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int)
y_test = np.array(y_test)
```

Here we have an accuracy of 0.977 and an F1 score of 0.556.

Below is the classification:



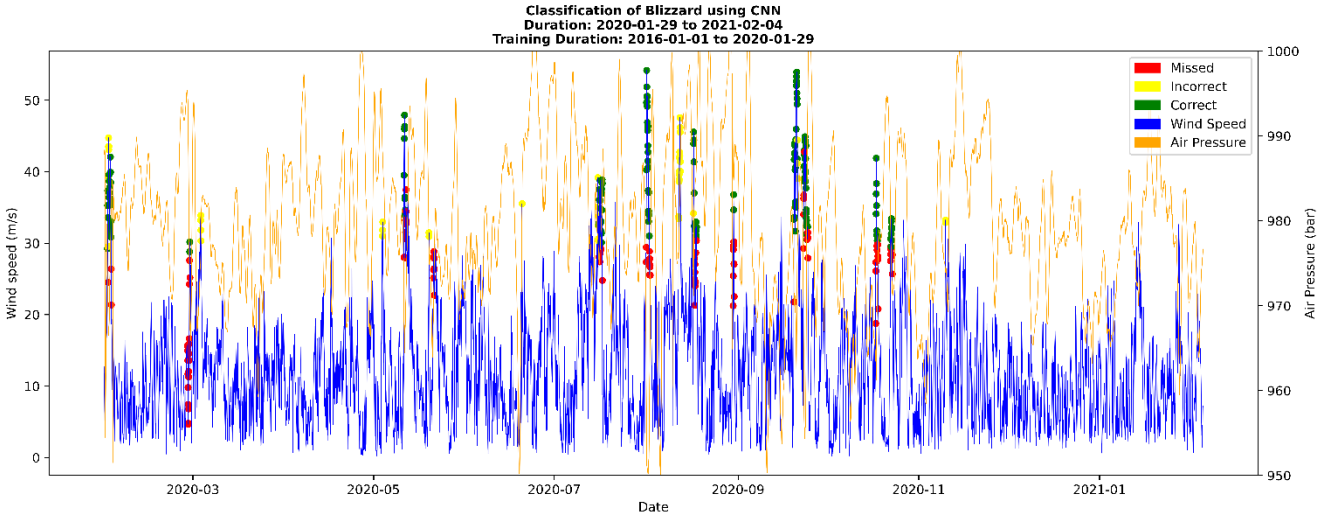
Using CNN:

Code for CNN:

```
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
model.fit(X_train_scaled, y_train, epochs=200, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int)
y_test = np.array(y_test)
```

We have an accuracy of 0.986 and f1 score of 0.586.

Below is the classification:



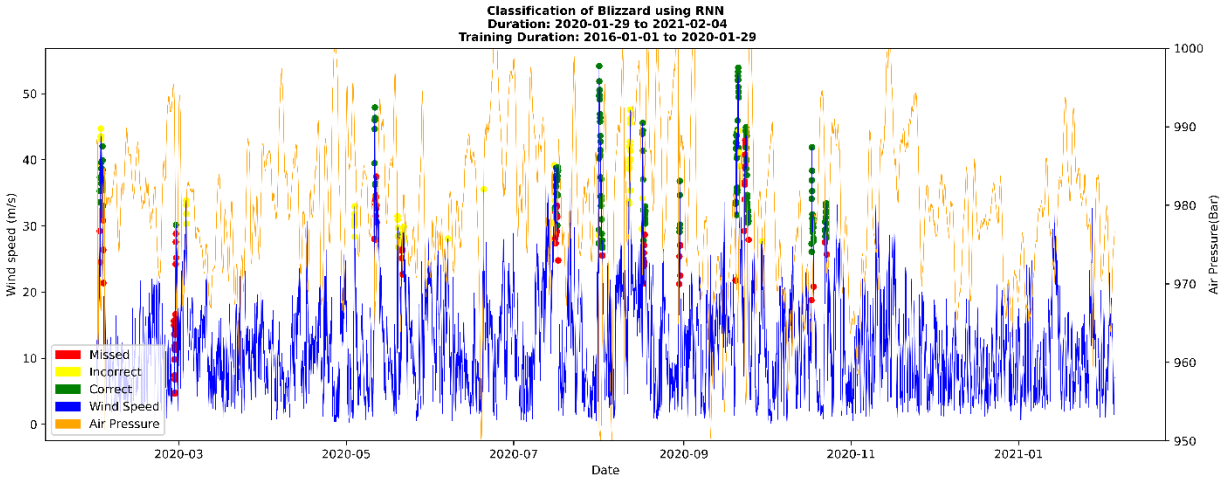
Using RNN:

Code for RNN:

```
model = Sequential()
model.add(SimpleRNN(units=64, activation='relu',input_shape=(time_steps, number_of_features)))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
model.fit(X_train_scaled, y_train, epochs=200, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int)
y_test = np.array(y_test)
```

Here the accuracy is 0.982 and F1 score is 0.707.

Below is the classification:



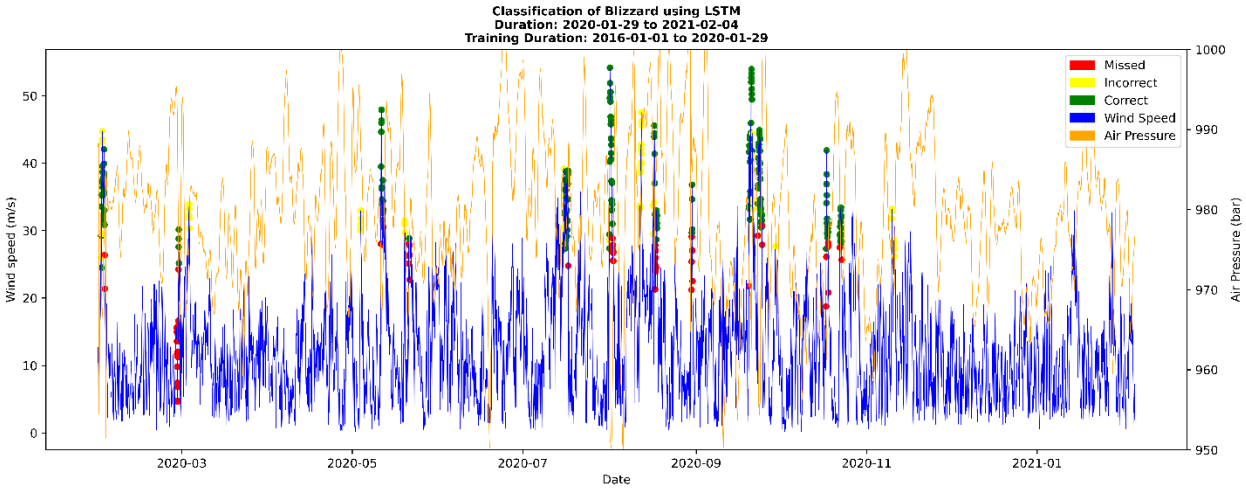
Using LSTM:

Code for LSTM:

```
model = Sequential()
model.add(LSTM(units=64, input_shape=(X_train_scaled.shape[1],X_train_scaled.shape[2])))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
model.fit(X_train_scaled, y_train, epochs=200, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int)
y_test = np.array(y_test)
```

Here accuracy is 0.976 and f1 score is 0.504.

Below is the classification:



Results:

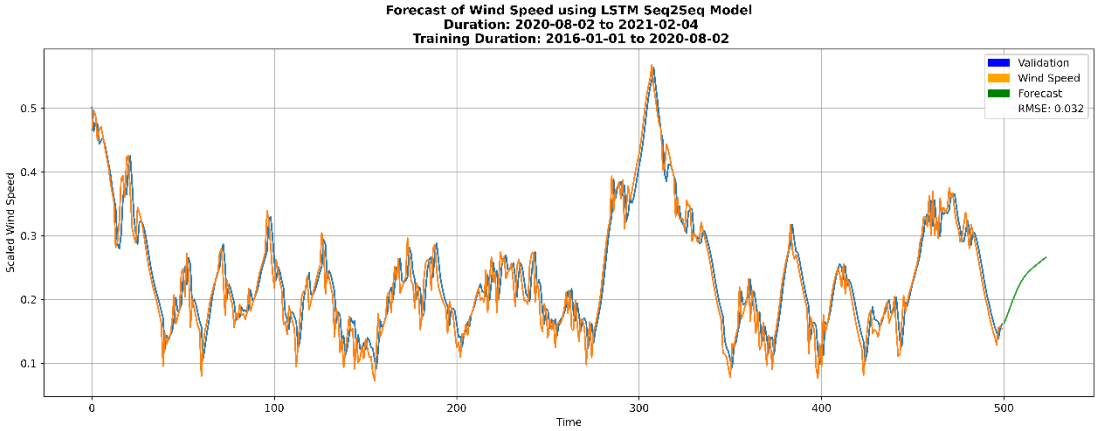
S.no	Model	Accuracy	F1 Score
1	ANN	0.977	0.556
2	CNN	0.986	0.586
3	RNN	0.982	0.707
4	LSTM	0.976	0.504

Here RNN has the highest F1 score and accuracy and is considered the best.

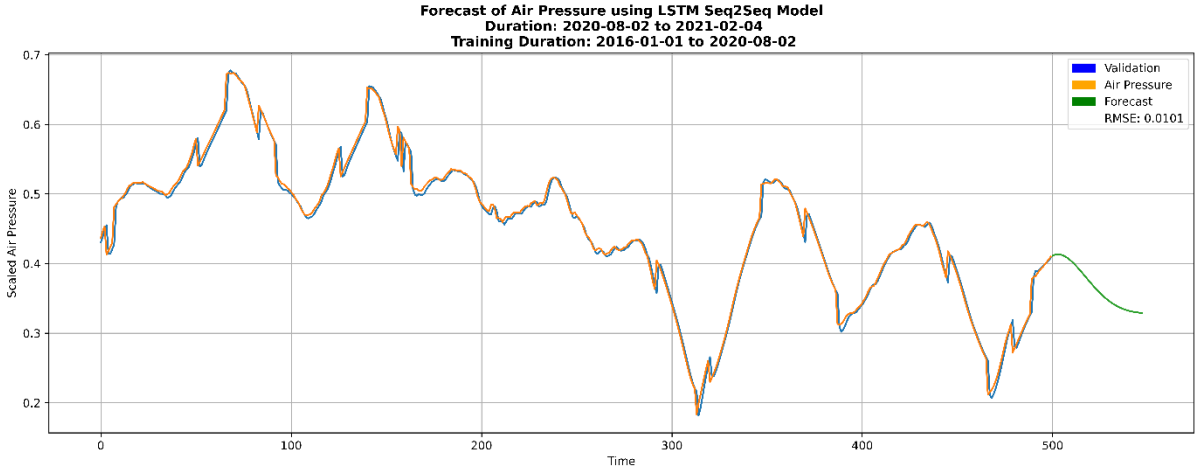
Prediction of blizzards:

For predicting blizzards, we need to predict wind speed, air pressure, temperature, and humidity, as predicting all of them leads to a lot of uncertainty we used only wind speed, air pressure and temperature for this prediction. We used the LSTM Seq2Seq model for prediction of wind speed and air pressure.

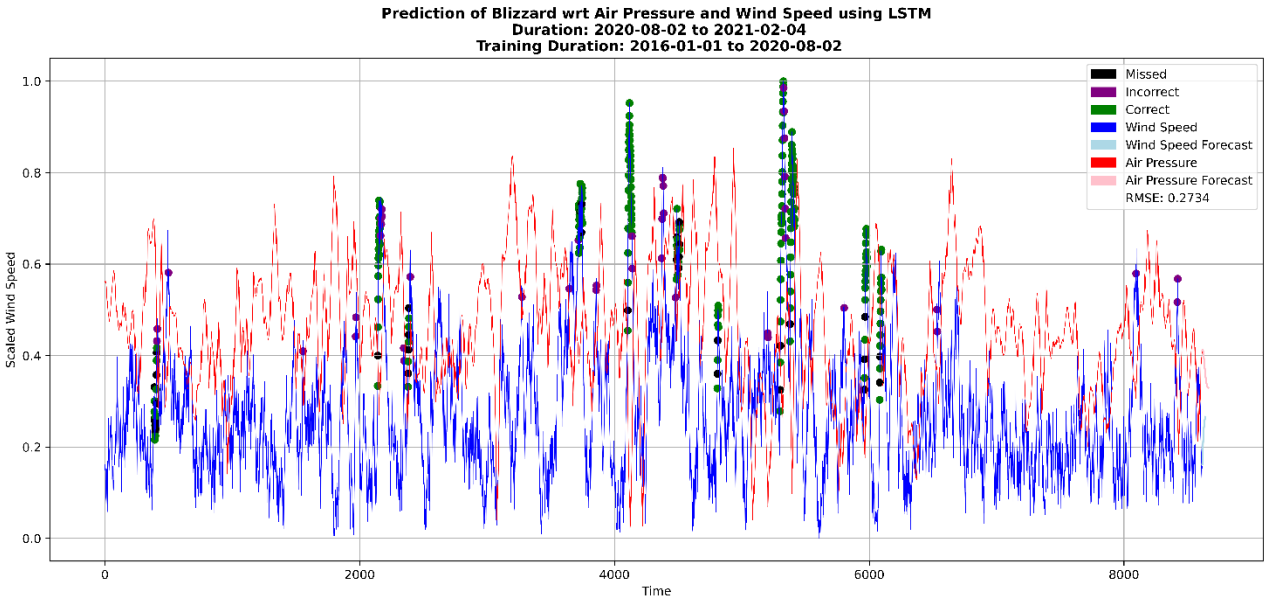
Prediction of wind speed:



Prediction of air pressure:



Prediction of blizzards:



CONCLUSION:

In conclusion, the analysis and visualization of weather data in polar regions using R and Python provided valuable insights into the trends and patterns of various weather parameters. By leveraging powerful statistical software like R and packages such as "openair," we were able to plot and visualize data related to temperature, wind direction, and wind speed.

With polar plots color-coded based on temperature, we observed that Antarctica experiences warmer temperatures during winter and colder temperatures during summer, aligning with its location in the southern hemisphere. This information can contribute to a better understanding of the unique climate dynamics in polar regions.

The analysis further extended to specific weather data collected from Bharathi station in Antarctica. By plotting graphs of air pressure and wind speed over time, we explored the relationship between these variables.

The hypothesis that higher air pressure corresponds to lower wind speed was examined, providing insights into the atmospheric conditions in the region. The visualization of wind speed, wind direction, and temperature in polar plots helped us identify patterns during blizzard weather conditions, enhancing our understanding of extreme weather phenomena.

Moreover, machine learning techniques were employed to predict future temperature values. By training the machine learning models on historical temperature data and using previous temperature values as inputs, we successfully predicted temperature values for the next three days. This predictive capability can be valuable for planning and decision-making in polar regions, where accurate weather forecasting is crucial.

Overall, the combination of data analysis, visualization, and machine learning techniques enabled a comprehensive exploration of weather trends and predictions in polar regions. The findings from this research can contribute to better understanding the unique weather patterns and conditions in these remote areas.

References:

1. openair: Tools for the Analysis of Air Pollution Data (r-project.org)
2. https://www.theactuary.com/2021/07/02/supervised-learning-techniques-claims-frequency-modelling