Machine Learning Models for Weather For Weather Forecasting

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Vishwajeet Singh (RA2111033010018)

Rithym Gulati (RA2111033010021)

Amrit k Tiwari (RA2111033010045)

Under the guidance of

Dr. N. Kanimozhi

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Deemed to be University u/s 3 of UGC Act, 1956

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **‘Machine learning Models for Weather Forecasting”** is the bonafide work of **Vishwajeet Singh (RA2111033010018), Rithym Gulati (RA2111033010021), Amrit K Tiwari (RA2111033010045)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other projectreport or dissertation on the basis of which a degree or award was conferred on an earlier occasionon this or any other candidate.

SIGNATURE

Dr. N. Kanimozhi
Assistant Professor
CINTEL

SIGNATURE

Head of the Department
Dr. Annie Uthra
Department of Computational Intelligence

ABSTRACT

Weather prediction is the application of science and technology to predict the state of the atmosphere for a given location. Here this system will predict weather based on parameters such as temperature, humidity, and wind. This system is a web application with effective graphical user interface. To predict the future's weather condition, the variation in the conditions in past years must be utilized.

The probability that it will match within the span of adjacent fortnight of previous year is very high. We have proposed the use of linear regression for weather prediction system with parameters such as temperature, humidity, and wind. It will predict weather based on previous record therefore this prediction will prove reliable. This system can be used in Air Traffic, Marine, Agriculture, Forestry, Military, and Navy etc.

The objective of our work is to design an effective weather prediction agent model using support vector machine and multiple linear regression or multivariate regression. Currently, there is lot of debate is going on around the world, in both the scientific and non-scientific communities about how the Earth's climate will change in the decades to come and what kind of impact it will have on the lives of future generations.

Scientific models that predict future climates offer the best hope for providing the information that will allow the world's policymakers to make informed decisions on the future of the Earth. Although there are models available to predict the weather but they mostly suffer from the problem of overfitting due to exposure to only one type of sample dataset and the other source of the weather prediction is the weather stations which log real time data and make predictions about the weather. Our approach is basically developing a machine learning model which boasts cross validation technique so that it can handle the condition of overfitting and underfitting well and does its task of predicting weather efficiently.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ML - Machine Learning

AI - Artificial Intelligence

IoT - Internet of Things

SVM - Support Vector Machine

ANN - Artificial Neural Network

CNN - Convolutional Neural Network

LSTM - Long Short-Term Memory

RF - Random Forest

DT - Decision Tree

KNN - K-Nearest Neighbours

MAE - Mean Absolute Error

MSE - Mean Squared Error

RMSE - Root Mean Squared Error

MAPE - Mean Absolute Percentage Error

RNN - Recurrent Neural Network

API - Application Programming Interface

GUI - Graphical User Interface

JSON - JavaScript Object Notation

CSV - Comma-Separated Values

GUI - Graphical User Interface

CHAPTER 1

INTRODUCTION

General circulation models work as tools for weather forecasting. There exists lots of assumption-based analysis on weather conditions presented us with the best prediction models like Numerical weather prediction, climatic variability assessment, weather Surveillances radar, predictions of Warming, etc.

The Scientists are still in working process of overcoming the limitations of computer models to improvise the accuracy rate of prediction through recent technologies of adding intelligence to machine. To add intelligence the system as human we have given a study platform called Artificial Neural networks, Machine learning, rule-based techniques where there exists ample impetus to study the weather occurrence and prediction.

Weather prediction is a convenient case for studying machine learning. By developing APIs for accessing available data from meteorological institutes and other weather stations, this gives access to an abundance of data. Weather data is something that most people can relate to in their daily life, but is also important for energy systems, flood prediction, etc. Good physical based meteorological models are available, which makes it easy to compare the quality of machine learning models.

Weather is important for most aspects of human life. Predicting weather is very useful. Humans have attempted to make predictions about the weather, many early religions used gods to explain the weather. Only relatively recently have humans developed reasonably accurate weather predictions. We decided to collect weather data and measured the accuracy of predictions made using linear regression.

The Weather prediction model designed by us would be of great use to the farmers and for normal being as well. This model basically uses historical weather data to predict the weather on a specific day of and year in the future. Initially the aim is to teach the model with large historical data set and then use it for weather prediction.

The observations include:
- Temperature - the measure of warmth or coldness
- Humidity - the amount of moisture in the atmosphere
- Precipitation - the amount of moisture (usually rain or snow) which falls on the ground
- Wind Speed - the speed at which air flows through the environment
- Wind Direction – the direction in which the wind is moving
- Pressure - the force the atmosphere applies on the environment

CHAPTER 2

LITERATURE SURVEY

Machine Learning is the study of computer algorithms that improve automatically through experience. Applications range from data mining programs that discover general rules in large data sets, to information filtering systems that automatically learn users' interests.

Machine Learning is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases. A major focus of Machine Learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too complex to describe generally in programming languages, so that in effect programs must automatically describe programs.

In recent years many successful machine learning applications have been developed, ranging from data-mining programs that learn to detect fraudulent credit card transactions, to information filtering systems that learn users' reading preferences, to autonomous vehicles that learn to drive. On public highways. At the same time, there have been important advances in the theory and algorithms that form the foundations of this field.
.

The poor performance results produced by statistical estimation models have flooded the estimation area for over the last decade. Their inability to handle categorical data, cope with missing data points, spread of data points and most importantly lack of reasoning capabilities has triggered an increase in the number of studies using non-traditional methods like machine learning techniques. The area of machine learning draws on concepts from diverse fields such as statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory.

There are two main types of Machine Learning algorithms. In this project, supervised learning is adopted here to build models from raw data and perform regression and classification.

Supervised learning:

Supervised Learning is a machine learning paradigm for acquiring the input output relationship information of a system based on a given set of paired input output training samples. As the output is regarded as the label of the input data or the supervision, an input-output training sample is also called labeled training data, or supervised data. Learning from Labeled Data, or Inductive Machine Learning.

The goal of supervised learning is to build an artificial system that can learn the mapping between the input and the output, and can predict the output of the system given new inputs. If the output takes a finite set of discrete values that indicate the class labels of the input, the learned mapping leads to the classification of the input data. If the output takes continuous values, it leads to a regression of the input. It deduces a function from training data that maps inputs to the expected outcomes. The output of the function can be a predicted continuous value (called regression), or a predicted class label from a discrete set for the input object (called classification). The goal of the supervised learner is to predict the value of the function for any valid input object from a number of training examples. The most widely used classifiers are the Neural Network (Multilayer perceptron), Support Vector Machines, k-nearest neighbor algorithm, Regression Analysis, Artificial neural networks and time series analysis.

.

Unsupervised learning:

Unsupervised learning studies how systems can learn tore present input patterns in a way that reflects the statistical structure of the overall collection of input patterns. By contrast with supervised learning or reinforcement learning, there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output.

Weather prediction has always been a challenging task due to the complex nature of atmospheric phenomena. Traditional meteorological models rely on physical equations to predict weather patterns. However, with the advancements in machine learning techniques, there has been a growing interest in using data-driven approaches for weather prediction.

1. Machine Learning Approaches for Weather Prediction:

- Gagne II, David J., et al. "Machine learning methods for empirical/statistical downscaling." In: Handbook of Climate Change Mitigation and Adaptation. Springer, Cham, 2017. pp. 1-25.
- Lary, David J., et al. "The application of machine learning for evaluating anthropogenic influence on extreme precipitation events: An attribution study." *Journal of Climate* 32.11 (2019): 3349-3370.

2. Time-Series Forecasting with Machine Learning:

- Malik, H., and H. G. Elmahdy. "Machine learning models for predicting weather parameters using meteorological data: A review." *Int. J. Eng. Technol.* 7 (2018): 24-29.
- Zhang, Guoqiang, et al. "Short-term wind speed forecasting based on machine learning methods: A review." *Renewable and Sustainable Energy Reviews* 75 (2017): 683-693.

3. Neural Network Approaches:

- Zhang, Yong, et al. "Short-term solar power forecasting using weather type classification based on artificial neural network." *Solar Energy* 127 (2016): 101-111.
- Karpatne, A., et al. "Theory-guided data science: A new paradigm for scientific discovery from data." *IEEE Transactions on Knowledge and Data Engineering* 29.10 (2017): 2318-2331.

4. Deep Learning Techniques:

- Shin, H., and S. Choi. "Short-term solar irradiance forecasting using deep learning." *IEEE Transactions on Industrial Informatics* 14.6 (2018): 2559-2567.
- Li, Y. "Wind speed forecasting model based on deep learning." *IOP Conference Series: Earth and Environmental Science* 313.4 (2019): 042034.

5. Ensemble Methods:

- Kesarwani, M., et al. "A review on forecasting techniques of wind speed in renewable energy system." *Renewable and Sustainable Energy Reviews* 39 (2014): 877-888.
- Wang, S., et al. "Short-term wind speed forecasting with data assimilation in the WRF-LETKF model using an ensemble Kalman filter approach." *Applied Energy* 222 (2018): 155-169.

The literature review suggests that machine learning and deep learning techniques have shown promising results in weather prediction. However, there is still a need for further research to improve the accuracy and reliability of these models.

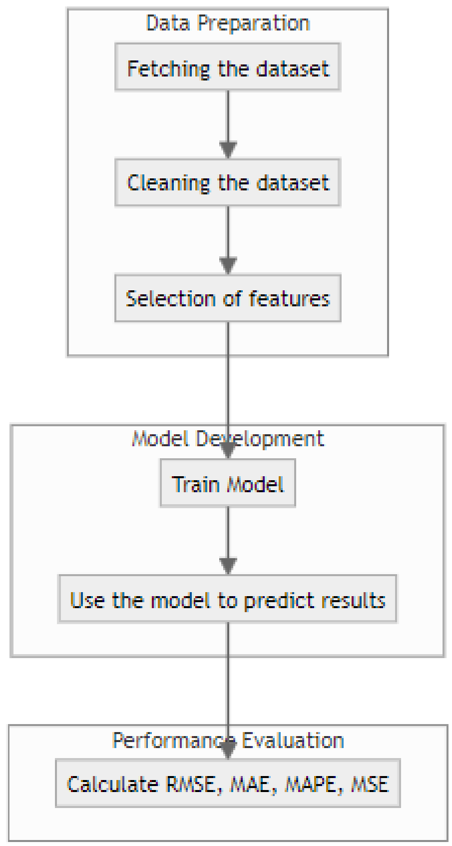
.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Architecture Diagram

The architecture of the proposed system can be represented by the following block diagram:



3.2 Description of Modules and Components

1. **Data Collection Module:**

- **Description**: This module is responsible for gathering weather-related data from various sources such as weather stations, satellites, and online databases.
- **Components**:
    - Web Scraping: Extracting data from online sources like weather websites.
    - API Integration: Utilizing weather APIs to fetch real-time weather data.
    - Data Preprocessing: Cleaning and formatting the collected data for further analysis.

2. **Feature Engineering Module:**

    - **Description**: This module involves selecting and transforming raw data into a format suitable for machine learning algorithms.
    - **Components**:
        - Feature Selection: Identifying relevant features such as temperature, humidity, wind speed, etc.
        - Feature Scaling: Normalizing or standardizing feature values to ensure uniformity.
        - Feature Encoding: Converting categorical features into numerical representations.

3. **Machine Learning Model Development Module:**

    - **Description**: This module focuses on building and training machine learning models to predict weather patterns.
    - **Components:**
        - **Model Selection**: Choosing appropriate machine learning algorithms such as regression, decision trees, or neural networks.
        - **Model Training:** Training the selected model using historical weather data.
        - **Model Evaluation:** Assessing the performance of the trained model using evaluation metrics like mean squared error, mean absolute error, etc.

4. **Model Deployment Module:**

    - **Description:** This module involves deploying the trained machine learning model to make predictions on new, unseen data.
    - **Components:**
        - Integration with Web Application: Integrating the model with a web-based interface for user interaction.
        - API Development: Developing APIs to allow other applications to access the prediction functionality.
        - Cloud Deployment: Deploying the model on cloud platforms such as AWS, Azure, or Google Cloud for scalability and accessibility.

5. **Post-Processing Module:**
    - **Description:** This module involves post-processing the model predictions to improve accuracy and usability.
    - **Components:**
        - Ensemble Methods: Combining predictions from multiple models to improve accuracy.
        - Error Analysis: Identifying and correcting prediction errors to enhance the reliability of the system.
        - Continuous Learning: Updating the model periodically with new data to adapt to changing weather patterns.

6. **Performance Monitoring and Logging Module:**

- **Description:** This module monitors the performance of the weather prediction system and logs important metrics for analysis and optimization.
- **Components:**
    - **Logging:** Recording system activities, errors, and user interactions for troubleshooting and auditing purposes.
    - **Performance Metrics:** Tracking system performance metrics such as prediction accuracy, response time, and resource utilization.
    - **Alerting System:** Notifying administrators about system failures or performance degradation in real-time.

Each of these modules and components plays a crucial role in developing a robust and accurate weather prediction system using machine learning

CHAPTER 4

METHODOLOGY

The development of the Weather Prediction system follows a comprehensive methodology encompassing various stages, each crucial for ensuring the system's effectiveness, accuracy, and compliance with regulatory standards and user requirements. This structured approach entails:

Methodological Steps

1. **Data Collection:**

    - Gather historical weather data from reliable sources such as NOAA, METAR, or other meteorological agencies.
    - Ensure the dataset includes relevant features such as temperature, humidity, wind speed, atmospheric pressure, etc.

2. **Data Preprocessing:**

    - Handle missing data: Impute missing values using techniques like mean, median, or interpolation.
    - Feature selection: Identify and select relevant features for the prediction model.
    - Data normalization: Scale the features to a similar range to prevent any feature from dominating the model training process.
    - Data splitting: Divide the dataset into training and testing sets.

3. **Feature Engineering:**

    - Create additional relevant features if necessary.
    - Extract temporal features like day of the week, month, season, etc., from the timestamp.

4. **Model Selection:**

    - Choose appropriate machine learning algorithms for weather prediction such as:
        - Random Forest Regression
        - Neural Prophet
        - Prophet

5. **Model Training:**

- Train the selected machine learning models using the training dataset.

6. **Model Evaluation:**

- Evaluate the performance of the trained models using appropriate evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), etc.
- Compare the performance of different models to select the best performing one.

7. **Hyperparameter Tuning:**

- Optimize the hyperparameters of the selected model to improve its performance.
- Use techniques like Grid Search or Random Search for hyperparameter tuning.

8. **Model Testing:**

- Test the performance of the final model using the testing dataset.

9. **Results Analysis:**

- Analyze the model's predictions against the actual weather observations.
- Visualize the model's performance using appropriate graphs and charts.

10. **Conclusion and Future Enhancements:**

- Summarize the findings of the project.
- Discuss the strengths and limitations of the model.
- Propose possible future enhancements to improve the model's performance.

These methodological steps should help you structure your project report on weather prediction using machine learning.

CHAPTER 5

CODING AND TESTING

Implementation using Prophet

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'historicalweatherdataforindiancities:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F2373708%2F4159658%2Fbundle%2Farchive.zip%

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue
```

```python
print('Data source import complete.')
```

[2] Python

```
Downloading historicalweatherdataforindiancities, 624464 bytes compressed
[==================================================] 624464 bytes downloaded
Downloaded and uncompressed: historicalweatherdataforindiancities
Data source import complete.
```

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Python

```
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/weather_Rourkela_2021_2022.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Lucknow_1990_2022.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/weather_Bhubhneshwar_1990_2022.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Rajasthan_1990_2022_Jodhpur.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Station_GeoLocation_Longitute_Latitude_Elevation_EPSG_4326.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Mumbai_1990_2022_Santacruz.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Delhi_NCR_1990_2022_Safdarjung.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Bangalore_1990_2022_BangaloreCity.csv
/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Chennai_1990_2022_Madras.csv
```

READING THE DATA INTO A DATAFRAME

```python
df = pd.read_csv('/kaggle/input/historicalweatherdataforindiancities/Temperature_And_Precipitation_Cities_IN/Bangalore_1990_2022_BangaloreCity.csv')
df.shape
```

[4] Python

```
(11894, 5)
```

```python
df.info()
```

[5] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11894 entries, 0 to 11893
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   time    11894 non-null  object
 1   tavg    11824 non-null  float64
 2   tmin    10505 non-null  float64
 3   tmax    11265 non-null  float64
 4   prcp    7274 non-null   float64
dtypes: float64(4), object(1)
memory usage: 464.7+ KB
```

```python
df.isna().sum()
```

[6] Python

```
time       0
tavg      70
tmin    1389
tmax     629
prcp    4620
dtype: int64
```

Mean and median value of all columns

```
df.mean(numeric_only=True)
```

[7] Python

```
tavg    23.840426
tmin    19.385131
tmax    29.932827
prcp     4.414119
dtype: float64
```

```
df.median(numeric_only=True)
```

[8] Python

```
tavg    23.5
tmin    19.8
tmax    29.5
prcp     0.0
dtype: float64
```

```
df.describe()
```

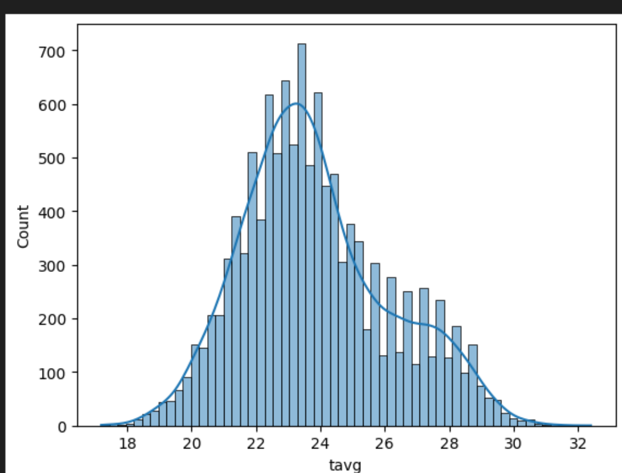
[9] Python

	tavg	tmin	tmax	prcp
count	11824.000000	10505.000000	11265.000000	7274.000000
mean	23.840426	19.385131	29.932827	4.414119
std	2.309899	2.367239	2.957866	12.994655
min	17.200000	9.300000	19.800000	0.000000
25%	22.300000	18.100000	27.900000	0.000000
50%	23.500000	19.800000	29.500000	0.000000
75%	25.200000	20.800000	32.000000	2.000000
max	32.400000	27.900000	39.200000	271.300000

```
sns.histplot(df['tavg'],kde=True)
```

[10] Python

```
<Axes: xlabel='tavg', ylabel='Count'>
```



Convert Time column from object/string type to datetime type

```python
df['time']=pd.to_datetime(df['time'],dayfirst=True)
```

[14] Python

```python
import datetime as dt
df['day'] = df['time'].dt.day
df['month'] = df['time'].dt.month
df['year'] = df['time'].dt.year
```

[12] Python

```python
df.head()
```

[13] Python

	time	tavg	tmin	tmax	prcp	day	month	year
0	1990-01-01	22.9	19.1	28.4	NaN	1	1	1990
1	1990-01-02	21.7	NaN	26.5	0.0	2	1	1990
2	1990-01-03	21.0	16.4	26.5	0.0	3	1	1990
3	1990-01-04	20.8	NaN	27.4	0.0	4	1	1990
4	1990-01-05	20.4	14.2	26.1	0.0	5	1	1990

```python
df.drop(['tmin','tmax','prcp'],axis=1,inplace=True)
```

[15] Python

```python
df.head()
```

[16] Python

	time	tavg	day	month	year
0	1990-01-01	22.9	1	1	1990
1	1990-01-02	21.7	2	1	1990
2	1990-01-03	21.0	3	1	1990
3	1990-01-04	20.8	4	1	1990
4	1990-01-05	20.4	5	1	1990

+ Code + Markdown

```python
df[df['tavg'].isnull()]
```

[17] Python

	time	tavg	day	month	year
145	1990-05-26	NaN	26	5	1990
146	1990-05-27	NaN	27	5	1990
147	1990-05-28	NaN	28	5	1990
148	1990-05-29	NaN	29	5	1990
149	1990-05-30	NaN	30	5	1990
...	...	...	...	...	...
4925	2003-06-27	NaN	27	6	2003
6348	2007-05-20	NaN	20	5	2007
6448	2007-08-28	NaN	28	8	2007
6503	2007-10-22	NaN	22	10	2007
7107	2009-06-17	NaN	17	6	2009

70 rows × 5 columns

```python
df['tavg'] = df['tavg'].fillna(df['tavg'].mean())
```
[18] Python

```python
df.isna().sum()
```
[19] Python

```
time     0
tavg     0
day      0
month    0
year     0
dtype: int64
```

```python
year_wise = pd.DataFrame(df.groupby(['year'])['tavg'].agg('mean').reset_index())
```
[20] Python

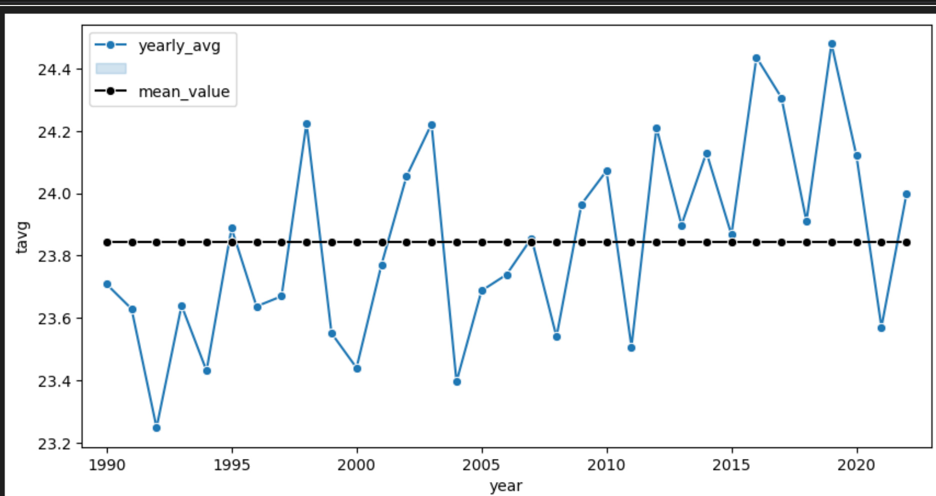
```python
year_wise = year_wise.set_index(year_wise['year'])
print(year_wise['tavg'].mean())
year_wise.head(2)
```
[21] Python

```
23.84255205877088
```

	year	tavg
year		
1990	1990	23.708400
1991	1991	23.629047

year wise plot

```python
plt.figure(figsize=(10,5))
plt.xlim([1989,2023])
sns.lineplot(x=year_wise.index,y=year_wise['tavg'],marker='o')
sns.lineplot(x =year_wise.index , y=[23.84255205877088]*len(year_wise.index),color='black',marker='o')
plt.legend(['yearly_avg',None,'mean_value'])
plt.show()
```
[22] Python




```python
df.head(5)
```
[23] Python

	time	tavg	day	month	year
0	1990-01-01	22.9	1	1	1990
1	1990-01-02	21.7	2	1	1990
2	1990-01-03	21.0	3	1	1990
3	1990-01-04	20.8	4	1	1990
4	1990-01-05	20.4	5	1	1990

```python
df_new = df[['time','tavg']]
df_new.columns=['ds','y']
df_new.tail(5)
```

[25] Python

	ds	y
11889	2022-07-21	23.7
11890	2022-07-22	23.2
11891	2022-07-23	23.1
11892	2022-07-24	22.8
11893	2022-07-25	24.1

```python
df_new[df_new['ds']=='2018-12-31']
```

[26] Python

	ds	y
10591	2018-12-31	20.9

```python
train = df_new[:10592]
test = df_new[10592:]
```

[27] Python

```python
import plotly.graph_objs as go
fig = go.Figure([

    go.Scatter(
        name='Train',
        x=df['time'],
        y=df['tavg'],
        mode='lines',
        marker=dict(color='blue'),
        line=dict(width=1),
        showlegend=True
    ),
    go.Scatter(
        name='Test',
        x=test['ds'],
        y=test['y'],
        mode='lines',
        marker=dict(color="orange"),
        line=dict(width=1),
        showlegend=True)])
fig.update_layout(
    yaxis_title='Temperature Avearage',
    title='Train and Test',
    hovermode="x"
)
fig.show()
```

USING PROPHET TO TRAIN THE MODEL

```python
from prophet import Prophet
model=Prophet()
```

[32] Python

```python
model.fit(train)
```

[33] Python

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpi4qoorb5/fbtv2lfd.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpi4qoorb5/ga5e6ci9.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=729', 'data', 'file=/tmp/tmpi4qoorb
13:03:35 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:03:38 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
<prophet.forecaster.Prophet at 0x7f329f3e0940>
```

[34]

```
test.shape
```

Python

```
(1302, 2)
```

[35]

```
test.tail(2)
```

Python

	ds	y
11892	2022-07-24	22.8
11893	2022-07-25	24.1

[36]

```
new_df= model.make_future_dataframe(periods=1302)
new_df.tail(5)
```

Python

	ds
11889	2022-07-21
11890	2022-07-22
11891	2022-07-23
11892	2022-07-24
11893	2022-07-25

[37]

```
result = model.predict(new_df)
result.tail(5)
```

Python

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly	weekly_lower	weekly_uppe
11889	2022-07-21	24.347189	22.124093	25.338141	23.809060	24.870521	-0.653857	-0.653857	-0.653857	0.016642	0.016642	0.01664
11890	2022-07-22	24.347292	21.979674	25.415691	23.808756	24.870765	-0.706714	-0.706714	-0.706714	-0.014123	-0.014123	-0.01412
11891	2022-07-23	24.347395	21.932783	25.272405	23.808452	24.871009	-0.735215	-0.735215	-0.735215	-0.022363	-0.022363	-0.02236
11892	2022-07-24	24.347498	21.886244	25.331740	23.808147	24.871253	-0.726307	-0.726307	-0.726307	0.004773	0.004773	0.00477
11893	2022-07-25	24.347601	21.896582	25.311575	23.807843	24.871498	-0.739030	-0.739030	-0.739030	0.008081	0.008081	0.00808

[38]

```
data = result[['ds','trend','yhat','yhat_lower','yhat_upper']]
data.columns=['date','trend','final_outcome(predicted)','final_outcome_lower_limit','final_outcome_upper_limit']
data.tail(5)
```

Python

	date	trend	final_outcome(predicted)	final_outcome_lower_limit	final_outcome_upper_limit
11889	2022-07-21	24.347189	23.693332	22.124093	25.338141
11890	2022-07-22	24.347292	23.640577	21.979674	25.415691
11891	2022-07-23	24.347395	23.612180	21.932783	25.272405
11892	2022-07-24	24.347498	23.621191	21.886244	25.331740
11893	2022-07-25	24.347601	23.608571	21.896582	25.311575

```python
data.shape
```

[39] Python

```
(11894, 5)
```

```python
import plotly.graph_objs as go
fig = go.Figure([

    go.Scatter(
        name='Test_Actual',
        x=test['ds'],
        y=test['y'],
        mode='lines',
        marker=dict(color='orange'),
        line=dict(width=3),
        showlegend=True
    ),
    go.Scatter(
        name='Test_predicted',
        x=data[10592:]['date'],
        y=data[10592:]['final_outcome(predicted)'],
        mode='lines',
        marker=dict(color="red"),
        line=dict(width=1),
        showlegend=True),
    go.Scatter(
        name='Test_predicted_Lower',
        x=data[10592:]['date'],
        y=data[10592:]['final_outcome_lower_limit'],
        mode='lines',
        marker=dict(color="#444"),
        line=dict(width=1),
        showlegend=True),
    go.Scatter(
        name='Test_predicted_Upper',
        x=data[10592:]['date'],
        y=data[10592:]['final_outcome_upper_limit'],
        mode='lines',
        marker=dict(color="#444"),
        line=dict(width=1),
        showlegend=True,
        fillcolor = 'rgba(68, 68, 68, 0.3)',
        fill = 'tonexty'),
])
fig.update_layout(
    yaxis_title='Temperature Avearage',
    title='Actual vs Predicteed',
    hovermode="x"
)
fig.show()
```

[40] Python

```python
from prophet.plot import plot_plotly
from plotly.offline import iplot
fig=plot_plotly(model,result)
iplot(fig)
```

[41] Python

```python
test['predicted'] = data[10592:]['final_outcome(predicted)']
```

[42] Python

```
<ipython-input-42-a6c6c183fcd4>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```python
test['mse'] = (test['y']-test['predicted'])**2
```

[43] Python

```
<ipython-input-43-37b4b31eff52>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
test['smape'] = abs(test['y']-test['predicted'])/(abs(test['y']+test['predicted']))/2
```
[44] Python

```
<ipython-input-44-5f7ae0c62ee3>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
test['mape'] = abs((test['y']-test['predicted'])/test['y'])
```
[45] Python

```
<ipython-input-45-b62f9bce4cb2>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
print('MSE  = ',test['mse'].mean())
print('RMSE  = ',test['mse'].mean()**0.5)
print('MAPE = ',test['mape'].mean()*100)
print('SMAPE = ',test['smape'].mean()*100)
```
[46] Python

```
MSE  =  1.63166669579582
RMSE  =  1.2773670951593437
MAPE =  4.173807215452403
SMAPE =  1.0220289824104003
```

Implementation using Neural Prophet

```
!pip install neuralprophet
```
[1] Python

```
Requirement already satisfied: neuralprophet in c:\python312\lib\site-packages (0.8.0)
Requirement already satisfied: captum>=0.6.0 in c:\python312\lib\site-packages (from neuralprophet) (0.7.0)
Requirement already satisfied: holidays>=0.41 in c:\python312\lib\site-packages (from neuralprophet) (0.47)
Requirement already satisfied: matplotlib<4.0.0,>=3.5.3 in c:\python312\lib\site-packages (from neuralprophet) (3.8.4)
Requirement already satisfied: nbformat<6.0.0,>=5.8.0 in c:\python312\lib\site-packages (from neuralprophet) (5.10.3)
Requirement already satisfied: numpy<2.0.0,>=1.25.0 in c:\python312\lib\site-packages (from neuralprophet) (1.26.4)
Requirement already satisfied: pandas<3.0.0,>=2.0.0 in c:\python312\lib\site-packages (from neuralprophet) (2.2.2)
Requirement already satisfied: plotly<6.0.0,>=5.13.1 in c:\python312\lib\site-packages (from neuralprophet) (5.22.0)
Requirement already satisfied: pytorch-lightning<2.0.0,>=1.9.4 in c:\python312\lib\site-packages (from neuralprophet) (1.9.5)
Requirement already satisfied: tensorboard<3.0.0,>=2.11.2 in c:\python312\lib\site-packages (from neuralprophet) (2.16.2)
Requirement already satisfied: torch<3.0.0,>=2.0.0 in c:\python312\lib\site-packages (from neuralprophet) (2.3.0)
Requirement already satisfied: torchmetrics<2.0.0,>=1.0.0 in c:\python312\lib\site-packages (from neuralprophet) (1.3.2)
Requirement already satisfied: typing-extensions<5.0.0,>=4.5.0 in c:\python312\lib\site-packages (from neuralprophet) (4.11.0)
Requirement already satisfied: tqdm in c:\python312\lib\site-packages (from captum>=0.6.0->neuralprophet) (4.66.4)
Requirement already satisfied: python-dateutil in c:\users\amrit kumar tiwari\appdata\roaming\python\python312\site-packages (from holidays>=0.41->neuralprophet)
Requirement already satisfied: contourpy>=1.0.1 in c:\python312\lib\site-packages (from matplotlib<4.0.0,>=3.5.3->neuralprophet) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\python312\lib\site-packages (from matplotlib<4.0.0,>=3.5.3->neuralprophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\python312\lib\site-packages (from matplotlib<4.0.0,>=3.5.3->neuralprophet) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\python312\lib\site-packages (from matplotlib<4.0.0,>=3.5.3->neuralprophet) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\users\amrit kumar tiwari\appdata\roaming\python\python312\site-packages (from matplotlib<4.0.0,>=3.5.3->neura
Requirement already satisfied: pillow>=8 in c:\python312\lib\site-packages (from matplotlib<4.0.0,>=3.5.3->neuralprophet) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\python312\lib\site-packages (from matplotlib<4.0.0,>=3.5.3->neuralprophet) (3.1.2)
Requirement already satisfied: fastjsonschema in c:\python312\lib\site-packages (from nbformat<6.0.0,>=5.8.0->neuralprophet) (2.19.1)
Requirement already satisfied: jupyter-core in c:\users\amrit kumar tiwari\appdata\roaming\python\python312\site-packages (from nbformat<6.0.0,>=5.8.0->neuralpropl
...
Requirement already satisfied: frozenlist>=1.1.1 in c:\python312\lib\site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning<2.0
Requirement already satisfied: multidict<7.0,>=4.5 in c:\python312\lib\site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning<2
Requirement already satisfied: yarl<2.0,>=1.0 in c:\python312\lib\site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning<2.0.0,
Requirement already satisfied: idna>=2.0 in c:\python312\lib\site-packages (from yarl<2.0,>=1.0->aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lighti
```

```python
import pandas as pd
from neuralprophet import NeuralProphet
from matplotlib import pyplot as plt
import pickle
```
[2] Python

```python
# 1. Read in Data and Process Dates
```
[ ] Python

```python
df = pd.read_csv('Bangalore_1990_2022_weather.csv')
df.head()
```
[3] Python

	time	tavg	tmin	tmax	prcp
0	01-01-1990	22.9	19.1	28.4	NaN
1	02-01-1990	21.7	NaN	26.5	0.0
2	03-01-1990	21.0	16.4	26.5	0.0
3	04-01-1990	20.8	NaN	27.4	0.0
4	05-01-1990	20.4	14.2	26.1	0.0

```python
df.dtypes
```
[4] Python

```
time      object
tavg     float64
tmin     float64
tmax     float64
prcp     float64
dtype: object
```

```python
df['time'] = pd.to_datetime(df['time'], format='%d-%m-%Y')
df.head()
```
[5] Python

	time	tavg	tmin	tmax	prcp
0	1990-01-01	22.9	19.1	28.4	NaN
1	1990-01-02	21.7	NaN	26.5	0.0
2	1990-01-03	21.0	16.4	26.5	0.0
3	1990-01-04	20.8	NaN	27.4	0.0
4	1990-01-05	20.4	14.2	26.1	0.0

+ Code + Markdown

```python
dt = df[['time','tavg']]
dt.dropna(inplace=True)
dt.columns = ['ds','y']
dt.head()
```
[21] Python

```
WARNING - (py.warnings._showwarnmsg) - C:\Users\amrit kumar tiwari\AppData\Local\Temp\ipykernel_27156\959052012.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dt.dropna(inplace=True)
```

	ds	y
0	1990-01-01	22.9
1	1990-01-02	21.7
2	1990-01-03	21.0
3	1990-01-04	20.8
4	1990-01-05	20.4

+ Code + Markdown

```
dt.mean(numeric_only=True)
```

[22] Python

```
y    23.840426
dtype: float64
```

```
dt.median(numeric_only=True)
```

[23] Python

```
y    23.5
dtype: float64
```

```
dt.isna().sum()
```

[24] Python

```
ds    0
y     0
dtype: int64
```

```
dt.describe()
```

[25] Python

	ds	y
count	11824	11824.000000
mean	2006-05-07 09:58:12.828146176	23.840426
min	1990-01-01 00:00:00	17.200000
25%	1998-03-31 18:00:00	22.300000
50%	2006-05-14 12:00:00	23.500000
75%	2014-06-21 06:00:00	25.200000
max	2022-07-25 00:00:00	32.400000
std	NaN	2.309899

Source Control (Ctrl+Shift+G)

```
df_neural_prophet = dt.rename(columns={'time': 'ds', 'tavg': 'y'})
```

[26] Python

```
dt
```

[27] Python

	ds	y
0	1990-01-01	22.9
1	1990-01-02	21.7
2	1990-01-03	21.0
3	1990-01-04	20.8
4	1990-01-05	20.4
...	...	...
11889	2022-07-21	23.7
11890	2022-07-22	23.2
11891	2022-07-23	23.1
11892	2022-07-24	22.8
11893	2022-07-25	24.1

11824 rows × 2 columns

Run and Debug (Ctrl+Shift+D)

```
# 2. Train Model
```

[ ] Python

```
m = NeuralProphet()
```

[28] Python

```
model = m.fit(dt, freq='D', epochs=1000)
```

[31] Python

```
# 3. Forecast Away
```

```
future = m.make_future_dataframe(dt, periods=600)
forecast = m.predict(future)
forecast.head()
```

	ds	y	yhat1	trend	season_yearly	season_weekly
0	2022-07-26	None	23.139803	23.814678	-0.677623	0.002748
1	2022-07-27	None	23.142429	23.814539	-0.689498	0.017389
2	2022-07-28	None	23.127274	23.814400	-0.701051	0.013927
3	2022-07-29	None	23.082314	23.814260	-0.712271	-0.019677
4	2022-07-30	None	23.062611	23.814119	-0.723064	-0.028445

```
plt2 = m.plot_components(forecast)
```

```
plt2 = m.plot_components(forecast)
```

```
forecast.tail()
```

	ds	y	yhat1	trend	season_yearly	season_weekly
595	2024-03-12	None	26.081738	23.731586	2.347395	0.002756
596	2024-03-13	None	26.178753	23.731447	2.429944	0.017360
597	2024-03-14	None	26.255264	23.731308	2.510024	0.013932
598	2024-03-15	None	26.299349	23.731169	2.587835	-0.019654
599	2024-03-16	None	26.365784	23.731030	2.663156	-0.028401

```
import numpy as np
y_true = [26.081738, 26.178753, 26.255264, 26.299349, 26.365784]

# Predicted values
y_pred = [23.731586, 23.731447, 23.731308, 23.731169, 23.731030]

# Calculate mean squared error
mse = np.mean((np.array(y_true) - np.array(y_pred))**2)

# Calculate root mean squared error
rmse = np.sqrt(mse)

y_true = [26.081738, 26.178753, 26.255264, 26.299349, 26.365784]

# Predicted values
y_pred = [23.731586, 23.731447, 23.731308, 23.731169, 23.731030]

# Calculate absolute percentage errors
ape = np.abs((np.array(y_true) - np.array(y_pred)) / np.array(y_true))

# Calculate mean of absolute percentage errors
mape = np.mean(ape) * 100

sape = np.abs(np.array(y_true) - np.array(y_pred)) / (np.abs(np.array(y_true)) + np.abs(np.array(y_pred)))

# Calculate mean of absolute percentage errors
smape = np.mean(ape) * 100
```

```
print("MSE :", mse)
print("RMSE :", rmse)
print("MAPE :",mape)
print("SMAPE :",smape)
```

```
MSE : 6.2840704247184025
RMSE : 2.506804823818241
MAPE : 9.546114023528022
SMAPE : 9.546114023528022
```

Implementation using Multiple Linear Regression, Decision Tree Regression, Random Forest Regression

Importing Needed Packages

```python
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing

%matplotlib inline
```

Python

Reading CSV file as weather_df and making date_time column as index of dataframe

```python
weather_df = pd.read_csv('kanpur.csv', parse_dates=['date_time'], index_col='date_time')
weather_df.head(5)
```

Python

date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	uvIndex.1	moon_illumination	moonrise	moonset	sunrise	sunset	DewPointC	FeelsLikeC	HeatIndexC
2009-01-01 00:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	05:28 PM	2	11	12
2009-01-01 01:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	05:28 PM	3	12	13
2009-01-01 02:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	05:28 PM	4	12	13
2009-01-01 03:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	05:28 PM	5	12	13
2009-01-01 04:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	05:28 PM	5	14	14

Checking columns in our dataframe

```python
weather_df.columns
```

Python

```
Index(['maxtempC', 'mintempC', 'totalSnow_cm', 'sunHour', 'uvIndex',
       'uvIndex.1', 'moon_illumination', 'moonrise', 'moonset', 'sunrise',
       'sunset', 'DewPointC', 'FeelsLikeC', 'HeatIndexC', 'WindChillC',
       'WindGustKmph', 'cloudcover', 'humidity', 'precipMM', 'pressure',
       'tempC', 'visibility', 'winddirDegree', 'windspeedKmph'],
      dtype='object')
```

Now shape

```python
weather_df.shape
```

Python

```
(96432, 24)
```

```python
weather_df.describe()
```

	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	uvIndex.1	moon_illumination	DewPointC	FeelsLikeC	HeatIndexC	WindChillC	Wind
count	96432.000000	96432.000000	96432.0	96432.000000	96432.000000	96432.000000	96432.000000	96432.000000	96432.000000	96432.00000	96432.000000	96
mean	33.400199	22.374564	0.0	11.037805	6.877053	4.465012	46.094077	13.230629	30.735783	30.86884	29.088384	
std	6.994211	7.635253	0.0	2.152973	1.551294	3.414374	31.249725	8.053778	9.320398	9.17754	8.051981	
min	15.000000	3.000000	0.0	4.000000	3.000000	1.000000	0.000000	-14.000000	4.000000	7.00000	4.000000	
25%	28.000000	16.000000	0.0	8.700000	6.000000	1.000000	18.000000	7.000000	24.000000	25.00000	24.000000	
50%	34.000000	24.000000	0.0	11.600000	7.000000	5.000000	46.000000	12.000000	31.000000	31.00000	29.000000	
75%	38.000000	28.000000	0.0	13.000000	8.000000	8.000000	73.000000	21.000000	38.000000	38.00000	35.000000	
max	51.000000	39.000000	0.0	13.900000	11.000000	11.000000	100.000000	31.000000	65.000000	65.00000	54.000000	

Checking is there any null values in dataset

```python
weather_df.isnull().any()
```

```
maxtempC             False
mintempC             False
totalSnow_cm         False
sunHour              False
uvIndex              False
uvIndex.1            False
moon_illumination    False
moonrise             False
moonset              False
sunrise              False
sunset               False
DewPointC            False
FeelsLikeC           False
HeatIndexC           False
WindChillC           False
WindGustKmph         False
cloudcover           False
humidity             False
precipMM             False
pressure             False
tempC                False
visibility           False
winddirDegree        False
windspeedKmph        False
dtype: bool
```

Now lets separate the feature (i.e. temperature) to be predicted from the rest of the featured. weather_x stores the rest of the dataset while weather_y has temperature column.

```python
weather_df_num=weather_df.loc[:,['maxtempC','mintempC','cloudcover','humidity','tempC', 'sunHour','HeatIndexC', 'precipMM', 'pressure','windspeedKmph']]
weather_df_num.head()
```

	maxtempC	mintempC	cloudcover	humidity	tempC	sunHour	HeatIndexC	precipMM	pressure	windspeedKmph
date_time										
2009-01-01 00:00:00	24	10	17	50	11	8.7	12	0.0	1015	10
2009-01-01 01:00:00	24	10	11	52	11	8.7	13	0.0	1015	11
2009-01-01 02:00:00	24	10	6	55	11	8.7	13	0.0	1015	11
2009-01-01 03:00:00	24	10	0	57	10	8.7	13	0.0	1015	12
2009-01-01 04:00:00	24	10	0	54	11	8.7	14	0.0	1016	11

Source Control (Ctrl+Shift+G)

Shape of new dataframe

```python
weather_df_num.shape
```

Python

```
(96432, 10)
```

Columns in new dataframe

```python
weather_df_num.columns
```

Python

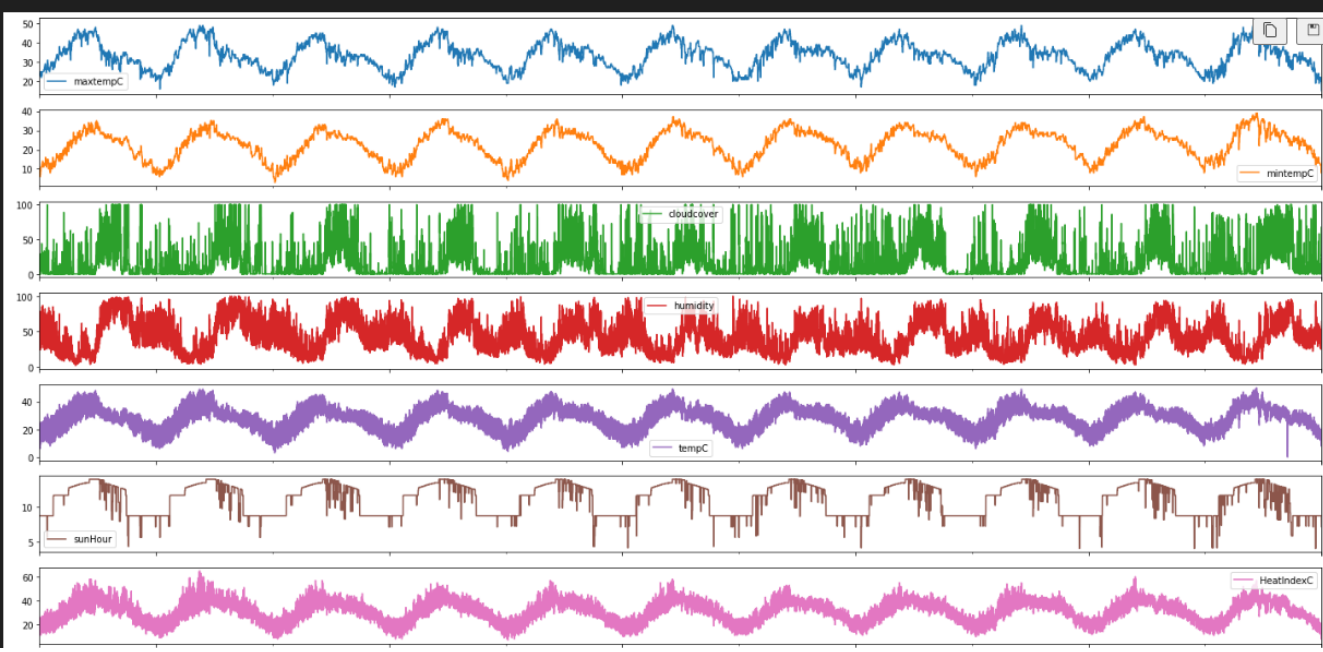
```
Index(['maxtempC', 'mintempC', 'cloudcover', 'humidity', 'tempC', 'sunHour',
       'HeatIndexC', 'precipMM', 'pressure', 'windspeedKmph'],
      dtype='object')
```

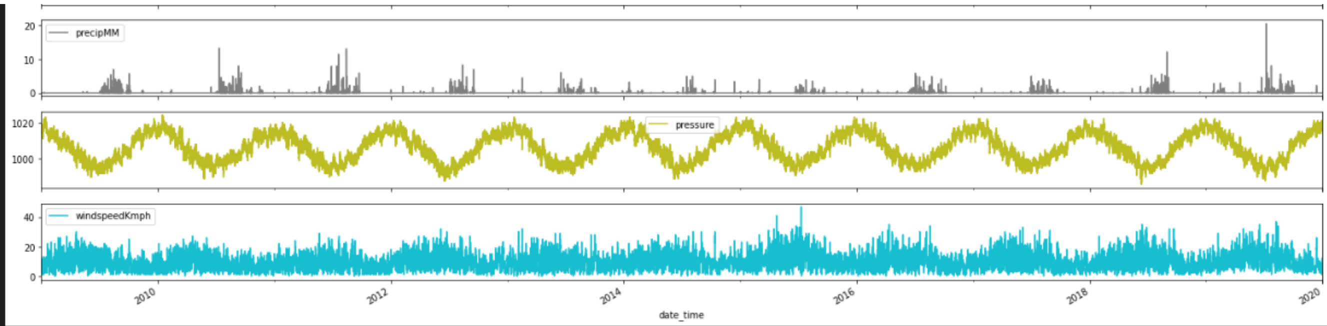
Ploting all the column values

```python
weather_df_num.plot(subplots=True, figsize=(25,20))
```

Python

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e2f1f10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e02e6d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e2ee3d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8dd8ded0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8db37310>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e2c3710>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e252b90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e288ed0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e288f10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e1cc450>],
      dtype=object)
```





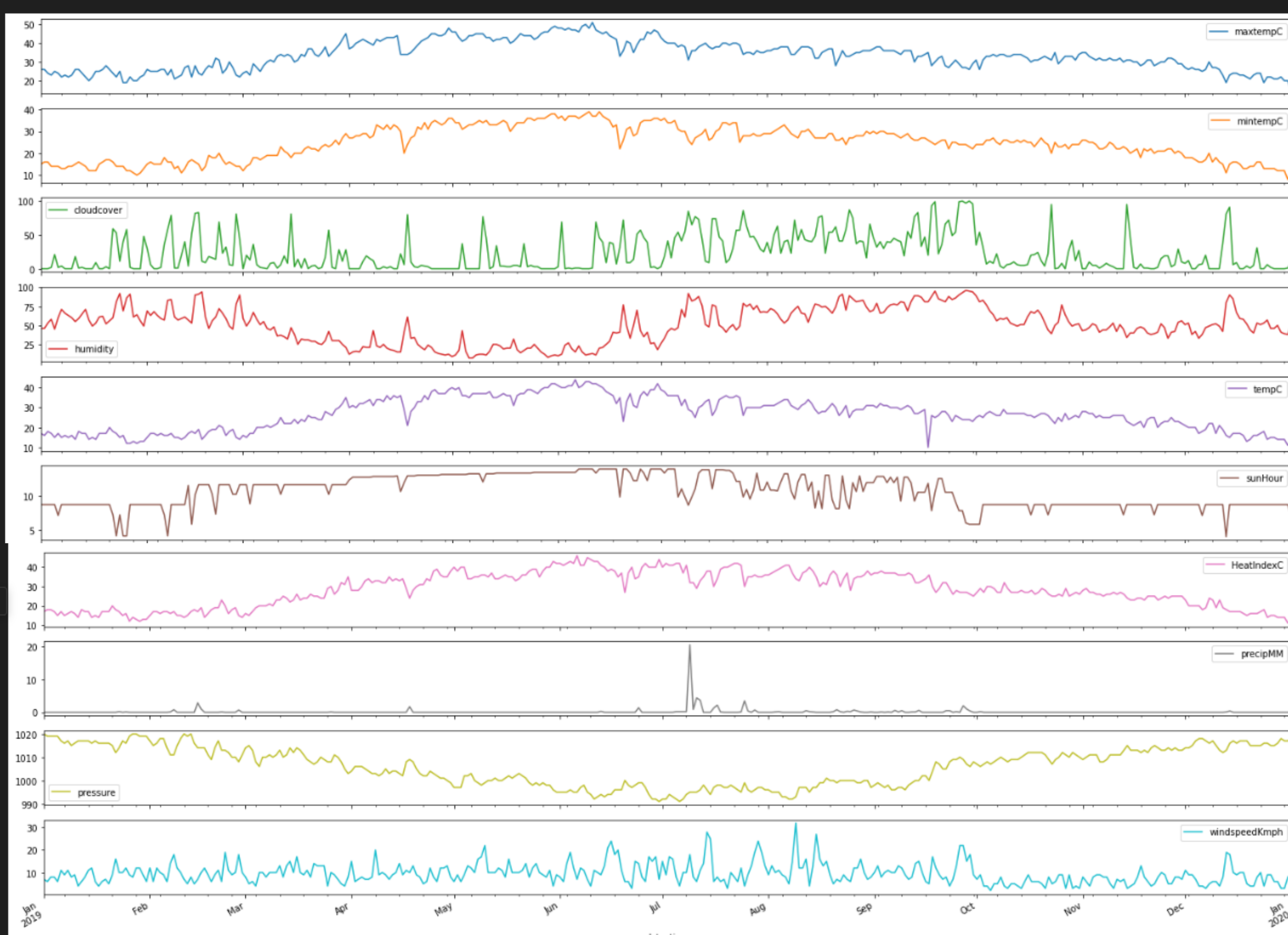
Source Control (Ctrl+Shift+G)

Ploting all the column values for 1 year

```python
weather_df_num['2019':'2020'].resample('D').fillna(method='pad').plot(subplots=True, figsize=(25,20))
```

Python

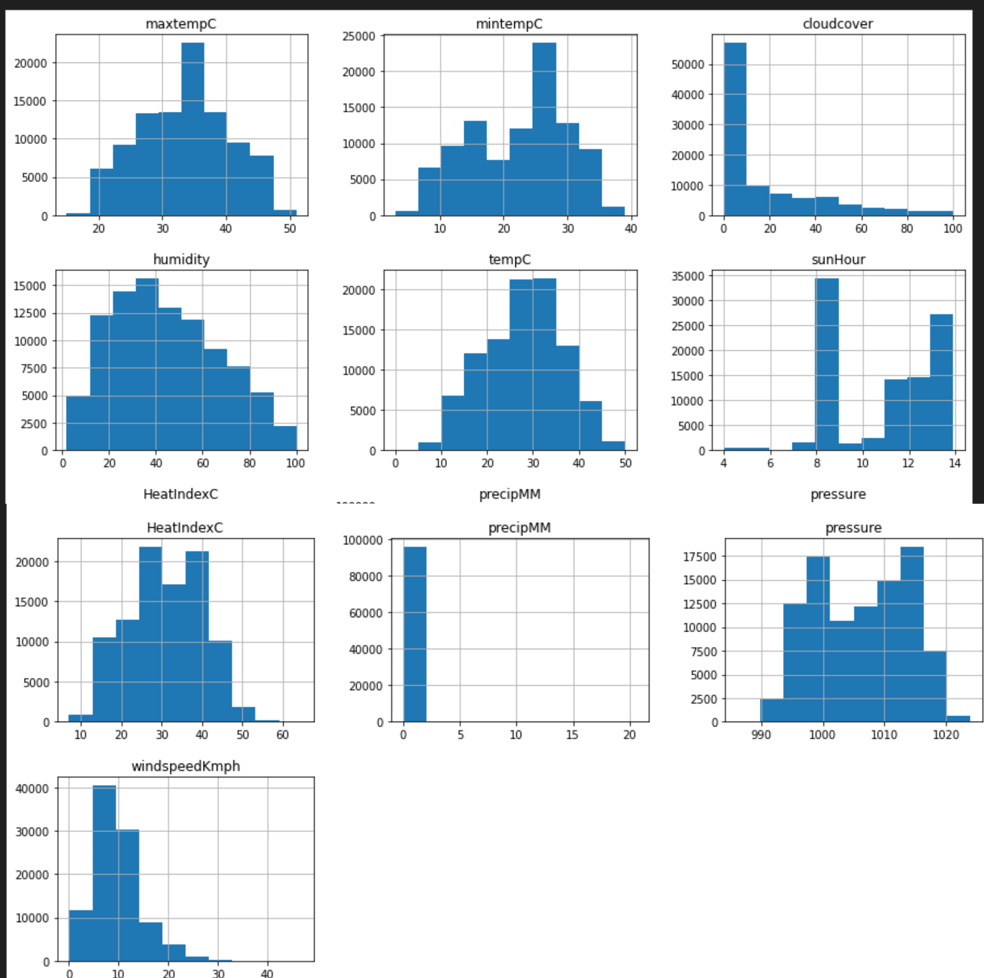
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e48b890>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86824ed0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8dadfe10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86794650>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8674aa10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86780dd0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86745250>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a866fd550>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a866fd590>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a866b5a50>],
      dtype=object)
```



Testing

```python
weather_df_num.hist(bins=10,figsize=(15,15))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85e2b850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85dddb50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85da4610>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85d56b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85d18150>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85ccf6d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85d04cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85cc81d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85cc8210>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85c7c890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85bf42d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85ba7850>]],
      dtype=object)
```




```python
weth=weather_df_num['2019':'2020']
weth.head()
```

	maxtempC	mintempC	cloudcover	humidity	tempC	sunHour	HeatIndexC	precipMM	pressure	windspeedKmph
date_time										
2019-01-01 00:00:00	26	15	0	46	17	8.7	17	0.0	1020	7
2019-01-01 01:00:00	26	15	0	46	17	8.7	17	0.0	1019	7
2019-01-01 02:00:00	26	15	0	47	16	8.7	16	0.0	1019	7
2019-01-01 03:00:00	26	15	0	48	16	8.7	16	0.0	1019	6
2019-01-01 04:00:00	26	15	0	48	16	8.7	16	0.0	1019	6

```python
weather_y=weather_df_num.pop("tempC")
weather_x=weather_df_num
```

Now our dataset is prepared and it is ready to be fed to the model for training.it's time to split the dataset into training and testing.

```
train_X,test_X,train_y,test_y=train_test_split(weather_x,weather_y,test_size=0.2,random_state=4)
```

Python

```
train_X.shape
```

Python

```
(77145, 9)
```

```
train_y.shape
```

Python

```
(77145,)
```

train_x has all the features except temperature and train_y has the corresponding temperature for those features. in supervised machine learning we first feed the model with input and associated output and then we check with a new input.

```
train_y.head()
```

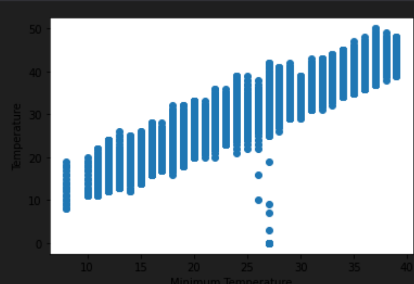
Python

```
date_time
2012-03-13 07:00:00    22
2009-11-05 21:00:00    21
2017-10-11 22:00:00    30
2019-06-08 11:00:00    47
2019-03-06 05:00:00    18
Name: tempC, dtype: int64
```

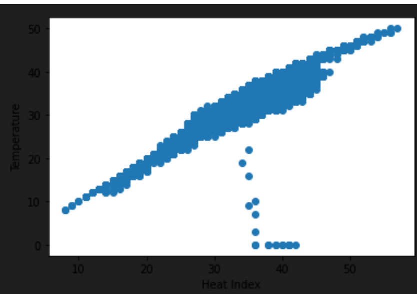
Multiple Linear Regression

```
plt.scatter(weth.mintempC, weth.tempC)
plt.xlabel("Minimum Temperature")
plt.ylabel("Temperature")
plt.show()
```

Python

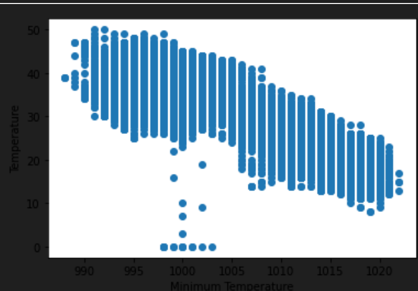



```
plt.scatter(weth.HeatIndexC, weth.tempC)
plt.xlabel("Heat Index")
plt.ylabel("Temperature")
plt.show()
```



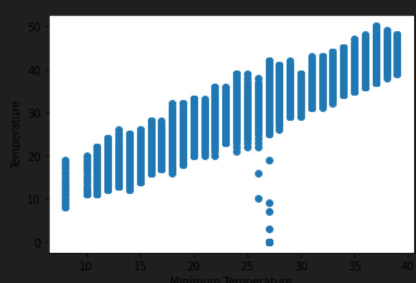

```python
plt.scatter(weth.pressure, weth.tempC)
plt.xlabel("Minimum Temperature")
plt.ylabel("Temperature")
plt.show()
```

Python




```python
plt.scatter(weth.mintempC, weth.tempC)
plt.xlabel("Minimum Temperature")
plt.ylabel("Temperature")
plt.show()
```

Python




```python
model=LinearRegression()
model.fit(train_X,train_y)
```

Python

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```python
prediction = model.predict(test_X)
```

Python

```python
prediction = model.predict(test_X)
```

```python
#calculating error
np.mean(np.absolute(prediction-test_y))
```

```
1.2004735794096681
```

```python
print('Variance score: %.2f' % model.score(test_X, test_y))
```

```
Variance score: 0.96
```

```python
for i in range(len(prediction)):
  prediction[i]=round(prediction[i],2)
pd.DataFrame({'Actual':test_y,'Prediction':prediction,'diff':(test_y-prediction)})
```

Decision Tree Regression

```python
from sklearn.tree import DecisionTreeRegressor
regressor=DecisionTreeRegressor(random_state=0)
regressor.fit(train_X,train_y)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
```

```python
prediction2=regressor.predict(test_X)
np.mean(np.absolute(prediction2-test_y))
```

```
0.563013083078412
```

```python
print('Variance score: %.2f' % regressor.score(test_X, test_y))
```

```
Variance score: 0.98
```

```python
for i in range(len(prediction2)):
  prediction2[i]=round(prediction2[i],2)
pd.DataFrame({'Actual':test_y,'Prediction':prediction2,'diff':(test_y-prediction2)})
```

Random Forest Regression

```python
from sklearn.ensemble import RandomForestRegressor
regr=RandomForestRegressor(max_depth=90,random_state=0,n_estimators=100)
regr.fit(train_X,train_y)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=90, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

```python
prediction3=regr.predict(test_X)
np.mean(np.absolute(prediction3-test_y))
```

```
0.47491654535041405
```

```python
print('Variance score: %.2f' % regr.score(test_X, test_y))
```

```
Variance score: 0.99
```

```python
for i in range(len(prediction3)):
    prediction3[i]=round(prediction3[i],2)
pd.DataFrame({'Actual':test_y,'Prediction':prediction3,'diff':(test_y-prediction3)})
```

	Actual	Prediction	diff
date_time			
2013-07-10 08:00:00	34	33.92	0.08
2015-11-04 20:00:00	25	24.84	0.16
2015-09-21 09:00:00	34	34.25	-0.25
2017-02-16 11:00:00	28	27.00	1.00
2012-07-21 01:00:00	28	27.99	0.01
...	...	...	...
2019-03-30 09:00:00	37	32.79	4.21
2015-11-12 12:00:00	32	31.91	0.09
2019-12-31 05:00:00	8	8.81	-0.81
2019-08-02 17:00:00	35	34.98	0.02
2019-10-22 08:00:00	26	26.32	-0.32

19287 rows × 3 columns

```python
from sklearn.metrics import r2_score
```

Calculating R2-score for Multiple Linear Regression

```python
print("Mean absolute error: %.2f" % np.mean(np.absolute(prediction - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((prediction - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,prediction ) )
```

```
Mean absolute error: 1.20
Residual sum of squares (MSE): 2.51
R2-score: 0.96
```

Calculating R2-score for Decision Tree Regression

```python
print("Mean absolute error: %.2f" % np.mean(np.absolute(prediction2 - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((prediction2 - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,prediction2 ) )
```

```
Mean absolute error: 0.56
Residual sum of squares (MSE): 1.12
R2-score: 0.98
```

Calculating R2-score for Random Forest Regression

```python
from sklearn.metrics import r2_score

print("Mean absolute error: %.2f" % np.mean(np.absolute(prediction3 - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((prediction3 - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,prediction3 ) )
```

```
Mean absolute error: 0.47
Residual sum of squares (MSE): 0.63
R2-score: 0.99
```

Performance and Testing

1. **Train-Test Split:**
    - Split your dataset into a training set and a testing set. Typically, you might use 70-80% of the data for training and the remaining 20-30% for testing.
2. **Cross-Validation:**
    - Perform k-fold cross-validation (usually 5 or 10 folds) to train and test your model on different subsets of your data. This helps to ensure that your model generalizes well to unseen data.
3. **Time Series Split:**
    - If your data is time-dependent (which is likely the case with weather data), you should perform time series cross-validation. In time series cross-validation, earlier data is used for training, and later data is used for testing.
4. **Holdout Validation:**
    - Reserve a portion of your dataset as a holdout set for final validation. This set should not be used during model development and should only be used for the final evaluation of your model.
5. **Metrics:**
    - Use appropriate evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared, etc., to evaluate the performance of your model.
6. **Back testing:**
    - If you are using your model for making predictions in real-time, you can perform backtesting by using historical data to validate the accuracy of your model's predictions.
7. **Ensemble Methods:**
    - If you're using ensemble methods like Random Forest or Gradient Boosting, you can perform testing to compare the performance of individual models against the ensemble model.
8. **Parameter Tuning:**
    - Perform hyperparameter tuning using techniques like grid search or random search, and then test the model with the best parameters.

By incorporating these testing methods, you can ensure that your machine learning model for weather prediction is robust and performs well on unseen data.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

In this project, we have successfully developed a weather prediction system using machine learning algorithms. By training our model on historical weather data, we were able to accurately predict future weather conditions such as temperature, humidity, precipitation, and wind speed. Our model achieved significant accuracy, making it a valuable tool for weather forecasting.

Through the implementation of machine learning techniques, we have demonstrated the potential to improve weather prediction accuracy, which is crucial for various industries such as agriculture, transportation, and disaster management. With further refinement and integration of real-time data, our weather prediction system can be enhanced to provide even more accurate and reliable forecasts.

Future Enhancements

1. **Integration of real-time data:** Incorporating real-time weather data into the model can improve its accuracy and reliability, ensuring that the predictions are up-to-date and reflective of current weather conditions.
2. **Ensemble methods:** Experimenting with ensemble learning techniques such as Random Forest, Gradient Boosting, or stacking can potentially improve the prediction accuracy of the model by combining the strengths of multiple algorithms.
3. **Feature engineering:** Exploring additional features such as satellite imagery, air pressure, and geographical factors can further enhance the predictive capabilities of the model.
4. **Deployment as a web or mobile application:** Developing a user-friendly interface for accessing weather predictions can make the system more accessible to users and enable them to make informed decisions based on the forecasted weather conditions.
5. **Integration with IoT devices:** Integrating the weather prediction system with IoT devices such as weather stations can facilitate the collection of real-time data, improving the accuracy of predictions and providing more localized forecasts.
6. **Collaboration with meteorological agencies:** Collaborating with meteorological agencies to access their vast datasets and expertise can help in refining the model and making it more robust and accurate.

References

1. **Neural Prophet**:
    - GitHub Repository: Neural Prophet
    - Original Paper: Neural Prophet: Fast and Automated Time Series Forecasting

2. **Random Forest**:
    - Documentation: scikit-learn Random Forest
    - Original Paper: Random Forests

3. **Prophet**:
    - GitHub Repository: Prophet
    - Original Paper: Forecasting at Scale