MACHINE LEARNING

(Predicting the AirQuality by Relative Humidity)

Summer Internship Report Submitted in partial fulfillment of

the requirement for undergraduate

degree of **Bachelor of Technology**

In

ComputerScience and Engineering

By

SaiGanesh Paindla

221710305039

Under the Guidance of

Department Of Computer Science

GITAM School of Technology

GITAM (Deemed to be University) Hyderabad-50232,July 2020.

DECLARATION

I submit this industrial training work entitled "**PREDICTING AIR QUALITY BY RELATIVE HUMIDITY**" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of **"Bachelor of Technology"** in "**Computer Science Engineering**". I declare that it was carried out independently by me under the guidance of , GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD **Sai Ganesh Paindla**

Date: **221710305039**

Ii

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled " **AIR QUALITY PREDICTION BY RELATIVE HUMIDITY**" is being submitted by StudentName (StudentRollNumber) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19

It is faithful record work carried out by her at the **Computer Science Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Dr.N.Seetharamaiah**, Principal, GITAM Hyderabad

I would like to thank respected **Prof. Phani Kumar**, Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties Mr. Phani Kumar who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end

Sai Ganesh Paindla

221710305039

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on Cereals data set My perception of understanding the given data set has been in the view of undertaking a client's requirement of overcoming the stagnant point of sales of the products being manufactured by client.

To get a better understanding and work on a strategical approach for solution of the client, I have adapted the view point of looking at ratings of the products and for further deep understanding of the problem, I have taken the stance of a consumer and reasoned out the various factors of choice of the products and they purchase , and my primary objective of this case study was to look up the factors which were dampening the sale of products and corelate them to ratings of products and draft out an outcome report to client regarding the various accepts of a product manufacturing , marketing and sale point determination

Table of Contents:

LIST OF FIGURES

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

Figure 1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data.

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

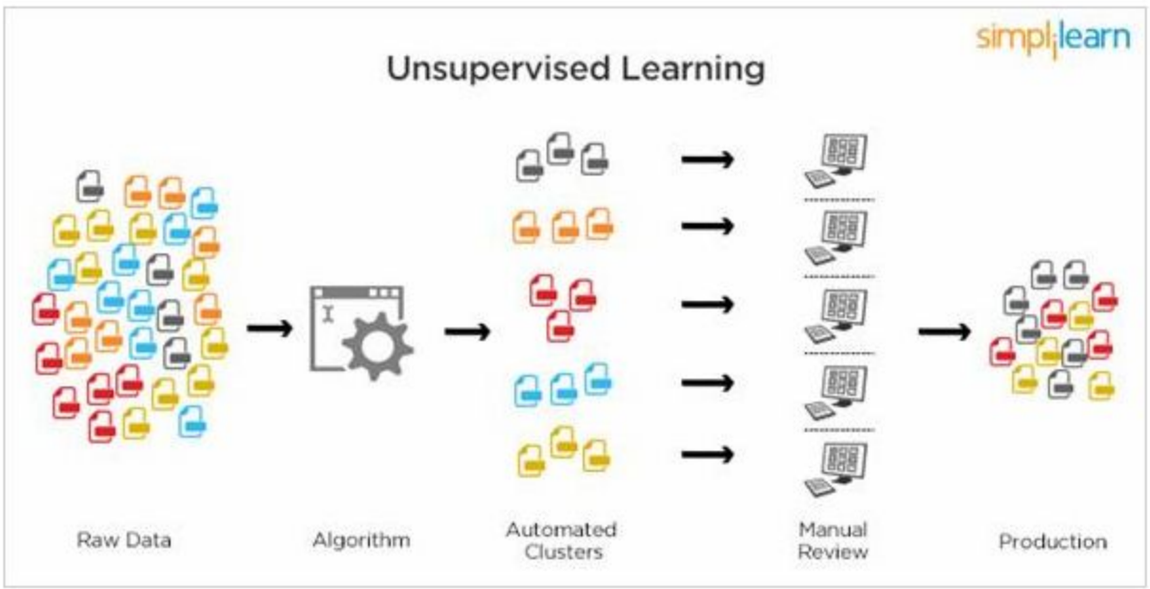

Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

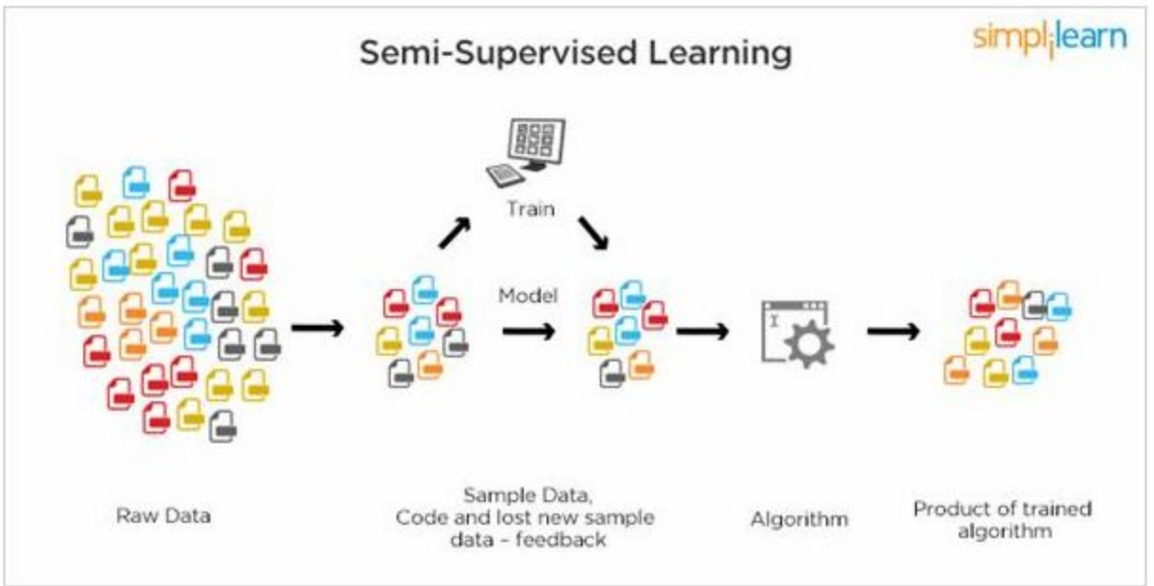

Figure 3 : Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYTHON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles.
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

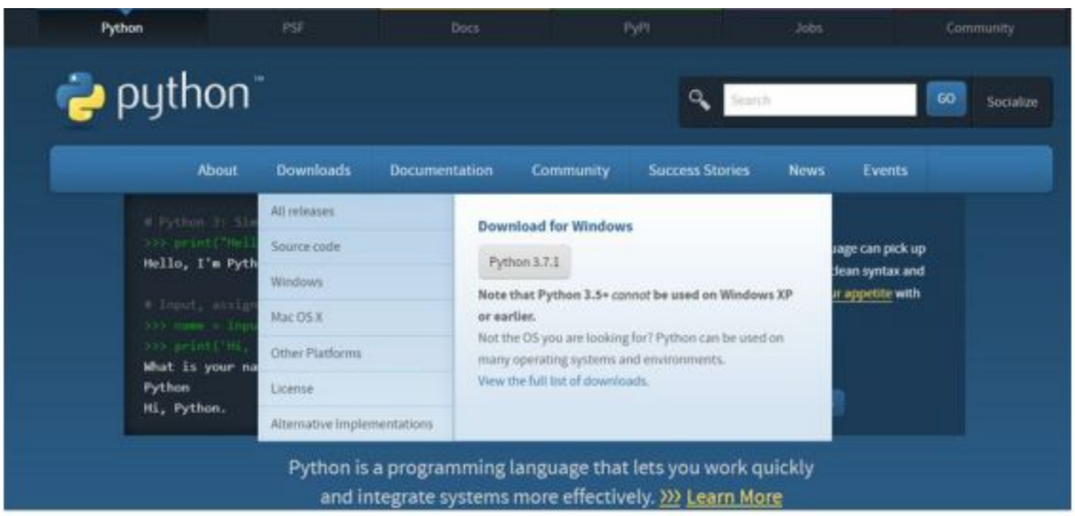

Figure 4 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

In WINDOWS:

- In windows
- Step 1: Open Anaconda.com/downloads in web browser.
- Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
- Step 3: select installation type(all users)
- Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
- Step 5: Open jupyter notebook (it opens in default browser)

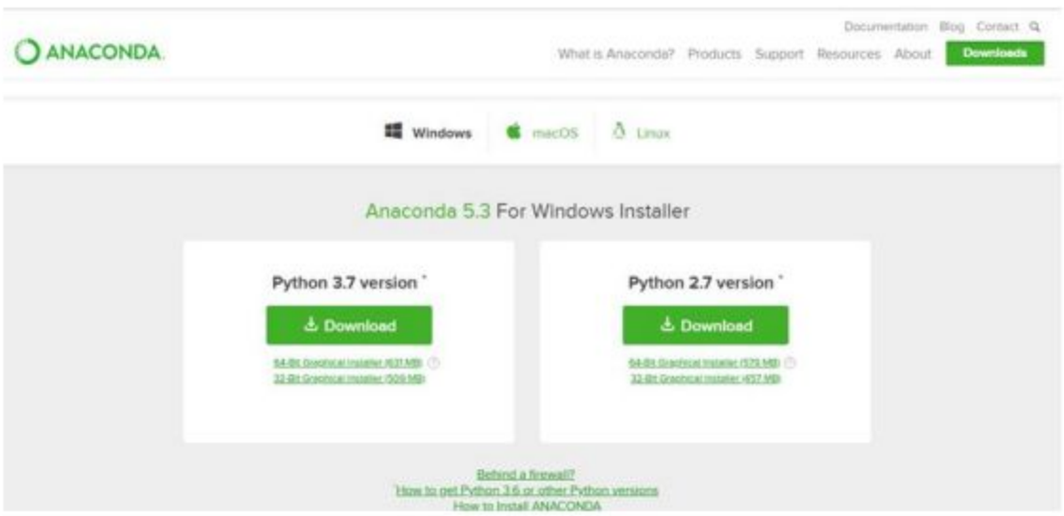

Figure 5 : Anaconda download

Figure 6 : Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
- Numbers
- Strings
- Lists
- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.

- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Defining a Class:

    ➔ We define a class in a very similar way how we define a function

➔ Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 7 : Defining a Class

2.7.2 __init__ method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.

- The init method has a special name that starts and ends with two underscores:__init__().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

- Prediction of pollution is an increasingly important problem. It can impact individuals and their health, e.g. asthma patients can be greatly affected by air pollution.
- Traditional air pollution prediction methods have limitations. Machine learning provides one approach that can offer new opportunities for prediction of air pollution. There are however many different machine learning approaches and identifying the best one for the problem at hand is often challenging.
- So we are predicting the Air Quality

MACHINE LEARNING MODEL USED

In the Air Quality Prediction Problem, as the name suggests it is a **Prediction** problem.

- I am going to use **Linear Regression** and **Decision Tree Regression** Machine Learning models.

3.2 DATA SET:

DESCRIPTION

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city.

Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses.

Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer.

Missing values are tagged with -200 value.

Sl No	Attribute	Description
0	Date	Date (DD/MM/YYYY)
1	Time	Time (HH.MM.SS)
2	CO(GT)	True hourly averaged concentration CO in mg/m^3 (reference analyzer)
3	PT08.S1(CO)	PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
4	NMHC(GT)	True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)
5	C6H6(GT)	True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)
6	PT08.S2(NMHC)	PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
7	NOx(GT)	True hourly averaged NOx concentration in ppb (reference analyzer)
8	PT08.S3(NOx)	PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
9	NO2(GT)	True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)
10	PT08.S4(NO2)	PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
11	PT08.S5(O3)	PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
12	T	Temperature in Â°C
13	RH	Relative Humidity (%)
14	AH	AH Absolute Humidity

Figure 3.2.1: Dataset Attributes and its Description

3.3 OBJECTIVE OF THE CASE STUDY:

We will predict the **Relative Humidity** of a given point of time based on the all other attributes affecting the change in RH.

CHAPTER 4

MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from client.

4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm

Importing Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Figure 4.1.2 : Importing Libraries

4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.

Loading Data

```
Air_Quality=pd.read_excel('C:\\Users\\DELL\Documents\\AirQualityUCI\\AirQualityUCI.xlsx')
Air_Quality.head()
```

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	F
0	2004-03-10	18:00:00	2.6	1360.00	150	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.60	48.8750
1	2004-03-10	19:00:00	2.0	1292.25	112	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.30	47.7000
2	2004-03-10	20:00:00	2.2	1402.00	88	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.90	53.9750
3	2004-03-10	21:00:00	2.2	1375.50	80	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.00	60.0000
4	2004-03-10	22:00:00	1.6	1272.25	51	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.15	59.5750

Figure 4.1.3 : Loading the Dataset

4.1.4 HANDLING MISSING VALUES:

```
# Changing Missing values to Nan
Air_Quality=Air_Quality.replace({'CO(GT)':-200,'PT08.S1(CO)':-200,'NMHC(GT)':-200,'C6H6(GT)':-200,'PT08.S2(NMHC)':-200,
                                 'NOx(GT)':-200,'PT08.S3(NOx)':-200,'NO2(GT)':-200,'PT08.S4(NO2)':-200,
                                 'PT08.S5(O3)':-200,'T':-200,'RH':-200,'AH':-200},np.nan)
Air_Quality
```

Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH
2004-03-10	18:00:00	2.6	1360.00	150.0	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.600	48.875001
2004-03-10	19:00:00	2.0	1292.25	112.0	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.700000
2004-03-10	20:00:00	2.2	1402.00	88.0	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.900	53.975000
2004-03-10	21:00:00	2.2	1375.50	80.0	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.000	60.000000
2004-03-10	22:00:00	1.6	1272.25	51.0	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.150	59.575001
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2005-04-04	10:00:00	3.1	1314.25	NaN	13.529605	1101.25	471.7	538.50	189.8	1374.25	1728.50	21.850	29.250000
2005-04-04	11:00:00	2.4	1162.50	NaN	11.355157	1027.00	353.3	603.75	179.2	1263.50	1269.00	24.325	23.725000
2005-04-04	12:00:00	2.4	1142.00	NaN	12.374538	1062.50	293.0	603.25	174.7	1240.75	1092.00	26.900	18.350000

Figure 4.1.4.1 : Replacing -200 values with Nan

```
# Missing values in each column
Air_Quality.isnull().sum()
```

```
Time             0
CO(GT)        1683
PT08.S1(CO)    366
NMHC(GT)      8443
C6H6(GT)       366
PT08.S2(NMHC)  366
NOx(GT)       1639
PT08.S3(NOx)   366
NO2(GT)       1642
PT08.S4(NO2)   366
PT08.S5(O3)    366
T              366
RH             366
AH             366
dtype: int64
```

Figure 4.1.4.2 : Checking the total number of Missing values in each column

```
# Since there are 80-90 % of missing values from NMHC(GT) column will
Air_Quality.drop('NMHC(GT)',axis=1,inplace=True)
Air_Quality.head()
```

Date	Time	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
2004-03-10	18:00:00	2.6	1360.00	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.60	48.875001	0.757754
2004-03-10	19:00:00	2.0	1292.25	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.30	47.700000	0.725487
2004-03-10	20:00:00	2.2	1402.00	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.90	53.975000	0.750239
2004-03-10	21:00:00	2.2	1375.50	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.00	60.000000	0.786713
2004-03-10	22:00:00	1.6	1272.25	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.15	59.575001	0.788794

Figure 4.1.4.3 : Using Drop to drop a column

```
#droping end rows with NaN values
Air_Quality.dropna(how='all',inplace=True)
#drop RH NAN rows
Air_Quality.dropna(thresh=10,axis=0,inplace=True)
```

Figure 4.1.4.4 : Using Dropna to handle missing values

Mean and Median imputation:

- mean and median imputation can be performed by using fillna().
- mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.
- median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

```
Air_Quality.mean()
```

```
CO(GT)             2.129711
PT08.S1(CO)     1099.707856
C6H6(GT)          10.082993
PT08.S2(NMHC)    939.029205
NOx(GT)          242.174351
PT08.S3(NOx)     835.370973
NO2(GT)          112.129974
PT08.S4(NO2)    1456.143486
PT08.S5(O3)     1022.780725
T                 18.316054
RH                49.232360
AH                 1.025530
dtype: float64
```

Figure 4.1.4.5 : Checking mean

```
# Applying Mean Imputation to fill Nan Valus of C0_GT,NOX_GT,NO2_GT columns
Air_Quality=Air_Quality.fillna(Air_Quality.mean())
Air_Quality
```

Date	Time	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
2004-03-10	18:00:00	2.6	1360.00	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.600	48.875001	0.757754
2004-03-10	19:00:00	2.0	1292.25	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.700000	0.725487
2004-03-10	20:00:00	2.2	1402.00	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.900	53.975000	0.750239
2004-03-10	21:00:00	2.2	1375.50	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.000	60.000000	0.786713
2004-03-10	22:00:00	1.6	1272.25	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.150	59.575001	0.788794
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2005-04-04	10:00:00	3.1	1314.25	13.529605	1101.25	471.7	538.50	189.8	1374.25	1728.50	21.850	29.250000	0.756824
2005-04-04	11:00:00	2.4	1162.50	11.355157	1027.00	353.3	603.75	179.2	1263.50	1269.00	24.325	23.725000	0.711864
2005-04-04	12:00:00	2.4	1142.00	12.374538	1062.50	293.0	603.25	174.7	1240.75	1092.00	26.900	18.350000	0.640649
2005-04-04	13:00:00	2.1	1002.50	9.547187	960.50	234.5	701.50	155.7	1041.00	769.75	28.325	13.550000	0.513866

Figure 4.1.4.6 : Applying Mean Imputation

4.2 TRAINING THE MODEL:

4.2.1 Method 1:

- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier
- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt)
- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%)
- First we need to identify the input and output variables and we need to separate the input set and output set
- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method
  This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables)

TRAIN AND TEST DATA :

```
# Spliting the data into Input and Output
inpcol=Air_Quality.columns.tolist()[2:] # Input Columns
X=Air_Quality[inpcol].drop('RH',1)     #X-input features
y=Air_Quality['RH']                    #y-input features
```

```
# Divide the X and y into train and test
from sklearn.model_selection import train_test_split
X_train, X_test,y_train, y_test = train_test_split(X, y, test_size= 0.3,random_state=42)
```

Figure 4.2.1.1 : Training the Model

- Then we need to import linear regression method from linear_model package from scikit learn library
- We need to train the model based on our train set (that we have obtained from splitting)
- Then we have to test the model for the test set ,that is done as follows
- We have a method called predict , using this method we need to predict the output for input test set and we need to compare the out but with the output test data
- If the predicted values and the original values are close then we can say that model is trained with good accuracy

```
print('Training data size:',X_train.shape)
print('Test data size:',X_test.shape)
```

```
Training data size: (6293, 10)
Test data size: (2698, 10)
```

4.3 EVALUATING THE CASE STUDY:

- MULTIPLE LINEAR REGRESSION: A multiple linear regression model allows us to capture the relationship between multiple feature columns and the target column. Here's what the formula looks like: y^=a0+a1x1+a2x2+...+anxn
- Importing the required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Figure 4.3.1 : Importing Libraries

- Reading the Data-Set

```
Air_Quality=pd.read_excel('C:\\Users\\DELL\\Documents\\AirQualityUCI\\AirQualityUCI.xlsx')
Air_Quality.head()
```

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	
0	2004-03-10	18:00:00	2.6	1360.00	150	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.60	48.8750
1	2004-03-10	19:00:00	2.0	1292.25	112	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.30	47.7000
2	2004-03-10	20:00:00	2.2	1402.00	88	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.90	53.9750
3	2004-03-10	21:00:00	2.2	1375.50	80	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.00	60.0000
4	2004-03-10	22:00:00	1.6	1272.25	51	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.15	59.5750

Figure 4.3.2 : Reading the dataset

- Handling the missing values

```
# Changing Missing values to Nan
Air_Quality=Air_Quality.replace({'CO(GT)':-200,'PT08.S1(CO)':-200,'NMHC(GT)':-200,'C6H6(GT)':-200,'PT08.S2(NMHC)':-200,
                                 'NOx(GT)':-200,'PT08.S3(NOx)':-200,'NO2(GT)':-200,'PT08.S4(NO2)':-200,
                                 'PT08.S5(O3)':-200,'T':-200,'RH':-200,'AH':-200},np.nan)
Air_Quality
```

Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH
2004-03-10	18:00:00	2.6	1360.00	150.0	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.600	48.87500
2004-03-10	19:00:00	2.0	1292.25	112.0	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.70000
2004-03-10	20:00:00	2.2	1402.00	88.0	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.900	53.97500
2004-03-10	21:00:00	2.2	1375.50	80.0	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.000	60.00000
2004-03-10	22:00:00	1.6	1272.25	51.0	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.150	59.57500
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2005-04-04	10:00:00	3.1	1314.25	NaN	13.529605	1101.25	471.7	538.50	189.8	1374.25	1728.50	21.850	29.25000
2005-04-04	11:00:00	2.4	1162.50	NaN	11.355157	1027.00	353.3	603.75	179.2	1263.50	1269.00	24.325	23.72500
2005-04-04	12:00:00	2.4	1142.00	NaN	12.374538	1062.50	293.0	603.25	174.7	1240.75	1092.00	26.900	18.35000
2005-04-04	13:00:00	2.1	1002.50	NaN	9.547187	960.50	234.5	701.50	155.7	1041.00	769.75	28.325	13.55000
2005-04-04	14:00:00	2.2	1070.75	NaN	11.932060	1047.25	265.2	654.00	167.7	1128.50	816.00	28.500	13.12500

Figure 4.3.3.1 : Replacing -200 with Nan

```
import missingno as msno
msno.matrix(Air_Quality)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2149da67f88>
```

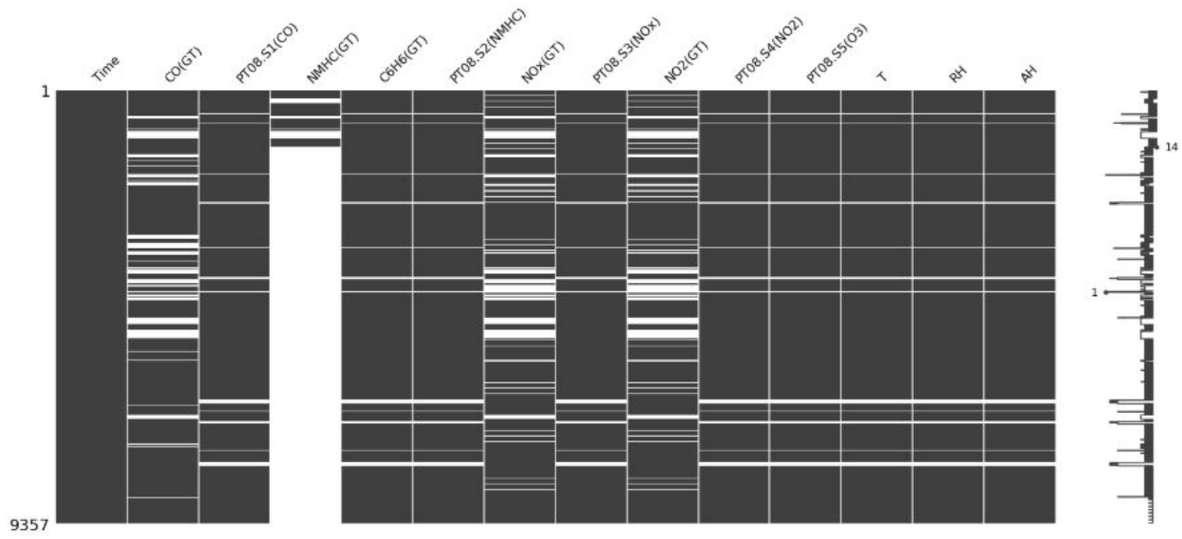

Figure 4.3.3.2 : Visualizing The missing values

```
# Since there are 80-90 % of missing values from NMHC(GT) column will
Air_Quality.drop('NMHC(GT)',axis=1,inplace=True)
Air_Quality.head()
```

Date	Time	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
2004-03-10	18:00:00	2.6	1360.00	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.60	48.875001	0.757754
2004-03-10	19:00:00	2.0	1292.25	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.30	47.700000	0.725487
2004-03-10	20:00:00	2.2	1402.00	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.90	53.975000	0.750239
2004-03-10	21:00:00	2.2	1375.50	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.00	60.000000	0.786713
2004-03-10	22:00:00	1.6	1272.25	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.15	59.575001	0.788794

Figure 4.3.3.3 : Using drop

```
# There are about 30% missing values in C0_GT,NOX_GT,NO2_GT columns Which can be filled with avearge Mean
import missingno as msno
msno.matrix(Air_Quality)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2149dc9ed08>
```

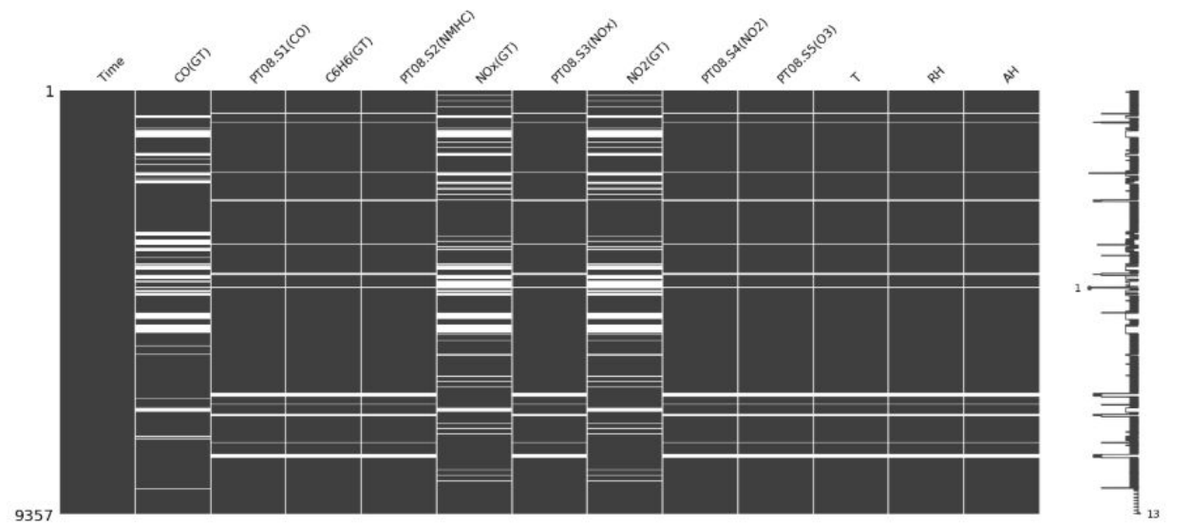

Figure 4.3.3.4 : Visualizing after Drop function

```
: #droping end rows with NaN values
Air_Quality.dropna(how='all',inplace=True)
#drop RH NAN rows
Air_Quality.dropna(thresh=10,axis=0,inplace=True)
```

Figure 4.3.3.5 : Using Dropna

```
Air_Quality.mean()
```

```
CO(GT)              2.129711
PT08.S1(CO)      1099.707856
C6H6(GT)           10.082993
PT08.S2(NMHC)     939.029205
NOx(GT)           242.174351
PT08.S3(NOx)      835.370973
NO2(GT)           112.129974
PT08.S4(NO2)     1456.143486
PT08.S5(O3)      1022.780725
T                  18.316054
RH                 49.232360
AH                  1.025530
dtype: float64
```

Figure 4.3.3.6 : Checking mean

```
# Applying Mean Imputation to fill Nan Valus of C0_GT,NOX_GT,NO2_GT columns
Air_Quality=Air_Quality.fillna(Air_Quality.mean())
Air_Quality
```

Date	Time	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
2004-03-10	18:00:00	2.6	1360.00	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.600	48.875001	0.757754
2004-03-10	19:00:00	2.0	1292.25	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.700000	0.725487
2004-03-10	20:00:00	2.2	1402.00	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.900	53.975000	0.750239
2004-03-10	21:00:00	2.2	1375.50	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.000	60.000000	0.786713
2004-03-10	22:00:00	1.6	1272.25	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.150	59.575001	0.788794
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2005-04-04	10:00:00	3.1	1314.25	13.529605	1101.25	471.7	538.50	189.8	1374.25	1728.50	21.850	29.250000	0.756824
2005-04-04	11:00:00	2.4	1162.50	11.355157	1027.00	353.3	603.75	179.2	1263.50	1269.00	24.325	23.725000	0.711864
2005-04-04	12:00:00	2.4	1142.00	12.374538	1062.50	293.0	603.25	174.7	1240.75	1092.00	26.900	18.350000	0.640649
2005-04-04	13:00:00	2.1	1002.50	9.547187	960.50	234.5	701.50	155.7	1041.00	769.75	28.325	13.550000	0.513866
2005-04-04	14:00:00	2.2	1070.75	11.932060	1047.25	265.2	654.00	167.7	1128.50	816.00	28.500	13.125000	0.502804

8991 rows × 13 columns

Figure 4.3.3.7 : Applying Mean Imputation

4.4.1 Building the model (using splitting):

- Import the train_test_split from model_selection package from scikitlearn library
- Then assigning the output to four different variables, before assigning we have to mention the train size or test size as a parameter to train_test_split. Then this method will split according to the size and assigns it to four variables.
- Import linear regression method which is available in linear_model package from scikit learn library

```
# Spliting the data into Input and Output
inpcol=Air_Quality.columns.tolist()[2:] # Input Columns
X=Air_Quality[inpcol].drop('RH',1)     #X-input features
y=Air_Quality['RH']                    #y-input features
```

```
# Divide the X and y into train and test
from sklearn.model_selection import train_test_split
X_train, X_test,y_train, y_test = train_test_split(X, y, test_size= 0.3,random_state=42)
```

Figure 4.4.1 : Building the Model

Linear Regression:

Linear regression models are used to show or predict the relationship between two variables or factors. The factor that is being predicted (the factor that the equation solves for) is called the dependent variable.

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable *causes* the other (for example, higher SAT scores do not *cause* higher college grades), but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear regression line has an equation of the form $Y = a + bX$, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when $x = 0$).

Linear Regression

```
# Sklearn library: import, instantiate, fit
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train, y_train)
```

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

```
print('Training data size:',X_train.shape)
print('Test data size:',X_test.shape)
```

```
Training data size: (6293, 10)
Test data size: (2698, 10)
```

Figure 4.4.2.1 : Applying Linear Regression

```
# Intercept and the coefficient values
print(lm.intercept_)
lm.coef_
```

```
34.74062514709794

array([ 8.84944501e-03, -8.51167911e-01,  1.63256317e-03,  1.86059426e-02,
       -1.04013965e-03, -4.35652866e-02,  1.52211178e-02, -1.37333335e-03,
       -2.28850830e+00,  3.33199476e+01])
```

Figure 4.4.2.2 : Intercept

```
## Create a dataframe for coefficients
coefficients = pd.DataFrame([X_train.columns, lm.coef_]).T
coefficients
```

	0	1
0	PT08.S1(CO)	0.00884945
1	C6H6(GT)	-0.851168
2	PT08.S2(NMHC)	0.00163256
3	NOx(GT)	0.0186059
4	PT08.S3(NOx)	-0.00104014
5	NO2(GT)	-0.0435653
6	PT08.S4(NO2)	0.0152211
7	PT08.S5(O3)	-0.00137333
8	T	-2.28851
9	AH	33.3199

Figure 4.4.2.3 : Coefficients

Prediction :

```
## Checking the model prediction on training data
y_train_pred = lm.predict(X_train)
y_train_pred
```

```
array([36.40291872, 46.47438462, 63.0739674 , ..., 59.6448551 ,
       69.76388157, 60.79902788])
```

```
## We need to compare the actual values(y_train) and the predicted  #values(y_train_pred)
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print('R^2:', r2_score(y_train,y_train_pred))
print('Adjusted R^2:', 1- (1-r2_score(y_train, y_train_pred))*(len(X_train)-1)/
                                  (len(X_train)-X_train.shape[1]-1))

print('MAE:', mean_absolute_error(y_train, y_train_pred))

print('MSE:', mean_squared_error(y_train, y_train_pred))

print('RMSE', np.sqrt(mean_squared_error(y_train, y_train_pred)))
```

```
R^2: 0.87950463414027
Adjusted R^2: 0.8793128236247341
MAE: 4.641358271356434
MSE: 35.98708404702954
RMSE 5.998923574028056
```

Figure 4.4.2.4 : Prediction Rmse and other error metrics

Decision Tree Regression:

The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with *Standard Deviation Reduction*.

```
from sklearn.tree import DecisionTreeRegressor
dt_one_reg=DecisionTreeRegressor()
dt_model=dt_one_reg.fit(X_train,y_train)         #fit the model
y_pred_dtone=dt_model.predict(X_test)
print('RMSE of Decision Tree Regression:',np.sqrt(mean_squared_error(y_pred_dtone,y_test)))
```

```
RMSE of Decision Tree Regression: 1.1806814921313866
```

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
print('R^2:',r2_score(y_test,y_pred_dtone))
print("Adjuted R^2:",1-(1-r2_score(y_test,y_pred_dtone))*(len(X_test)-1)/
     (len(X_test)-X_test.shape[1]-1))
print("MAE:",mean_absolute_error(y_test,y_pred_dtone))
print("MSE:",mean_squared_error(y_test,y_pred_dtone))
```

```
R^2: 0.99539189559552
Adjuted R^2: 0.9953747459698986
MAE: 0.8704719600602107
MSE: 1.3940087858615975
```

Figure 4.4.3 : Decision Tree Regression

Grid SearchCV:

Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary. Grid-searching does NOT only apply to one model type. Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model. It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run. Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination. Without further ado, lets jump into some examples and implementation.

```
from sklearn.model_selection import GridSearchCV
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

```
from sklearn.metrics import make_scorer
scoring = make_scorer(rmse)
g_cv = GridSearchCV(DecisionTreeRegressor(random_state=0),
             param_grid={'min_samples_split': range(2, 10)},
             scoring=scoring, cv=5, refit=True)
```

Figure 4.4.4.1 : Importing GridSearchCV

```
grid_model=g_cv.fit(X_train, y_train)
grid_model
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=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'),
             iid='deprecated', n_jobs=None,
             param_grid={'min_samples_split': range(2, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(rmse), verbose=0)
```

Figure 4.4.4.2 : Passing parameters

```
g_cv.best_params_
```

```
{'min_samples_split': 9}
```

Figure 4.4.4.3 : Parameters

```
result = g_cv.cv_results_
result
```

```
{'mean_fit_time': array([0.06406054, 0.05312138, 0.05774336, 0.05900187, 0.05276175,
        0.05430145, 0.05377326, 0.05322247]),
 'std_fit_time': array([0.00633888, 0.00765392, 0.00691447, 0.01071552, 0.00995145,
        0.00324958, 0.00514974, 0.00351083]),
 'mean_score_time': array([0.00020003, 0.        , 0.00119987, 0.00139942, 0.00040007,
        0.00200348, 0.00159893, 0.00160394]),
 'std_score_time': array([0.00040007, 0.        , 0.00097978, 0.00119932, 0.00048998,
        0.000638  , 0.00048967, 0.00080201]),
 'param_min_samples_split': masked_array(data=[2, 3, 4, 5, 6, 7, 8, 9],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'min_samples_split': 2},
  {'min_samples_split': 3},
  {'min_samples_split': 4},
  {'min_samples_split': 5},
  {'min_samples_split': 6},
  {'min_samples_split': 7},
  {'min_samples_split': 8},
  {'min_samples_split': 9}],
 'split0_test_score': array([1.39080207, 1.39944669, 1.42226515, 1.39125157, 1.38085851,
        1.41554128, 1.4164491 , 1.43983282]),
 'split1_test_score': array([1.49642233, 1.46605576, 1.49520302, 1.43328633, 1.4492296 ,
        1.48155869, 1.536389  , 1.54829201]),
 'split2_test_score': array([1.46535744, 1.42752681, 1.41870194, 1.42247374, 1.44730034,
        1.44511879, 1.46803696, 1.48664688]),
 'split3_test_score': array([1.41021421, 1.41001888, 1.39277189, 1.4075535 , 1.4034477 ,
        1.38997686, 1.41861233, 1.43068183]),
```

Figure 4.4.4.5 : Storing in result

```
y_pred_grid=grid_model.predict(X_test) # For Testing data
y_pred_grid
```

```
array([51.67000008, 46.33749994, 70.03214291, ..., 46.2750003 ,
       50.24583308, 41.97500006])
```

```
y_pred_traingrid=grid_model.predict(X_train) #for training data
y_pred_traingrid
```

```
array([28.51071419, 45.25833352, 76.97500076, ..., 57.69166634,
       77.65833378, 66.12083387])
```

Figure 4.4.4.6 : Predicting for X train and X test

```
#calculate RMSE
print('RMSE of GridSearch Regression:',np.sqrt(mean_squared_error(y_pred_grid,y_test)))
```

```
RMSE of GridSearch Regression: 1.2635772369963703
```

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
print('R^2:',r2_score(y_test,y_pred_grid))
print("Adjuted R^2:",1-(1-r2_score(y_test,y_pred_grid))*(len(X_test)-1)/
     (len(X_test)-X_test.shape[1]-1))
print("MAE:",mean_absolute_error(y_test,y_pred_grid))
print("MSE:",mean_squared_error(y_test,y_pred_grid))
```

```
R^2: 0.9947221093691205
Adjuted R^2: 0.9947024670519233
MAE: 0.9617704753662504
MSE: 1.5966274338553814
```

Figure 4.4.4.7 : Calculating Rmse for GridSearchCV

CHAPTER 5

DATA VISUALIZATION

5.1 Correlation

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. Correlation is described as the analysis which lets us know the association or the absence of the relationship between two variables ‘x’ and ‘y’. It is a statistical measure that represents the strength of the connection between pairs of variables.

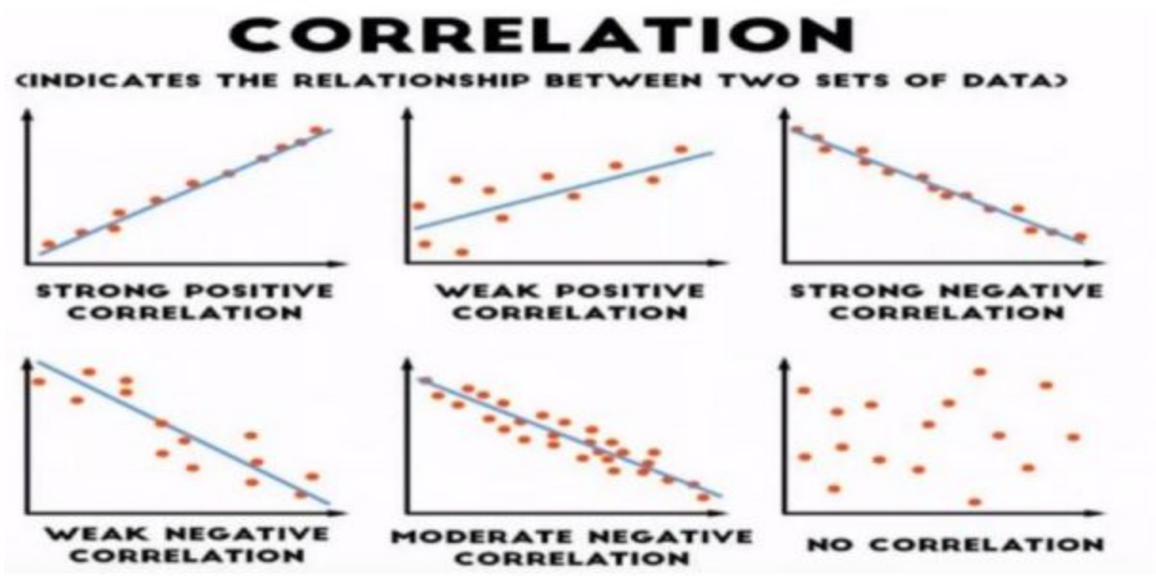

Figure 5.1.1 : Correlation

```
# Using Heatmap to see correlation between variables
plt.figure(figsize=(10,10))
sns.heatmap(Air_Quality.corr(), annot=True, cmap='viridis')
plt.title('Heatmap of co-relation between variables',fontsize=16)
plt.show()
```

Figure 5.1.2 : Using Heatmap to see Correlation between Variables

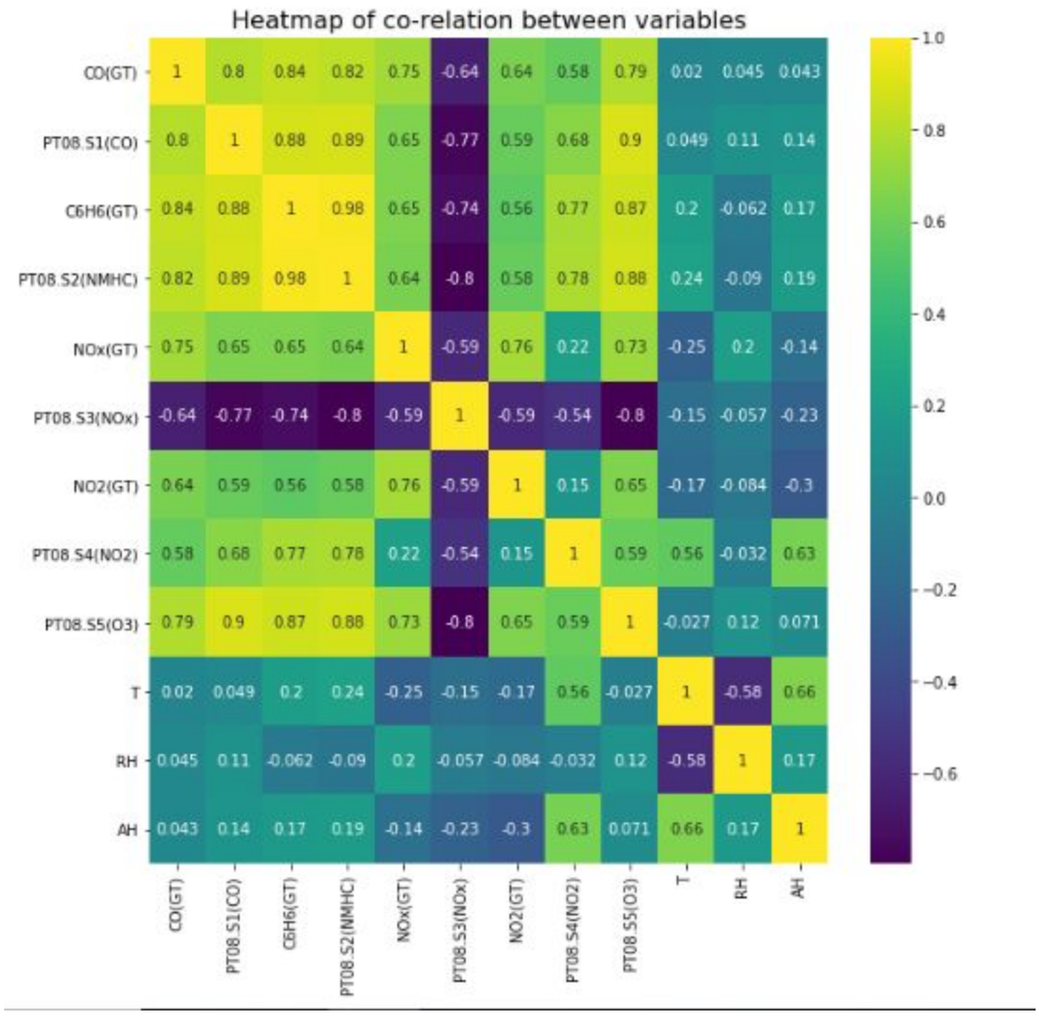

Figure 5.1.3 : Heatmap of correlation

- ➔ In the above plot is used to describe the heatmap of correlation between the variables.
- ➔ The dark yellow color indicates a strong positive correlation between the variables.
- ➔ The dark purple color indicted the a strong negative correlation between the variables.
- ➔ The peacock Blue color indicates a medium correlation between the variables.

5.2 Understanding the Linearity between features

My output column is the RH column and features(input columns) is the rest of the columns

```
[ ]: # Ploting relation between RH column and Features

[ ]: sns.lmplot(x='CO(GT)',y='RH',data=Air_Quality,markers='.')

[ ]: <seaborn.axisgrid.FacetGrid at 0x2149ec38788>
```

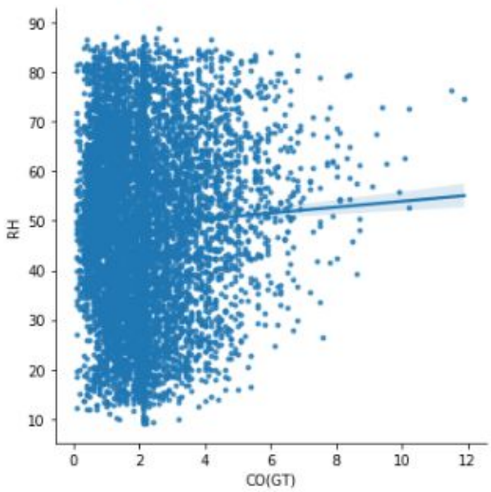

Figure 5.2.1 : Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='PT08.S1(CO)',y='RH',data=Air_Quality,markers='.')

<seaborn.axisgrid.FacetGrid at 0x2149df50a88>
```

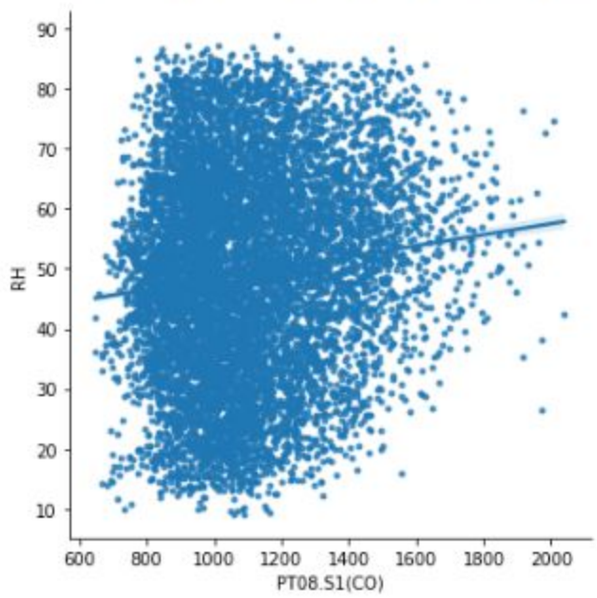

Figure 5.2.2 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='C6H6(GT)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149f1b0208>
```

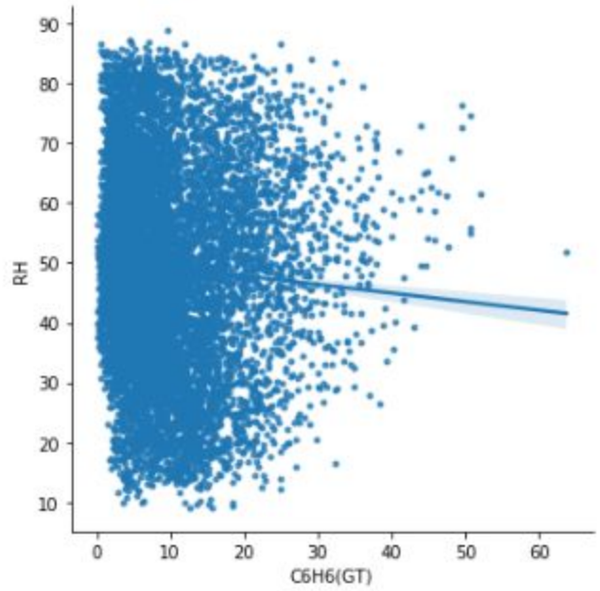

Figure 5.2.3 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='PT08.S2(NMHC)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149f2070c8>
```

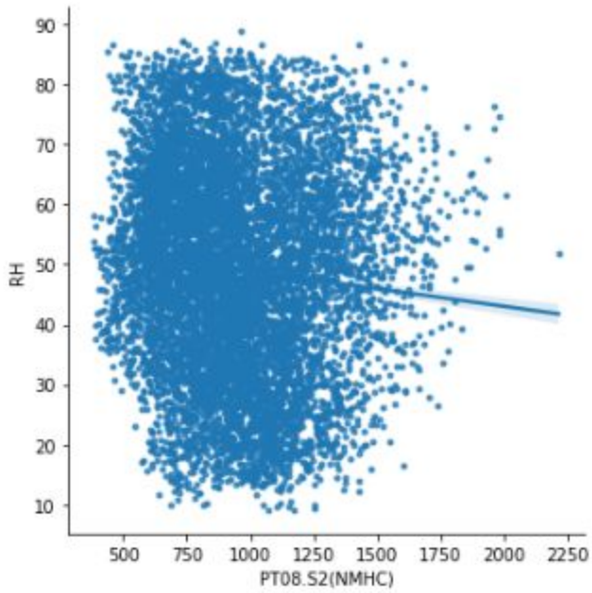

Figure 5.2.4 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='NOx(GT)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149df1c5c8>
```

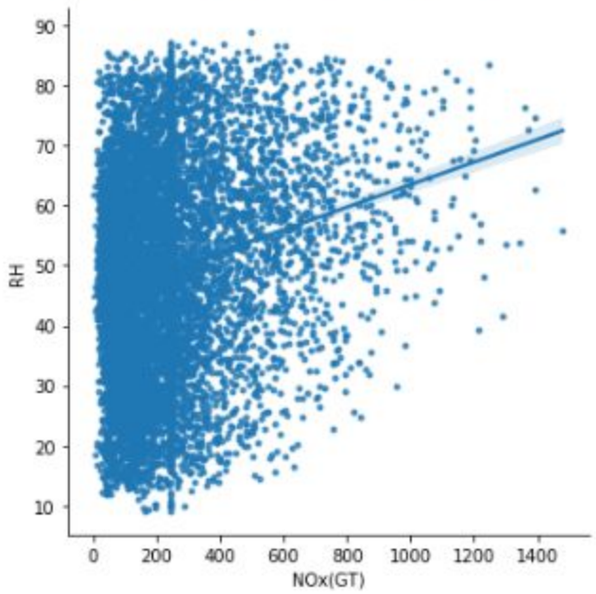

Figure 5.2.5 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='PT08.S3(NOx)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149df3e388>
```

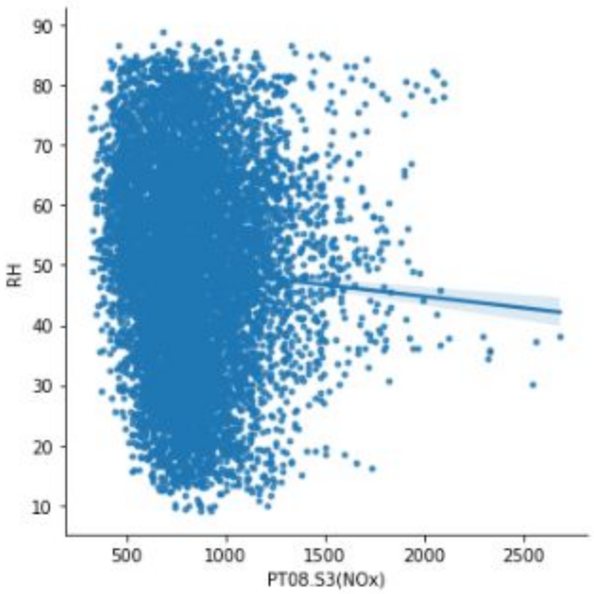

Figure 5.2.6 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='NO2(GT)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149d521248>
```

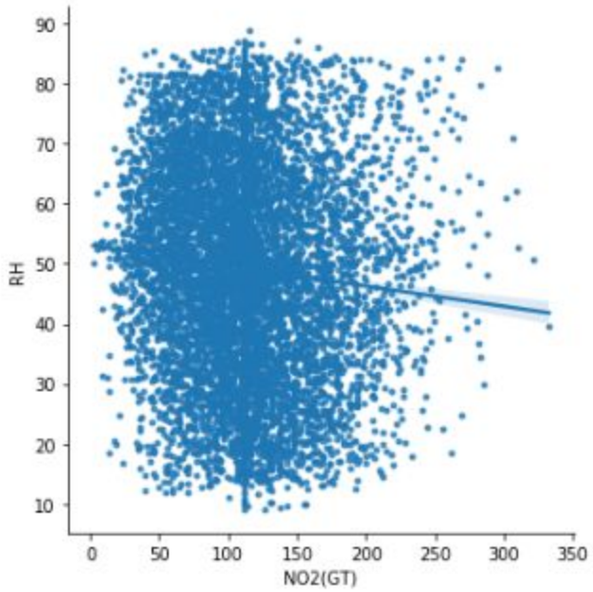

Figure 5.2.7 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='PT08.S4(NO2)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149d83b508>
```

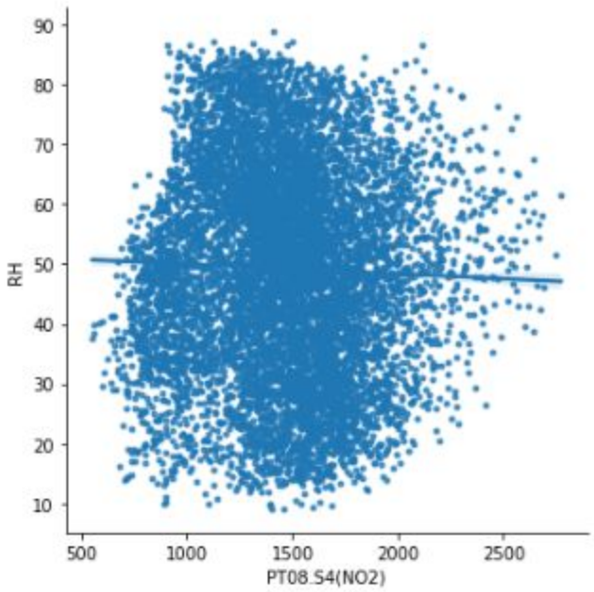

Figure 5.2.8 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='PT08.S5(O3)',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149d9d1908>
```

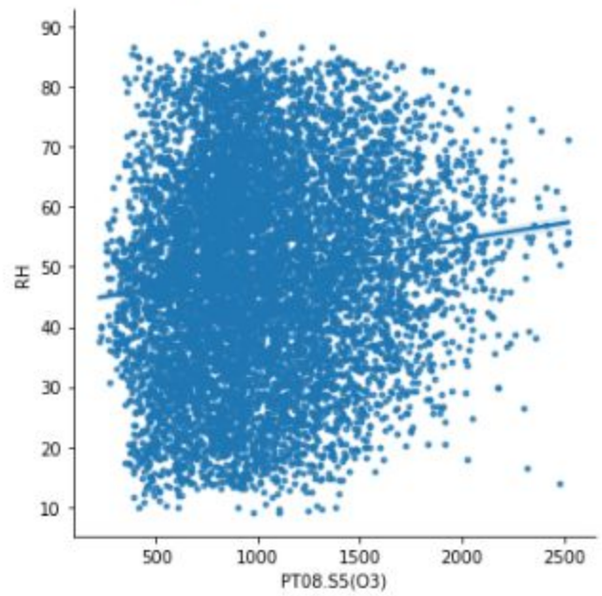

Figure 5.2.9 :Comparing linearity b/w Input features and target feature

```
sns.lmplot(x='T',y='RH',data=Air_Quality,markers='.')
```

```
<seaborn.axisgrid.FacetGrid at 0x2149d8cac48>
```

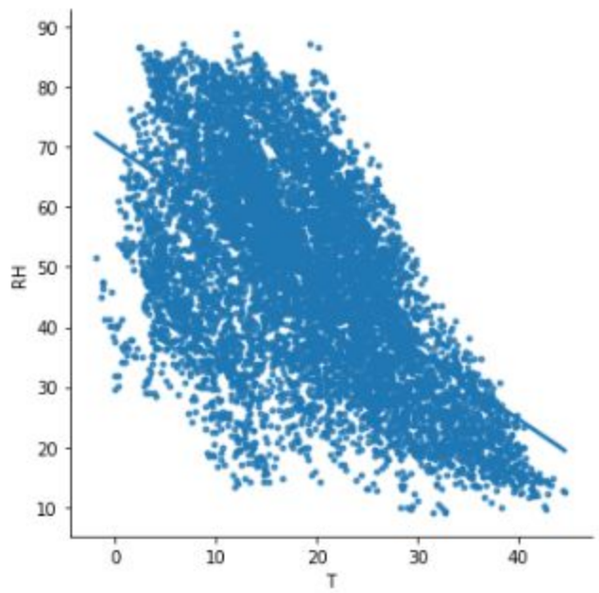

Figure 5.2.10 :Comparing linearity b/w Input features and target feature

```
: sns.lmplot(x='AH',y='RH',data=Air_Quality,markers='.')
: <seaborn.axisgrid.FacetGrid at 0x2149d4e6bc8>
```

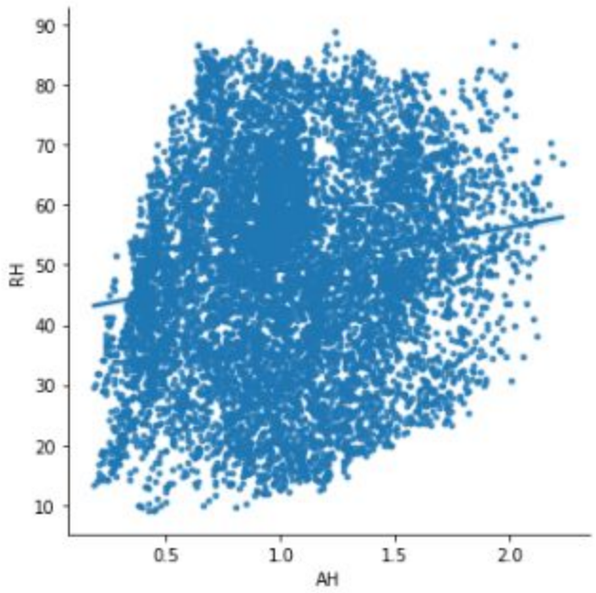

Figure 5.2.11 :Comparing linearity b/w Input features and target feature

- ➔ The plots describe the Linearity between the Input features((CO(GT),PT08.S1(CO),C6H6(GT),PT08.S2(NMHC),NOx(GT),PT08.S3(NOx), NO2(GT),PT08.S4(NO2),PT08.S5(O3),T,AH)) and Output feature (RH).
- ➔ As we observe the plot the metal oxides most affecting the target column (RH) are (CO(GT),PT08.S1(CO),,NOx(GT),,PT08.S5(O3).

CHAPTER 6

COMPARING THE MODEL WITH THE HELP OF GRAPHS:

6.1 By Barplot:

A barplot (or barchart) is one of the most common type of plot. It shows the relationship between a numerical variable and a categorical variable. For example, you can display the height of several individuals using bar chart. Barcharts are often confounded with histograms, which is highly different. (It has only a numerical variable as input and shows its distribution). A common mistake is to use barplots to represent the average value of each group. If you have several values per group, showing only the average dissimulate a part of the information. In this case, consider doing a boxplot or a violinplot. At least, you should show the number of observation per group and the confidence interval of each group. Last tip: ordering the bars often makes the chart more informative.

```
x=['Linear Regression','Decision Tree Regression','Grid SearchCV']
y=[5.998923574028056,1.1880589789763267,1.2635772369963703]
plt.bar(x,y)
plt.xlabel('Models')
plt.ylabel('Range')
plt.title('RMSE values Comparision')
plt.show()
```

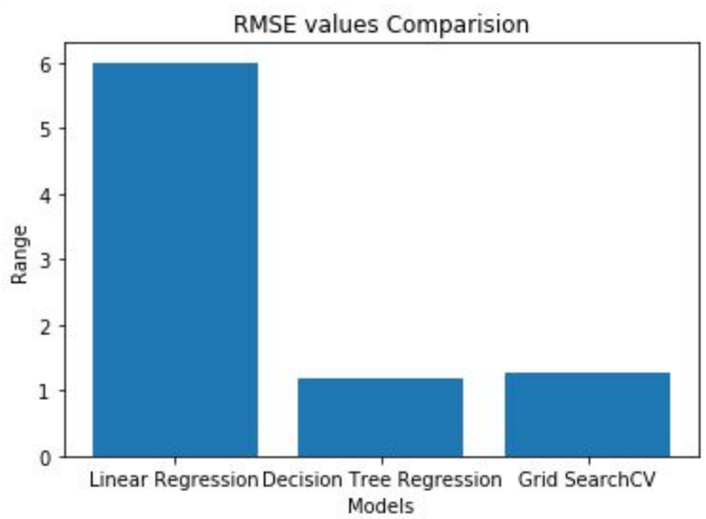

Figure 6.1 : Comparison by Barplot

6.2 By ScatterPlot:

Scatter plots are used to plot data points on horizontal and vertical axis in the attempt to show how much one variable is affected by another. Each row in the data table is represented by a marker the position depends on its values in the columns set on the X and Y axes. A third variable can be set to correspond to the color or size of the markers, thus adding yet another dimension to the plot.

```
plt.scatter(x,y)
plt.show()
```

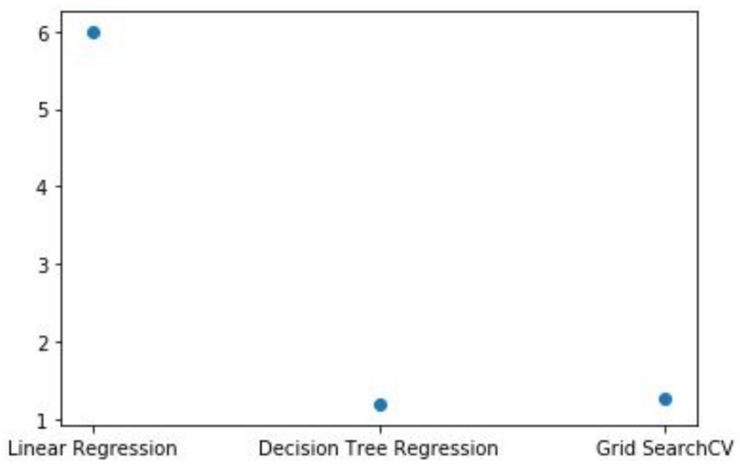

Figure 6.2 : Comparison by scatter plot

CHAPTER 7

CONCLUSION

For designing the model for predicting RH, I have applied Linear Regression, Decision Tree Regression.When tested on test data below are RMSE obtained from different algorithms:

RMSE:

-Linear Regression: 5.998923574028056

-Decision Tree: **1.1880589789763267**

-GridSearchCV**:**1.2635772369963703

As the RMSE values are between 1-10 they fit properly,the accurate RMSE score is given by **Decision tree regression** model.

CHAPTER 8

REFERENCES

1.DATASET: https://archive.ics.uci.edu/ml/datasets/Air+quality

2.WIKIPEDIA: https://en.wikipedia.org/wiki/Air_quality_index

3.GITHUB: