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

Sai Adarsh Kasula

221710305046

Under the Guidance of

Mr. Phani Kumar

Professor and HOD

Department Of Electronics and Communication Engineering GITAM
School of Technology GITAM (Deemed to be University)
Hyderabad-50232,June 2019

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 Mr. Phani Kumar, Professor and HOD, 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 Adarsh Kasula**

Date: 12-06-2020 **221710305046**

Ii

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329,

India Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled "PREDICTING AIR QUALITY BY RELATIVE HUMIDITY" is being submitted by Sai Adarsh Kasula (221710305046) 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.

Prof.Phani Kumar

Professor and HOD

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. CH. Sanjay, 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 Adarsh Kasula

221710305046

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:

vii

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).

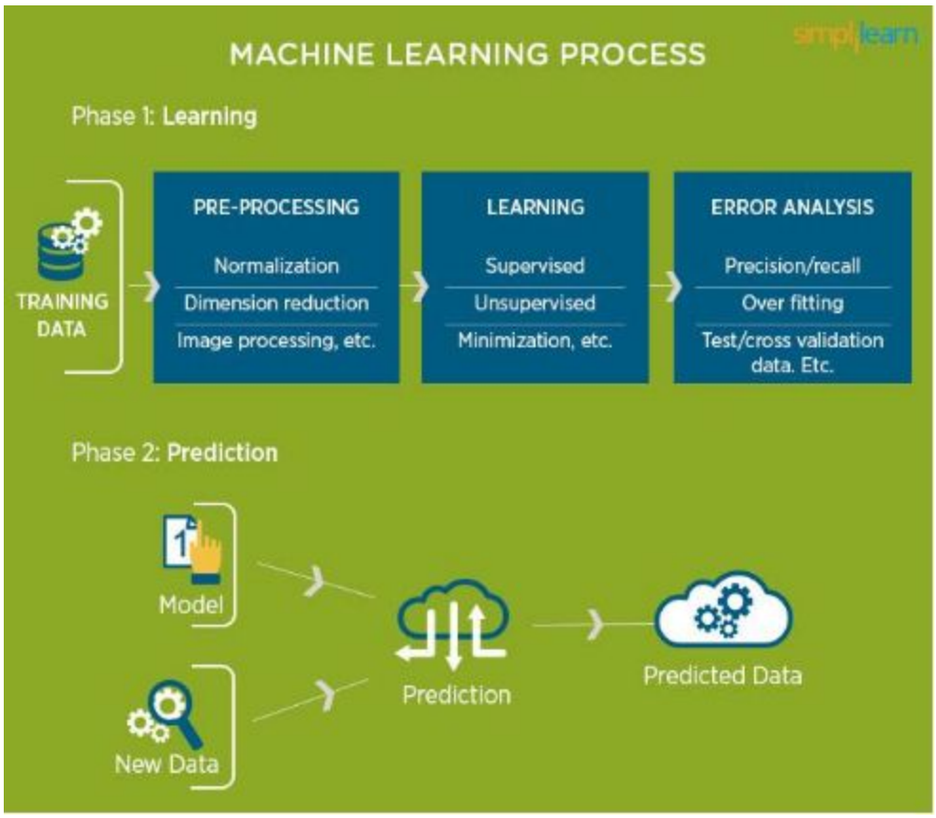
1.2 IMPORTANCE OF MACHINE LEARNING:

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



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 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. 2 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.

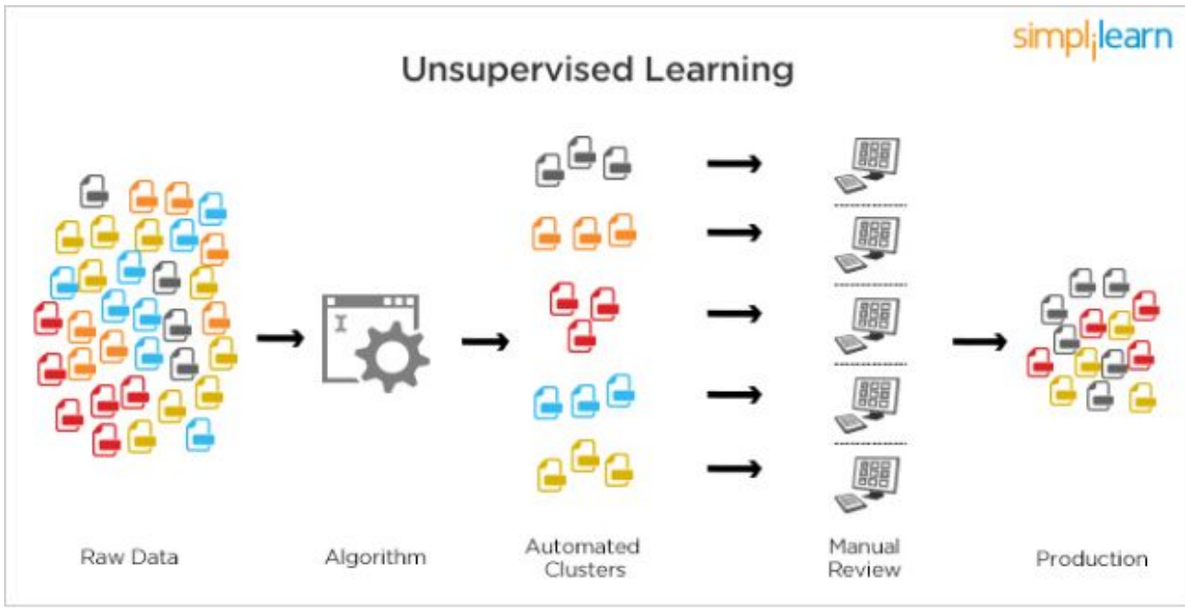


Figure 2 : Unsupervised Learning

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.

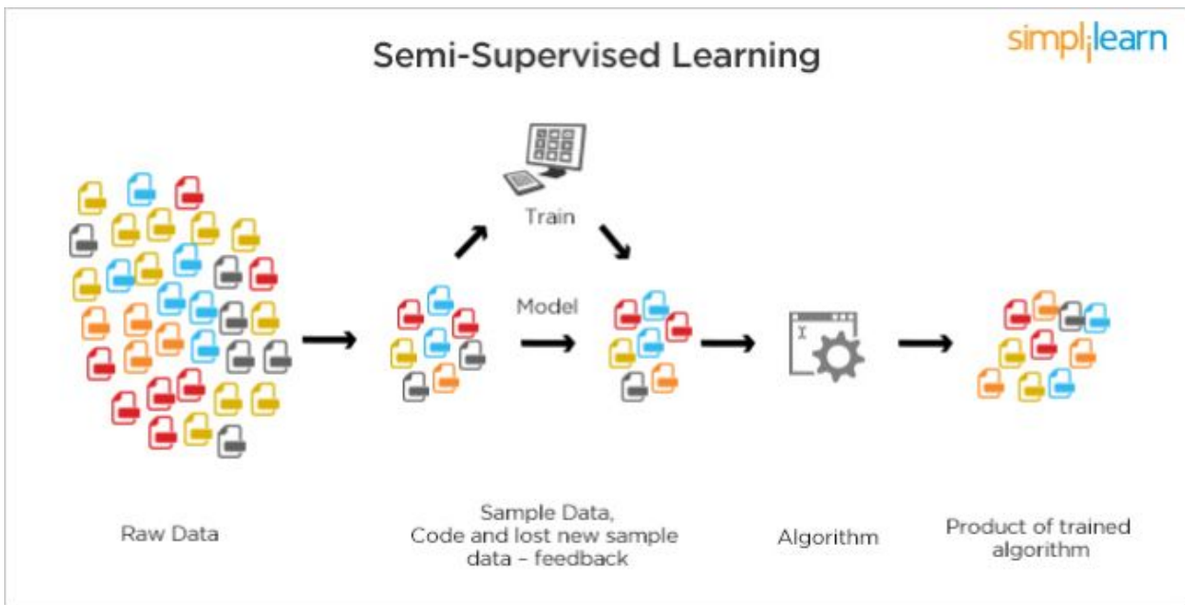


Figure 3 : Semi Supervised Learning

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 5 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.

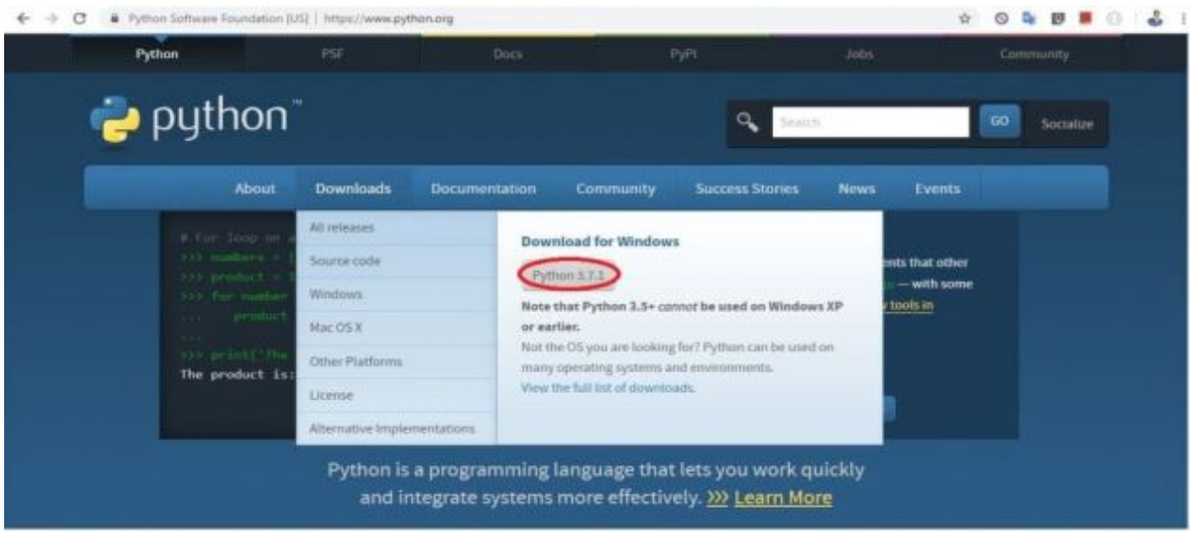
2.4.1 Installation(using python IDLE):

● 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.



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 a web browser.

● Step 2: Download python 3.4 version for (32-bits graphic 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)

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 a web browser.

● Step 2: Download python 3.4 version for (32-bits graphic 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)

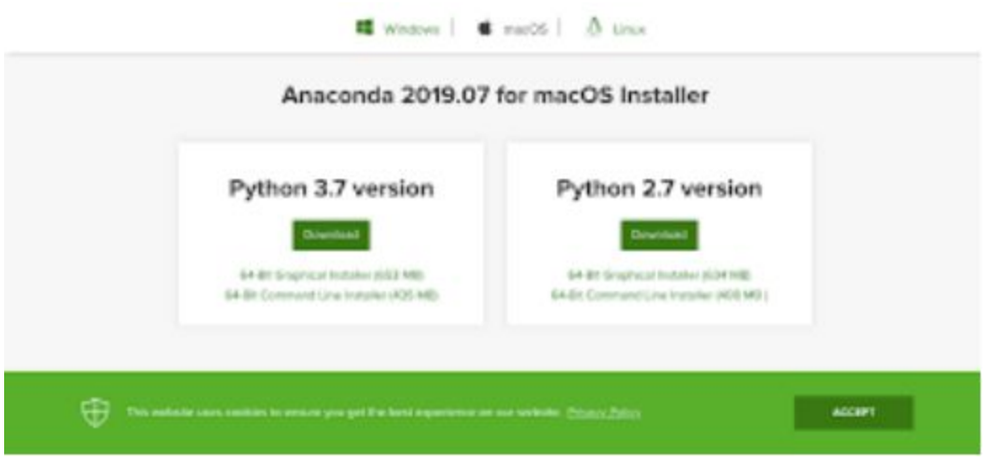


Figure 5 : Anaconda download

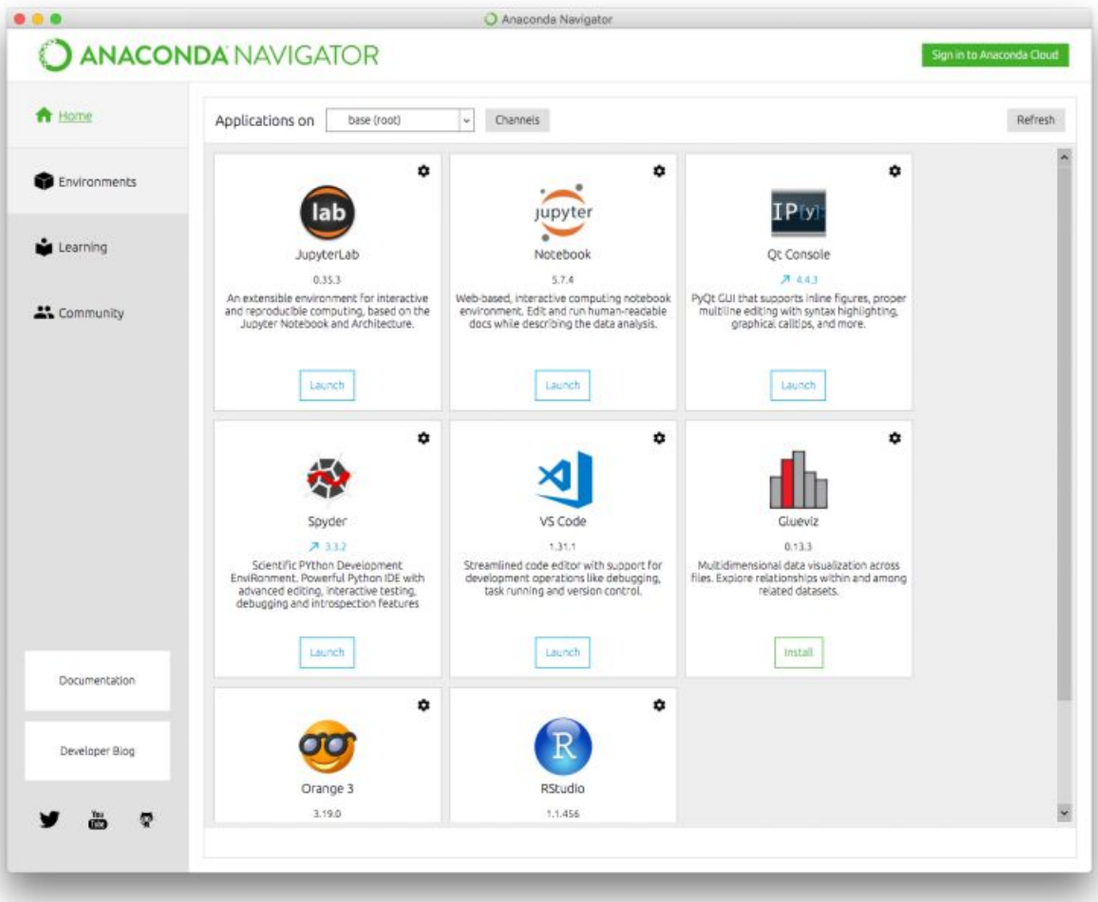


Figure 6 : Jupyter notebook

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 a 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 OOPs 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:

o We define a class in a very similar way how we define a function.
o 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 14 indented like a functions body is indented like a functions body is.

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:

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

3.2 DATA SET (including DataTypes):

The given data set consists of the following parameters:

Date	datetime64[ns]
Time	object
CO(GT)	float64
PT08.S1(CO)	float64
NMHC(GT)	int64
C6H6(GT)	float64
PT08.S2(NMHC)	float64
NOx(GT)	float64
PT08.S3(NOx)	float64
NO2(GT)	float64
PT08.S4(NO2)	float64
PT08.S5(O3)	float64
T	float64
RH	float64

AH float64

3.3 OBJECTIVE OF THE CASE STUDY:

The objective of the case study is to identify the Relative Humidity and Air Quality Prediction according to the given dataset

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 the client.

4.1.2 IMPORTING THE LIBRARIES: We have to import the libraries as per the requirement of the algorithm.

Importing the Libraries:

LOADING DATA

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

VERSION:

```
#VERSION CHECKING
print(pd.__version__)
print(np.__version__)
print(sns.__version__)
```

```
0.25.1
1.16.5
0.9.0
```

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 accessed using the dataframe. Any missing value or NaN value has to be cleaned.

Reading the dataset:

```
#reading the dataset by using read_excel
data=pd.read_excel("AirQualityUCI.xlsx")
data.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	R
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

```
data.columns
```

```
Index(['Date', 'Time', 'CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)',
```

Checking the columns:

```
data.columns
```

```
Index(['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', 'AH'],
      dtype='object')
```

Checking the information of the dataset:

```
#checking the information of dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9357 entries, 0 to 9356
Data columns (total 15 columns):
Date             9357 non-null datetime64[ns]
Time             9357 non-null object
CO(GT)           9357 non-null float64
PT08.S1(CO)      9357 non-null float64
NMHC(GT)         9357 non-null int64
C6H6(GT)         9357 non-null float64
PT08.S2(NMHC)    9357 non-null float64
NOx(GT)          9357 non-null float64
PT08.S3(NOx)     9357 non-null float64
NO2(GT)          9357 non-null float64
PT08.S4(NO2)     9357 non-null float64
PT08.S5(O3)      9357 non-null float64
T                9357 non-null float64
RH               9357 non-null float64
AH               9357 non-null float64
dtypes: datetime64[ns](1), float64(12), int64(1), object(1)
memory usage: 1.1+ MB
```

CHAPTER 5

Data Visualization

Checking the datatypes:

```
#datatypes of data
data.dtypes
```

```
Date              datetime64[ns]
Time                      object
CO(GT)                   float64
PT08.S1(CO)              float64
NMHC(GT)                   int64
C6H6(GT)                 float64
PT08.S2(NMHC)            float64
NOx(GT)                  float64
PT08.S3(NOx)             float64
NO2(GT)                  float64
PT08.S4(NO2)             float64
PT08.S5(O3)              float64
T                        float64
RH                       float64
AH                       float64
dtype: object
```

Using the Describe Function:

```
#checking the max,min,count by using describe function
data.describe()
```

	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
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000
mean	-34.207524	1048.869652	-159.090093	1.865576	894.475963	168.604200	794.872333	58.135898	1391.363266	974.951534	9.776600
std	77.657170	329.817015	139.789093	41.380154	342.315902	257.424561	321.977031	126.931428	467.192382	456.922728	43.203438
min	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000
25%	0.600000	921.000000	-200.000000	4.004958	711.000000	50.000000	637.000000	53.000000	1184.750000	699.750000	10.950000
50%	1.500000	1052.500000	-200.000000	7.886653	894.500000	141.000000	794.250000	96.000000	1445.500000	942.000000	17.200000
75%	2.600000	1221.250000	-200.000000	13.636091	1104.750000	284.200000	960.250000	133.000000	1662.000000	1255.250000	24.075000
max	11.900000	2039.750000	1189.000000	63.741476	2214.000000	1479.000000	2682.750000	339.700000	2775.000000	2522.750000	44.600000

Checking the Missing Values:

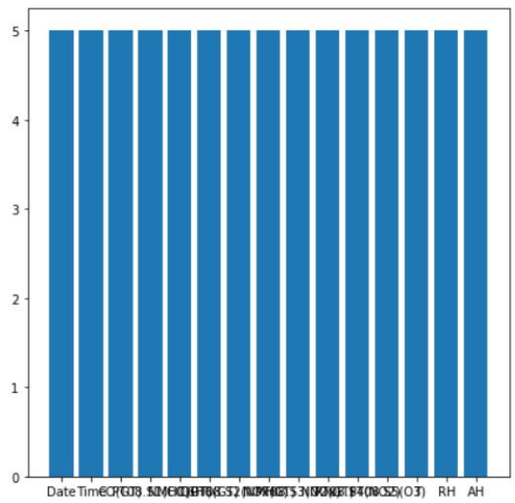
```
# missing values
data.isnull().sum()
```

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

Checking with the barplot

```
#checking the barplot of every input value
plt.figure(figsize=(7,7))
plt.bar(data.columns,height=5)
```

```
<BarContainer object of 15 artists>
```

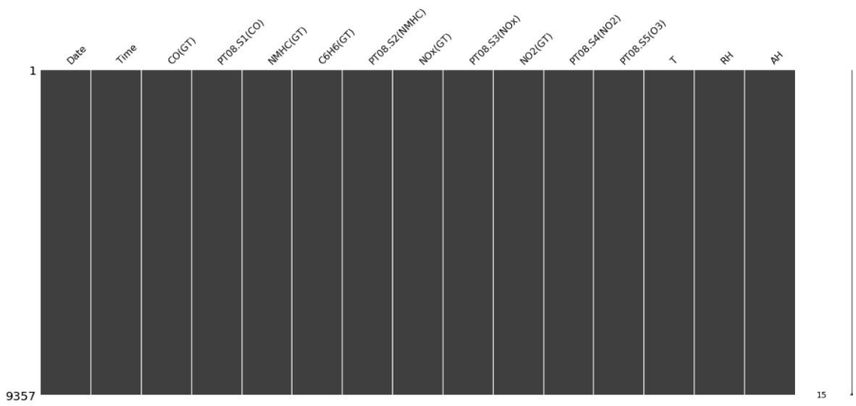


DataCleaning and DataVisualization:

DATA CLEANING AND VISULIZATION

```
## visualizing the missing values
import missingno as msno
msno.matrix(data)
```

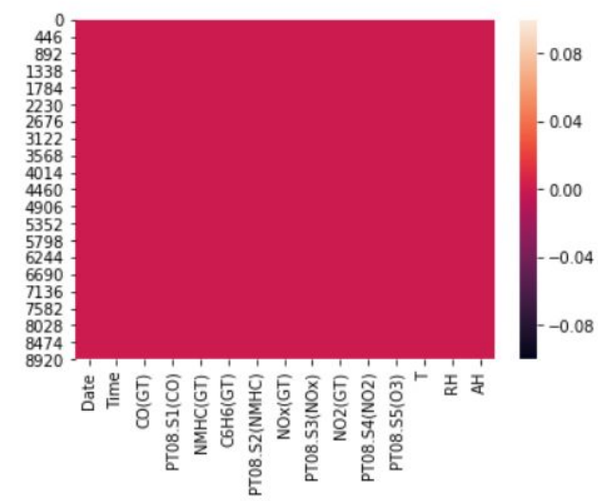
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4c6a8e548>
```



Checking the missing values by using Heatmap:

```
#checking missing values  using heatmap
sns.heatmap(data.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4c6c9dbc8>
```



Generating Plots

Visualize the data between Target and the Features

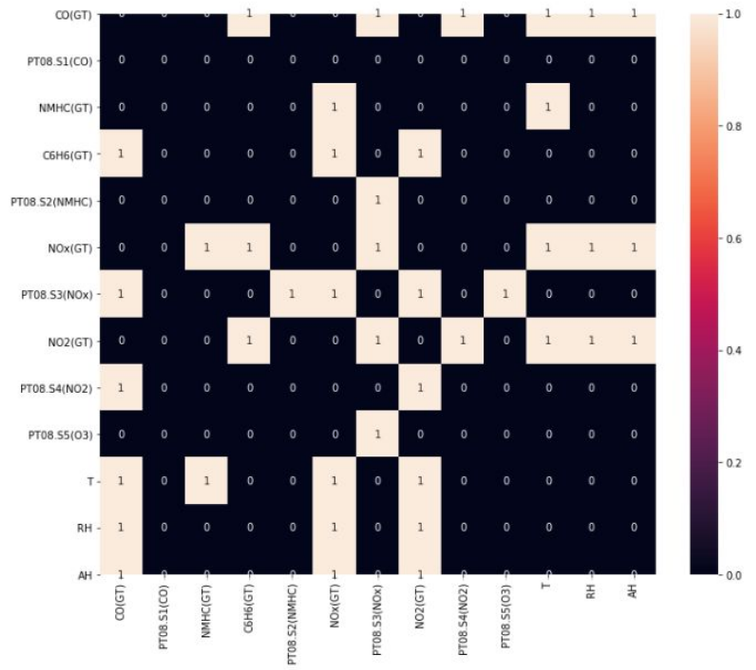
Data Correlation: Is a way to understand the relationship between multiple variables and attributes in your dataset. Using Correlation, you can get some insights such as: One or multiple attributes depend on another attribute or a cause for another attribute.

➢ A heat map (or heatmap) is a data visualization technique that shows the magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.

Considering Outliers in the DataSet:

```
#finding out corelation
plt.figure(figsize=(12,10))
sns.heatmap(data.corr()<0,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4c6d15a48>
```



Replacing the Nan values using Replace function:

```
d=data[data[data.columns[2:15]]<0]
```

```
data.replace(d,np.nan,inplace=True)
```

```
data
```

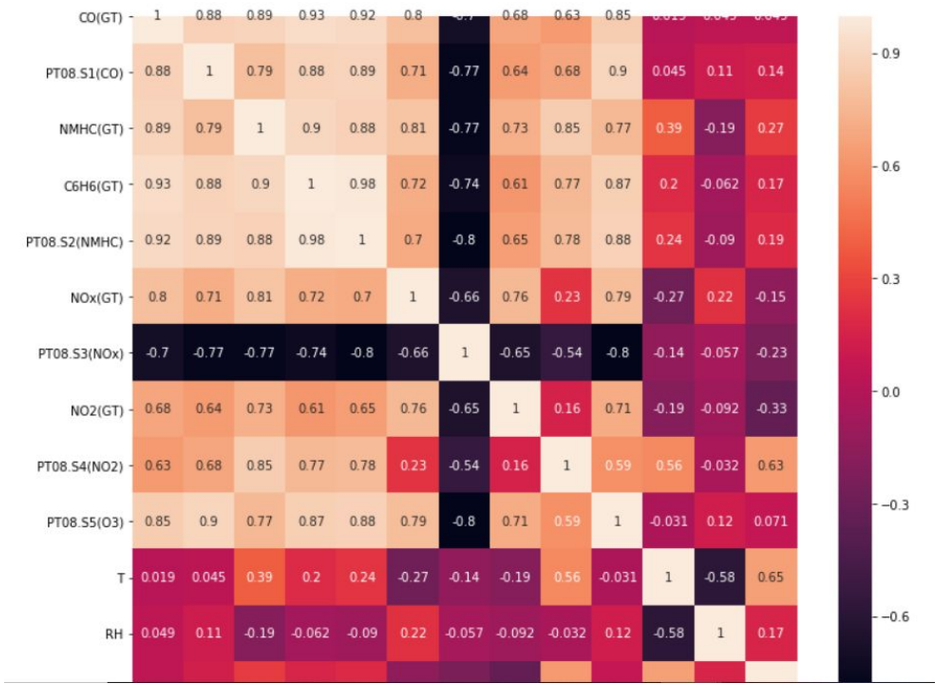
Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	A
18:00:00	2.6	1360.00	150.0	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.600	48.875001	0.7577
19:00:00	2.0	1292.25	112.0	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.700000	0.7254
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	0.7502
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	0.7867
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	0.7887
...	...	...	...	...	...	...	...	...	...	...	...	...	
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	0.7568
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	0.7118
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	0.6406
13:00:00	2.1	1002.50	NaN	9.547187	960.50	234.5	701.50	155.7	1041.00	769.75	28.325	13.550000	0.5138
14:00:00	2.2	1070.75	NaN	11.932060	1047.25	265.2	654.00	167.7	1128.50	816.00	28.500	13.125000	0.5028

15 columns

Heatmap Correleation without Outliers

```
#after removing the outliers from the dataset
plt.figure(figsize=(12,10))
sns.heatmap(data.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4c71ae048>
```



The Correlation Graph is used to checking Correlation between the input values And most of the correlation of the values is more than 0.7

Now Checking the Missing Values:

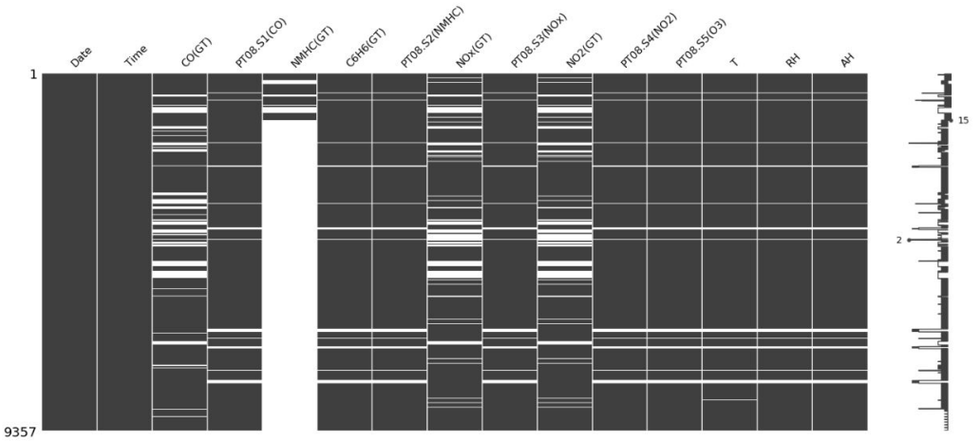
```
#after removing the outliers now finding out the missing values
data.isnull().sum()
```

```
Date              0
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               380
RH              366
AH              366
dtype: int64
```

Checking The MissingValues Using Missingno Matrix:

```
#using missing no matrix
import missingno as msno
msno.matrix(data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4c6aa7288>
```



Now changing the missing values replacing it with median of that column:

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.

```
##now filling the missing values by the median of the pativular column
data["CO(GT)"].fillna((data["CO(GT)"].median()), inplace=True)
data["PT08.S1(CO)"].fillna((data["PT08.S1(CO)"].median()), inplace=True)
data["NMHC(GT)"].fillna((data["NMHC(GT)"].median()), inplace=True)
data["C6H6(GT)"].fillna((data["C6H6(GT)"].median()), inplace=True)
data["PT08.S2(NMHC)"].fillna((data["PT08.S2(NMHC)"].median()), inplace=True)
data["NOx(GT)"].fillna((data["NOx(GT)"].median()), inplace=True)
data["PT08.S3(NOx)"].fillna((data["PT08.S3(NOx)"].median()), inplace=True)
data["NO2(GT)"].fillna((data["NO2(GT)"].median()), inplace=True)
data["PT08.S4(NO2)"].fillna((data["PT08.S4(NO2)"].median()), inplace=True)
data["PT08.S5(O3)"].fillna((data["PT08.S5(O3)"].median()), inplace=True)
data["T"].fillna((data["T"].median()), inplace=True)
data["RH"].fillna((data["RH"].median()), inplace=True)
data["AH"].fillna((data["AH"].median()), inplace=True)
```

Here We can see the values are been replaced with the values of the median

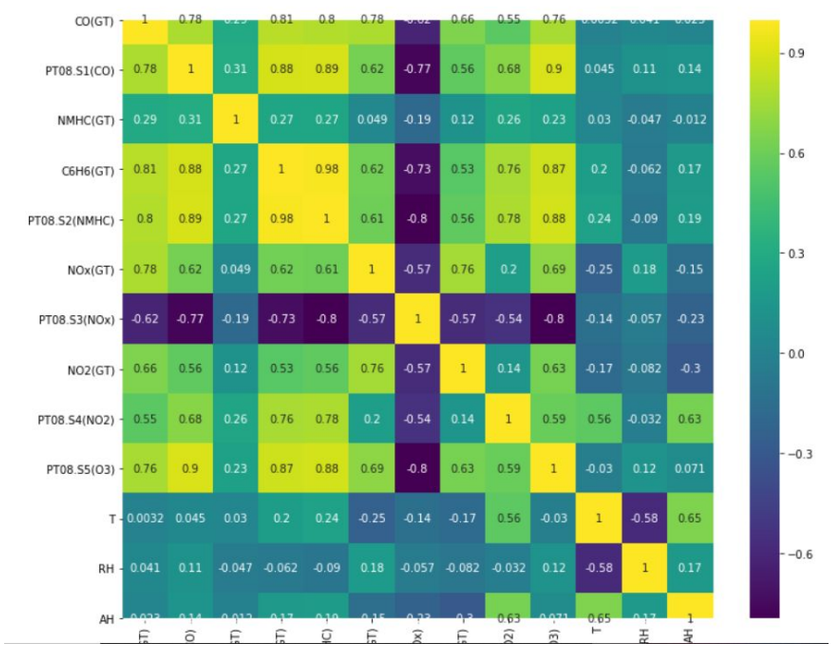
After remvoving NAN values with median:

```
data.isnull().sum()
```

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

Checking the Correlation After Removing the Outlier:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4b89a6488>
```



Here we are checking the correlation when there is no outliers present in dataset

CHAPTER 6

TRAINING THE MODEL

Training The Model:

TRAINING THE MODEL

After loading and cleaning the data the next step is to Train the dataset for prediction The methods going to train the model is

- LinearREgression
- RandomForest
- DecisionTree

```
#first consider the input and the output columns
X=data[col_].drop("RH",1)
y=data["RH"]
```

Scaling the Data:

```
#Applying Standard Scaler in the model
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

```
#fitting the data set
X_std=sc.fit_transform(X)
X_std
```

```
array([[ 0.38602857,  1.22932554, -0.10024072, ...,  0.6322262 ,
        -0.54740851, -0.67346846],
       [-0.06750209,  0.91110658, -0.66694332, ..., -0.12335538,
        -0.58218236, -0.7549785 ],
       [ 0.0836748 ,  1.42659781, -1.02486074, ...,  0.13703556,
        -0.74446035, -0.69245194],
       ...,
       [ 0.23485168,  0.2053885 , -0.10024072, ...,  0.18309981,
         0.99423249, -0.96929461],
       [ 0.00808636, -0.44983726, -0.10024072, ..., -0.64157814,
         1.15940833, -1.28956845],
       [ 0.0836748 , -0.12926982, -0.10024072, ..., -0.52321862,
         1.17969316, -1.31751337]])
```

Why is Scaling Done?:

Feature Scaling or Standardization: It is a step of Data Pre Processing which is applied to independent variables or features of data. It basically helps to normalise the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

SPLITTING OF DATA

Splitting Of Data:

● 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

Splitting the Data into Training and Testing Data:

```
#split the data into train and test with test size and 30% and train size as 70%
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X_std,y,test_size=0.3, random_state=30)
```

```
X_train
```

```
array([[-0.89897495,  1.0684547 , -0.10024072, ...,  0.96491242,
        -1.34430941, -0.62268747],
       [-0.59662118, -0.21381508, -0.10024072, ..., -0.5475303 ,
        -0.80821245,  0.12561909],
       [ 2.57809339,  3.28072234,  2.67361934, ...,  1.6865856 ,
        -0.68650392, -0.27130152],
       ...,
       [ 0.38602857,  1.17531051,  2.06217706, ...,  0.08457351,
        -0.54451068,  0.12689825],
       [-0.21867897, -0.40873887, -0.10024072, ..., -0.26346746,
         0.10170349,  2.26124202],
       [ 0.23485168,  0.14197956, -0.10024072, ...,  0.07497679,
        -0.07506363,  0.22315496]])
```

CHAPTER 7

MODEL BUILDING AND EVALUATION

7.1 Brief about the algorithms used

LinearRegression:

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

```
#importing LinearRegression algorithm
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

```
#fitting the model into th =e training data
lr_model=lr.fit(X_train,y_train)
```

```
#finding out intercept and the slope of the model
print('Intercept:',lr_model.intercept_)
print('---------------------------------')
print('Slope:')
list(zip(X.columns.tolist(),lr_model.coef_))
```

```
Intercept: 49.222328234307376
---------------------------------
Slope:
```

```
[('CO(GT)', -0.9173999549325367),
 ('PT08.S1(CO)', 2.3176323277413378),
 ('NMHC(GT)', -0.19916449611116532),
 ('C6H6(GT)', -4.399691535811749),
 ('PT08.S2(NMHC)', -0.6281084396370147),
 ('NOx(GT)', 2.939677232219961),
 ('PT08.S3(NOx)', -0.1972003619944458),
 ('NO2(GT)', -1.4497967392955138),
 ('PT08.S4(NO2)', 4.672327404830174),
 ('PT08.S5(O3)', -0.4644334570900912),
 ('T', -19.76570699862395),
 ('AH', 13.367901823570152)]
```

Prediction Of Data:

Predicting the data by fitting the training and testing data

```
('AH', 13.367901823570152)]
```

```
]: y_test_predlr=lr.predict(X_test)# predicting the test data
   y_test_predlr
```

```
]: array([61.35700099, 45.24908739, 68.19131023, ..., 40.28167299,
          31.071772  , 65.27335384])
```

```
]:
```

```
]: y_train_predlr=lr.predict(X_train)# predicting the traing data
   y_train_predlr
```

```
]: array([65.89778067, 66.82300104, 66.21295986, ..., 65.67828111,
          80.41256012, 54.51340966])
```

RMSE:

```
: def rmse(predictions, targets):
      return np.sqrt(((predictions - targets) ** 2).mean())
```

Finding out the RMSE,R2_Score,Mae,Mse values:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
rmse=np.sqrt(mean_squared_error(y_test,y_test_predlr))      #calculate rmse for testing data
a=print('Baseline RMSE of model:',rmse)
```

```
Baseline RMSE of model: 6.090169023141622
```

```
rmse=np.sqrt(mean_squared_error(y_train,y_train_predlr))      #calculate rmse for training data
a1=print('Baseline RMSE of model:',rmse)
```

```
Baseline RMSE of model: 5.944883527604205
```

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

```
R^2: 0.8707115916780975
Adjuted R^2: 0.8701565072774311
MAE: 4.638266119257476
MSE: 37.090158730433785
```

Decision Tree Regression

Decision Tree Algorithm:

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*.

Importing the DecisionTreeRegressor Packages:

DECISION TREE REGRESSION

```
from sklearn.tree import DecisionTreeRegressor          #Decision tree regression model
from sklearn.model_selection import cross_val_score   #import cross validation score pack
```

Predicting the Data:

```
dtr=DecisionTreeRegressor()
dtr_model=dtr.fit(X_train,y_train)
```

```
y_test_preddr=dtr_model.predict(X_test)  #predicting the values for the test data
y_test_preddr
```

```
array([64.625     , 41.27500057, 81.64999962, ..., 33.97499967,
       30.5       , 82.54999924])
```

```
y_train_preddr=dtr_model.predict(X_train)  #predicting the values for training data
y_train_preddr
```

```
array([78.67499924, 80.07500076, 63.87500095, ..., 69.20000076,
       87.17499924, 55.52499962])
```

:

Finding out the RMSE,R2_Score,Mae,Mse values of DecisionTreeRegressor:

```
#calculate RMSE
b=print('RMSE of Decision Tree Regression:',np.sqrt(mean_squared_error(y_test_preddr,y_test)))  #for test data
```

```
RMSE of Decision Tree Regression: 1.5686452391446677
```

```
#calculate RMSE
b1=print('RMSE of Decision Tree Regression:',np.sqrt(mean_squared_error(y_train_preddr,y_train))) # for traing data
```

```
RMSE of Decision Tree Regression: 5.921657016626916e-08
```

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

```
R^2: 0.9914227045785542
Adjuted R^2: 0.9913858789810381
MAE: 0.9250890317409216
MSE: 2.4606478862912318
```

Random Forest Regression in Python

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but multiple decision trees. In the case of a classification problem, the final output is

taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation.

RANDOM FOREST REGRESSOR

```
: from sklearn.ensemble import RandomForestRegressor          #import random
  rf=RandomForestRegressor()
```

```
: rf_model=rf.fit(X_train,y_train)
```

Fitting and Predicting Data:

```
]: rf_model=rf.fit(X_train,y_train)
   rf_model
```

```
]: RandomForestRegressor()
```

```
]: y_test_predrf=rf_model.predict(X_test)  #predicting the values for the test data
   y_test_predrf
```

```
]: array([64.96825029, 41.19275033, 80.64500079, ..., 33.9434998 ,
          31.22508324, 82.88099945])
```

```
]: y_train_predrf=rf_model.predict(X_train)  #predicting the values for training data
   y_train_predrf
```

```
]: array([79.06699965, 79.80550053, 63.9867508 , ..., 69.32475045,
          84.88499941, 55.60974975])
```

Finding out the RMSE,R2_Score,Mae,Mse values of RandomForest:

```
: #Calculate RMSE
  c=print('RMSE of predicted RH in RF model:',np.sqrt(mean_squared_error(y_test,y_test_predrf))) # for test data

  RMSE of predicted RH in RF model: 0.9056233118693106

: #Calculate RMSE
  c1=print('RMSE of predicted RH in RF model:',np.sqrt(mean_squared_error(y_train,y_train_predrf))) # for test data

  RMSE of predicted RH in RF model: 0.27878870724863597

: ## checking testing r2_score,mean_absoulte_error_mean_squaed_error

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

  R^2: 0.997141118966452
  Adjuted R^2: 0.9971288447008338
  MAE: 0.41291825738032495
  MSE: 0.8201535830011385
```

GridSearchCv

GridSearchCV:

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.

Importing and Fitting the GridSearchPackage:

```
: 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)
```

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

```
: GridSearchCV(cv=5, estimator=DecisionTreeRegressor(random_state=0),
               param_grid={'min_samples_split': range(2, 10)},
               scoring=make_scorer(rmse))
```

```
: g_cv.best_params_
```

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

Predicting The Values:

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

```
array([63.5062499 , 41.11666695, 80.13928604, ..., 33.18749984,
       31.36249995, 82.71874952])
```

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

```
array([77.45416673, 80.33125019, 63.77142865, ..., 69.02857154,
       83.72000008, 54.85625017])
```

Calculating the Rmse,R2 Score,Mae,Mse:

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

```
RMSE of GridSearch Regression: 1.3475439415357604
```

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

```
RMSE of GridSearch Regression: 0.6951434233828186
```

```
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: 0.9936702469227052
Adjuted R^2: 0.9936430708808707
MAE: 0.9459077480663457
MSE: 1.815874674369733
```

CHAPTER 8

COMPARING THE MODEL WITH THE HELP OF GRAPHS:

Comparing the Rmse Value:

By Histogram:

A histogram is a graphical display of data using bars of different heights. In a histogram, each bar groups numbers into ranges. Taller bars show that more data falls in that range. A histogram displays the shape and spread of continuous sample data.

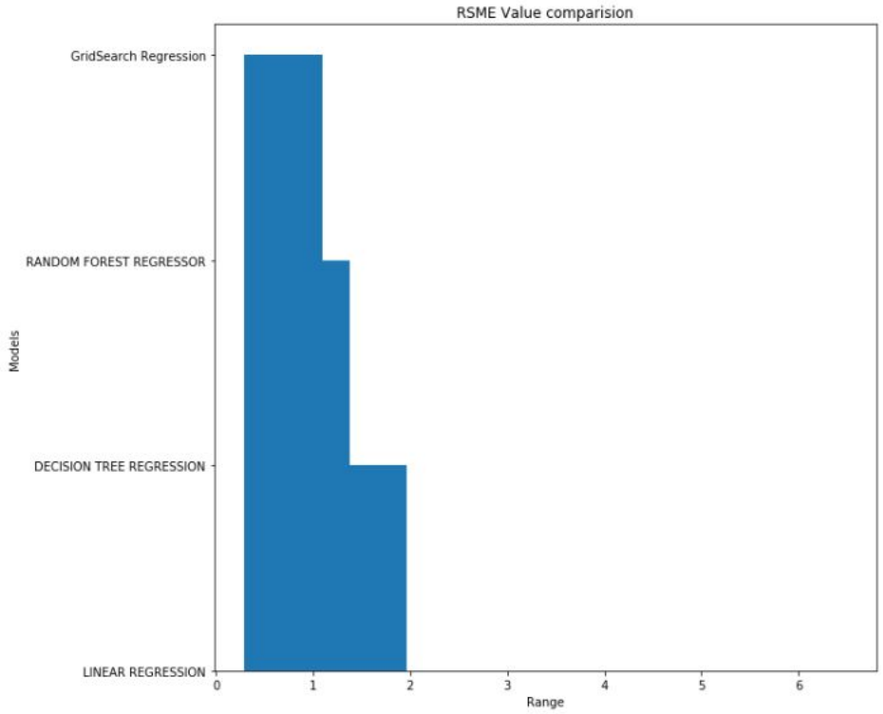
What is a histogram used for?

A histogram is a type of graph that has wide applications in statistics. Histograms provide a visual interpretation of numerical data by indicating the number of data points that lie within a range of values. ... The higher that the bar is, the greater the frequency of data values in that bin

comparing the RMSE VALUE

```
5]: x=[6.090169023141622,1.5637752690060125,0.977452852415959,0.6951434233828186]
    y=["LINEAR REGRESSION","DECISION TREE REGRESSION","RANDOM FOREST REGRESSOR","GridSearch Regression"]
    plt.figure(figsize=(10,10))
    plt.bar(x,y)
    plt.xlabel("Range")
    plt.ylabel("Models")
    plt.title("RSME Value comparision")

5]: Text(0.5, 1.0, 'RSME Value comparision')
```



By Barplot:

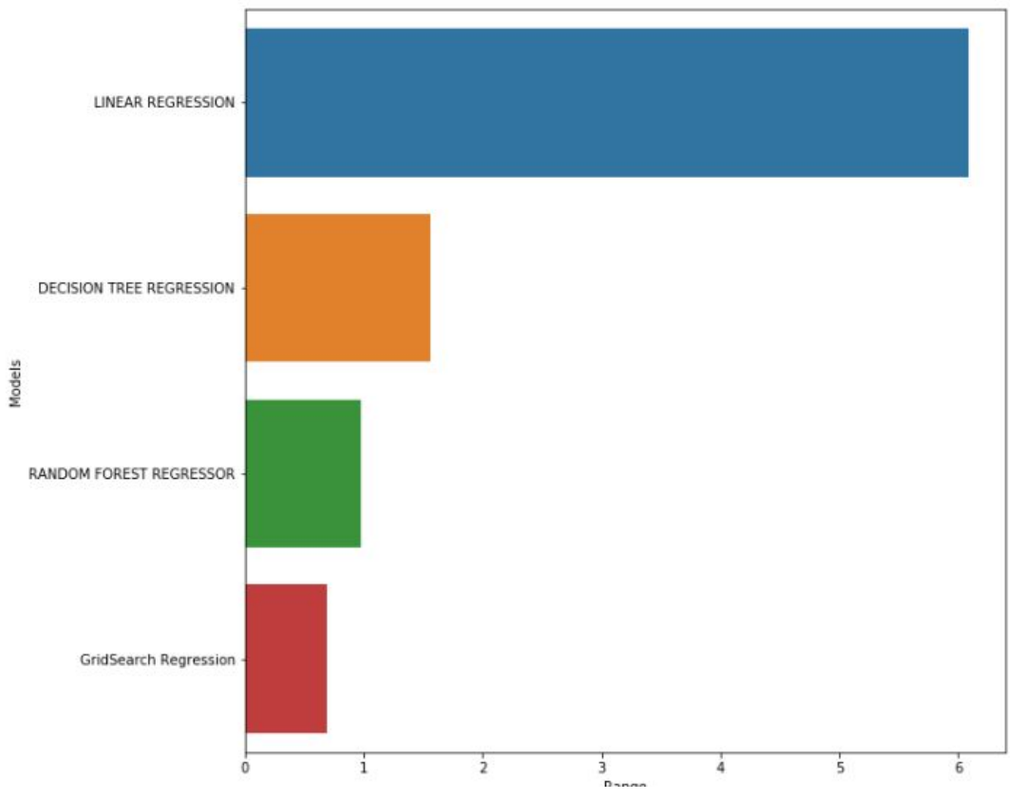
A barplot (or barchart) is one of the most common type of plot. It shows the relationship between a numerical variable and a categorical variable. For example, you can display the height of several individuals using bar chart. Barcharts are often confounded with

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.

```
]: #barplot of the rmse values
   plt.figure(figsize=(10,10))
   plt.xlabel("Range")
   plt.ylabel("Models")
   sns.barplot(x,y)
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1e0ca4ef648>
```



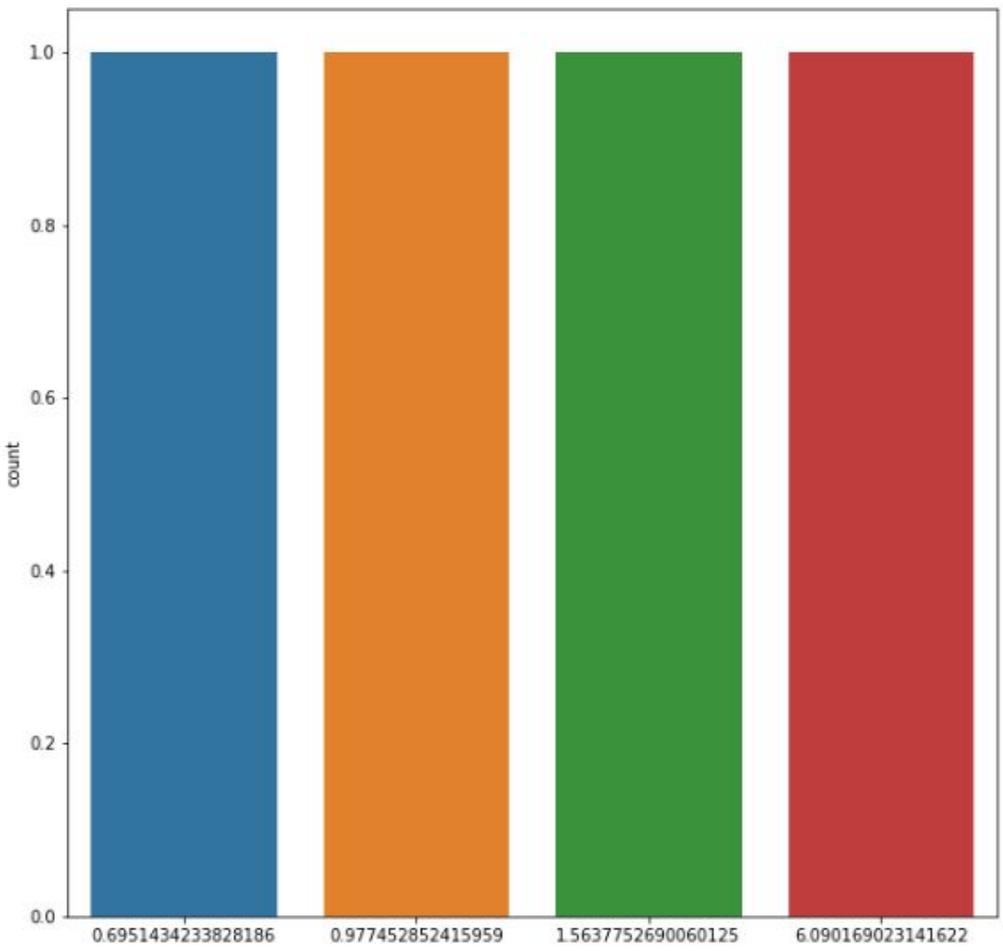
By CountPlot:

What is Countplot in Python?

Seaborn is a module in Python that is built on top of matplotlib that is designed for statistical plotting. ... One of the plots that seaborn can create is a countplot. A countplot is kind of likea histogram or a bar graph for some categorical area.

```
7]: #countplot of the rmse values
    plt.figure(figsize=(10,10))
    sns.countplot(x)
```

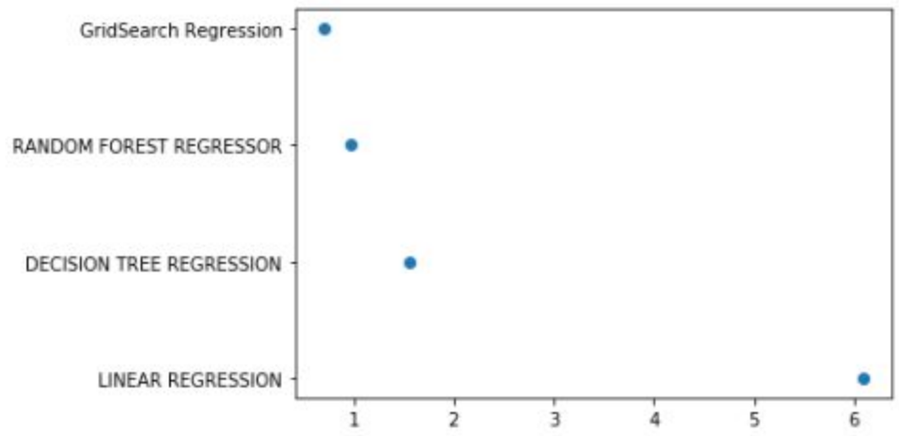
```
7]: <matplotlib.axes._subplots.AxesSubplot at 0x1e0c9713f88>
```



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.

```
#by visualizing with scatter plot
plt.scatter(x,y)
plt.show()
```



CHAPTER 9

Conclusion

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

RMSE:

-Linear Regression: 6.090169023141622

-Decision Tree: 1.5637752690060125

-Random Forest: 0.977452852415959

GridSearchCV:0.695143423382818

Final Conclusion:

First by importing the libraries ,reading the data set,cleaning the data(whether there is missing values or outliers present or not),visualizing the data by applying correlation into the dataset as wells as checking every input column with the output column ,splitting the data into train and test data and applying algorithms to the model.

After applying the various kinds of algorithm(Linear Regression,Decision Tree Regressor ,Random Forest Regressor and Grid Search CV) into the model and comparing the models with one another and checking their scores(RMSE,R2 Score,MAE,MSE ,Adjusted R2 Score) .I finally came to conclusion that the best algorithm to predict the Air Quality with respect to Relative Humidity is **GRID SEARCH CV with an RMSE SCORE of 0.695143423382818.**This score is best fitted for the model.

CHAPTER 10

References

Dataset:

(https://archive.ics.uci.edu/ml/datasets/Air+quality)

Information:

https://en.wikipedia.org/wiki/Machine_learning