MACHINE LEARNING

(Weather forecasting)

Summer Internship Report Submitted in partial fulfilment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Shraavya G kunch

221710313025

Under the Guidance of

Assistant Professor

Department of Computer Science Engineering GITAM
School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
July 2020

DECLARATION

I submit this industrial training work entitled "WEATHER FORCASTING" 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 and 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 SHRAAVYA G KUNCH
DATE: 221710313052

GITAM (DEEMED TO BE UNIVERSITY)
Hyderabad-502329, India
Dated:

<u>CERTIFICATE</u>

This is to certify that the Industrial Training Report entitled **“weather forecasting”** is being submitted by Shraavya G Kunch (221710313052) in partial fulfilment 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 2019-20

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

Mr.	**Dr.**
Assistant Professor	Professor and HOD
Department of CSE	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 **Dr.,** 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 an internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** 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.

Shraavya G Kunch

221710313052

ABSTRACT

This project explores two machine learning models for weather prediction namely Linear regression and Time Series based Recurrent Neural Network (RNN). It also discussed the steps followed to achieve results. RNN using time series along with a neural network is used to predict the weather. The results of these models are analysed and compared on the basis of Root Mean Squared Error between the predicted and actual values. For weather Forecasting, this paper uses Pandas, Numpy, Keras, Matplotlib. It is found that Time Series based RNN does the best job of predicting the weather.

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 selfdriving 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 analyse 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.

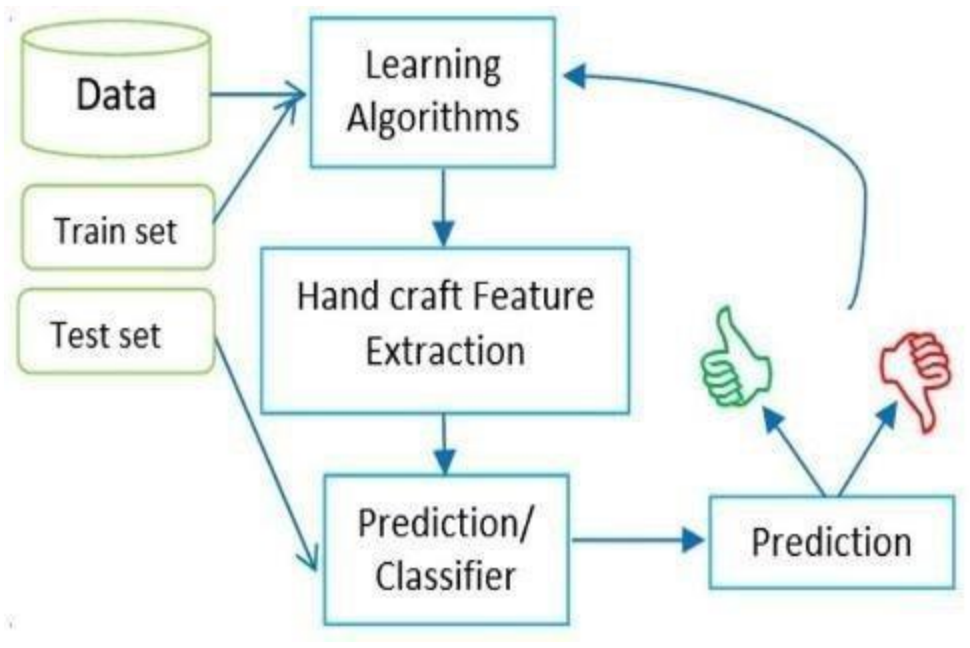


Figure 1.2 1 : The Process Flow

.

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 analyse 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 analysing 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.

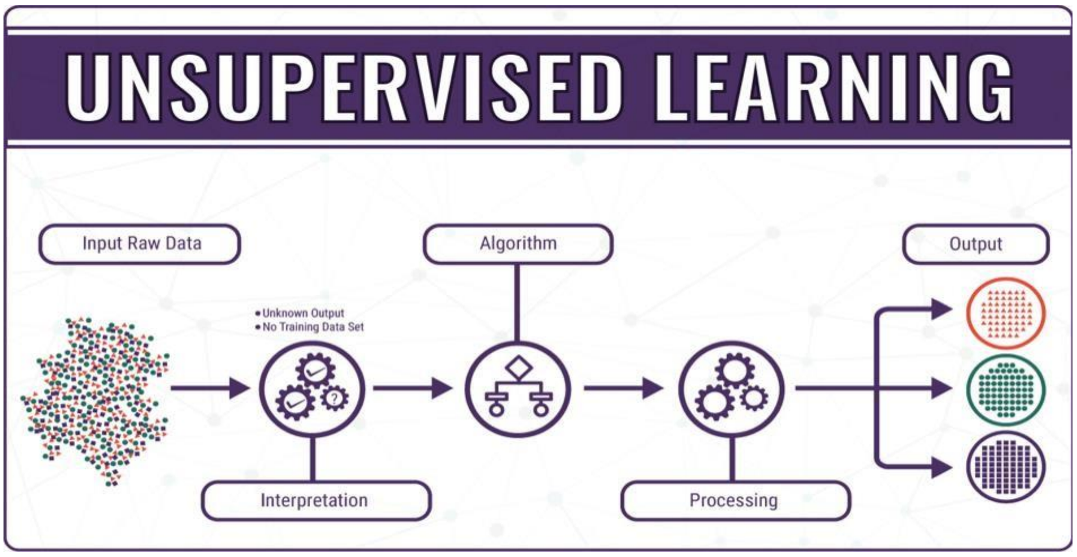


Figure 1.4.2 1: unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbour 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 labelled and unlabelled data for training. In a typical scenario, the algorithm would use a small amount of labelled data with a large amount of unlabelled data.

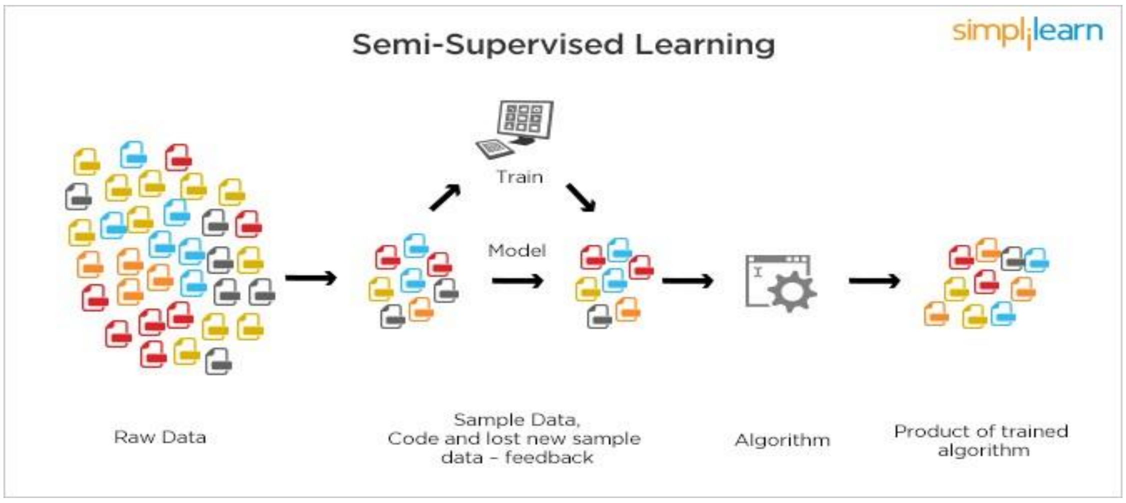


Figure 1.4.2.2: Semi Supervised Learning

CHAPTER 2
INFORMATION ABOUT DEEP LEARNING

2.1 Importance of Deep Learning:

Deep learning recently returned to the headlines when Google’s Alpha Go program crushed Lee Solo, one of the highest-ranking Go players in the word. Google has invested heavily in deep learning and Alpha Go is just their latest deep learning project to make the news. Google's search engine, voice recognition system and self-driving cars all rely heavily on deep learning. They've used deep learning networks to build a program that picks out an alternative still from a YouTube video to use as a thumbnail. Late last year Google announced smart reply, a deep learning network that writes short email responses for you. Deep learning is clearly powerful, but it also may seem somewhat mysterious.

2.2 Uses of Machine Learning:

The value of machine learning technology has been recognized by companies across several industries that deal with huge volumes of data. By leveraging insights obtained from this data, companies are able work in an efficient manner to control costs as well as get an edge over their competitors. This is how some sectors / domains are implementing machine learning

Financial Services

Companies in the financial sector are able to identify key insights in financial data as well as prevent any occurrences of financial fraud, with the help of machine learning technology. The technology is also used to identify opportunities for investments and trade. Usage of cyber surveillance helps in identifying those individuals or institutions which are prone to financial risk, and take necessary actions in time to prevent fraud.

Marketing and Sales

Companies are using machine learning technology to analyze the purchase history of their customers and make personalized product recommendations for their next purchase. This ability to capture, analyze, and use customer data to provide a personalized shopping experience is the future of sales and marketing.

Government

Government agencies like utilities and public safety have a specific need FOR Ml, as they have multiple data sources, which can be mined for identifying useful patterns and insights. For example, sensor data can be analyzed to identify ways to minimize costs and increase efficiency. Furthermore, ML can also be used to minimize identity thefts and detect fraud.

Healthcare

With the advent of wearable sensors and devices that use data to access health of a patient in real time, ML is becoming a fast-growing trend in healthcare. Sensors in wearable provide real-time patient information, such as overall health condition, heartbeat, blood pressure and other vital parameters. Doctors and medical experts can use this information to analyze the health condition of an individual, draw a pattern from the patient history, and predict the occurrence of any ailments in the future. The technology also empowers medical experts to analyze data to identify trends that facilitate better diagnoses and treatment.

Transportation

Based on the travel history and pattern of traveling across various routes, machine learning can help transportation companies predict potential problems that could arise on certain routes, and accordingly advise their customers to opt for a different route. Transportation firms and delivery organizations are increasingly using machine learning technology to carry out data analysis their customers make smart decisions when they travel. and data modeling to make informed decisions and help

Oil and Gas

This is perhaps the industry that needs the application of machine learning the most. Right from analyzing underground minerals and finding new energy sources to streaming oil distribution, ML applications for this industry are vast and are still expanding.

2.3 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 3
PYTHON

Basic programming language used for machine learning is: PYTHON

3.1 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.

3.2 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

3.3 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.

3.4 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.

3.4.1 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.

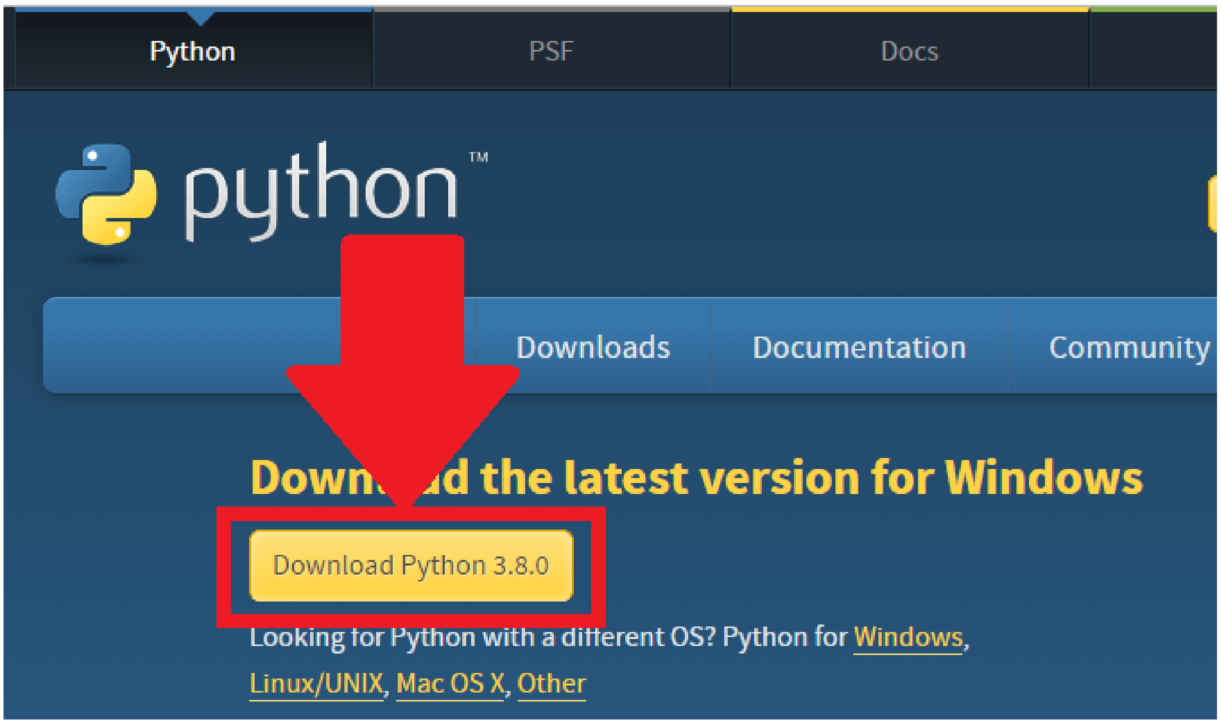

Python
PSF
Docs
python™
Downloads
Documentation
Community
Download the latest version for Windows
Download Python 3.8.0
Looking for Python with a different OS? Python for Windows,
Linux/UNIX, Mac OS X, Other

Figure 3.4.1 1: Python download

3.4.2 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 that 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-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)

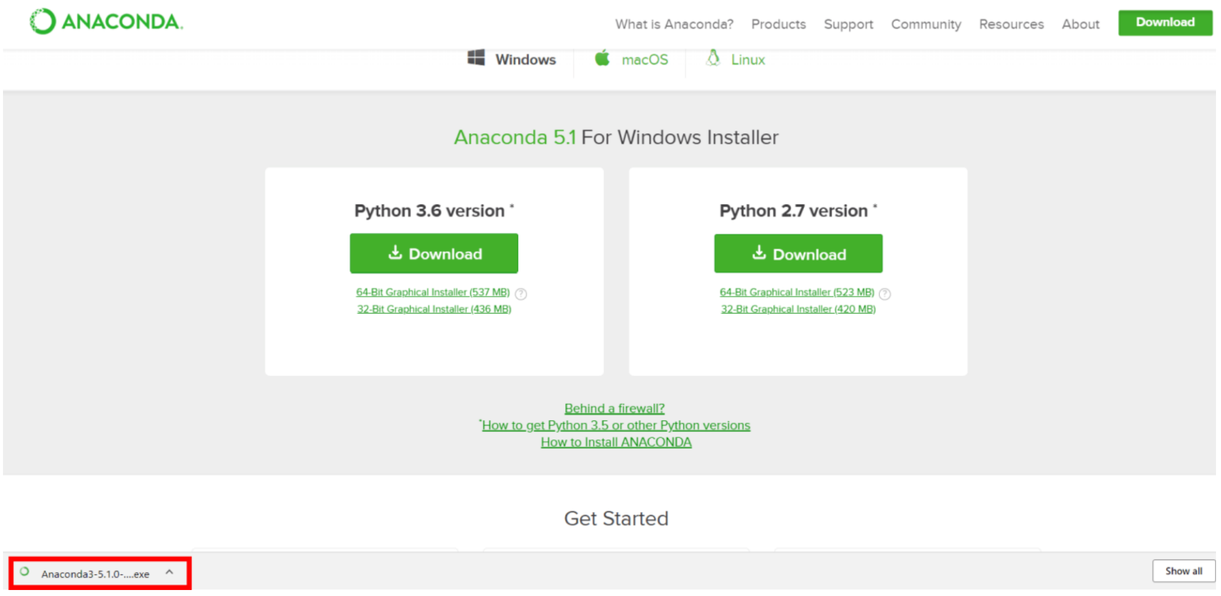

ANACONDA.
What is Anaconda? Products Support Community Resources About Download
Windows macOS Linux
Anaconda 5.1 For Windows Installer
Python 3.6 version *
Download
64-Bit Graphical Installer (537 MB)
32-Bit Graphical Installer (436 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (523 MB)
32-Bit Graphical Installer (420 MB)
Behind a firewall?
*How to get Python 3.5 or other Python versions
How to Install ANACONDA
Get Started
Anaconda3-5.1.0-....exe
Show all

Figure 3.4.2 1 : Anaconda download

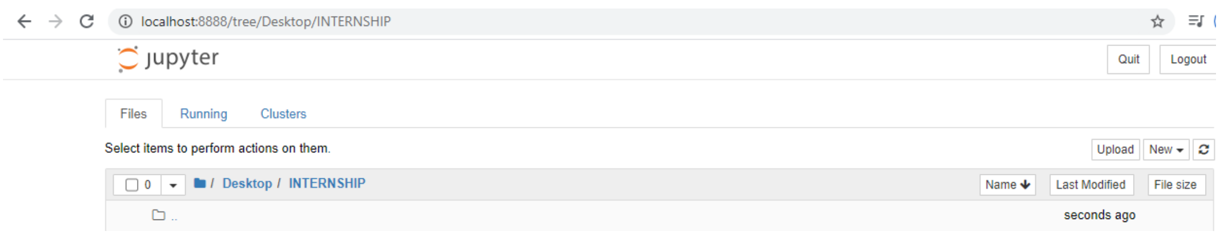


Figure 3.4.2 2 : Jupyter notebook

3.5 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 –

o Numbers

o Strings

o Lists

o Tuples

o Dictionary

3.5.1 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).

3.5.2 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.

3.5.3 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 types.

● 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.

3.5.4 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.

3.5.5 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.

3.6 PYTHON FUNCTION:

3.6.1 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 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.

3.6.2 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.

3.7 PYTHON USING OOPs CONCEPTS:

3.7.1 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.

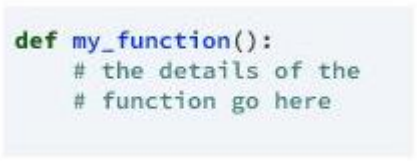
● Data member: A class variable or instance variable that holds data associated with a class and its objects.

● Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

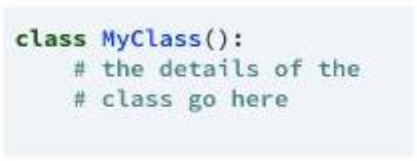
● Defining a Class:

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 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 3.7.1.1: Defining a Class

3.7.2 __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: __init().

CHAPTER 4

CASE STUDY

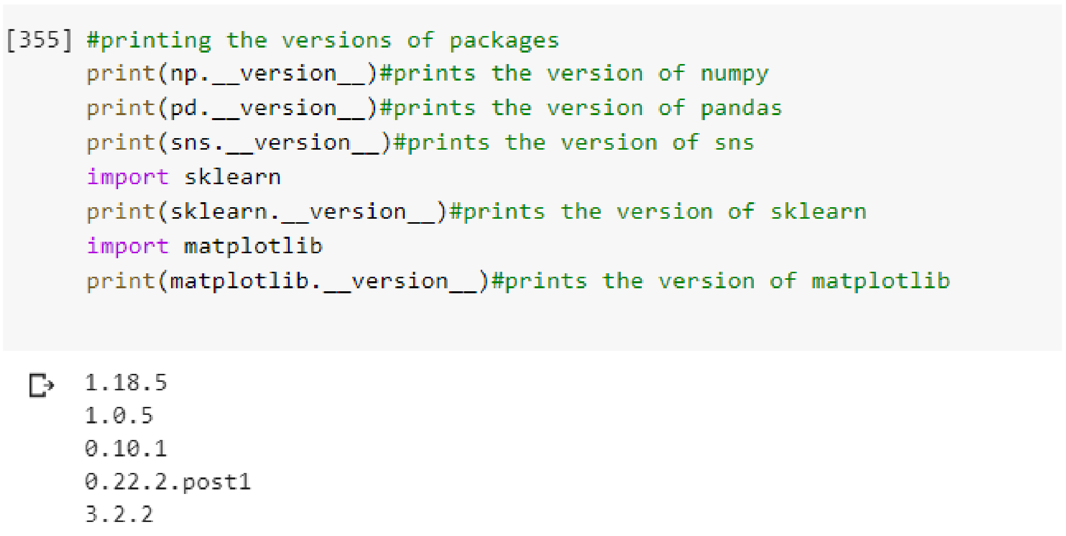
PROJECT NAME-WEATHER FORECASTING

4.1 Project Requirements

Packages used:

1. Pandas
2. Numpy
3. Matplotlib.pyplot
4. Seaborn
5. Matplotlib inline
6. Sklearn
7. From Sklearn.model_selection import train_test_split
8. From Sklearn.metrics import accuracy score
9. From Sklearn.linear_model import Linear Regression
10. From Sklearn.preprocessing import StandardScaler
11. From keras. Models import Sequential
12. From keras. Layers import Dense, Dropout, LSTM

VERSIONS OF PACKAGES:



```
[355] #printing the versions of packages
      print(np.__version__)#prints the version of numpy
      print(pd.__version__)#prints the version of pandas
      print(sns.__version__)#prints the version of sns
      import sklearn
      print(sklearn.__version__)#prints the version of sklearn
      import matplotlib
      print(matplotlib.__version__)#prints the version of matplotlib
```

```
1.18.5
1.0.5
0.10.1
0.22.2.post1
3.2.2
```

Fig 4.1.1: versions of packages which are imported.

Algorithms used:

1.LOGISTIC REGRESSION ALGORITHM

2.RECURRENT NEURAL NETWORK(RNN)

4.2 PROBLEM STATEMENT:

To predict the weather report based on the dataset.

4.3 DATASET DESCRIPTION:

1. Date Time-Date and time of the data record (end)
2. P(mbar)-air pressure
3. T(degC)-air temperature
4. Tpot(K)-potential temperature
5. Tdew(degC)-dew point temperature
6. Rh (%)-relative humidity
7. VPmax(mbar)-saturation water vapour pressure
8. VPmax(mbar)-actual water vapour pressure
9. VPdef(mbar)-water vapour pressure deficit
10. sh(g/kg)-specific humidity

11. H2OC (mmol/ mol)-water vapour concentration
12. rho(g/m**3)-air density
13. wv(m/s)-wind velocity
14. max.wv(m/s)-maximum wind velocity
15. wd(deg)-wind direction
16. rain(mm)-precipitation
17. raining(W/m^2)-duration of precipitation
18. SWDR(W/m^2)-short wave downward radiation
19. PAR-photo synthetically active radiation
20. Max.PAR-maximum photo synthetically active radiation
21. Tlog(degC)-internal logger temperature
22. CO2(ppm)- CO2 concentration of ambient air

4.4 OBJECTIVE OF THE CASE STUDY:

In an Online business, with multiple competitors in the same business it's really important to re-engage existing customers and keep them from churning. This project is my attempt to make a sample model for a company to predict customer's behaviour and prevent them from abandoning their product.

CHAPTER 5
MODEL BUILDING

5.1 PREPROCESSING OF THE DATA:

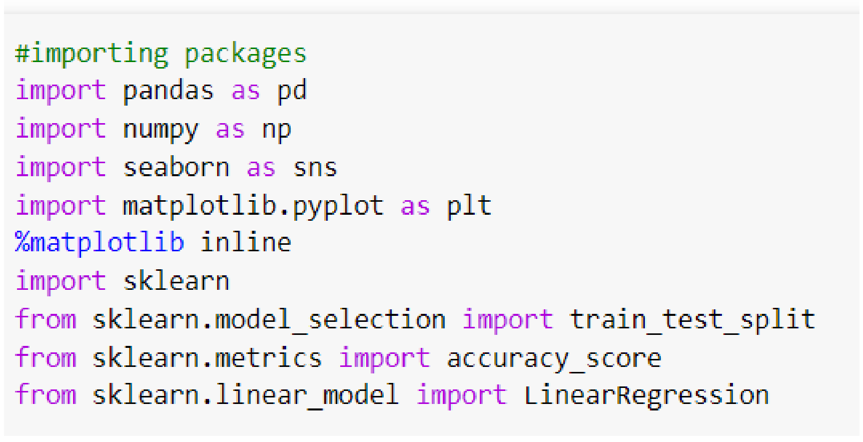
Pre-processing of the data actually involves the following steps:

5.2 GETTING THE DATASET:

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

5.3 IMPORTING THE LIBRARIES:

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

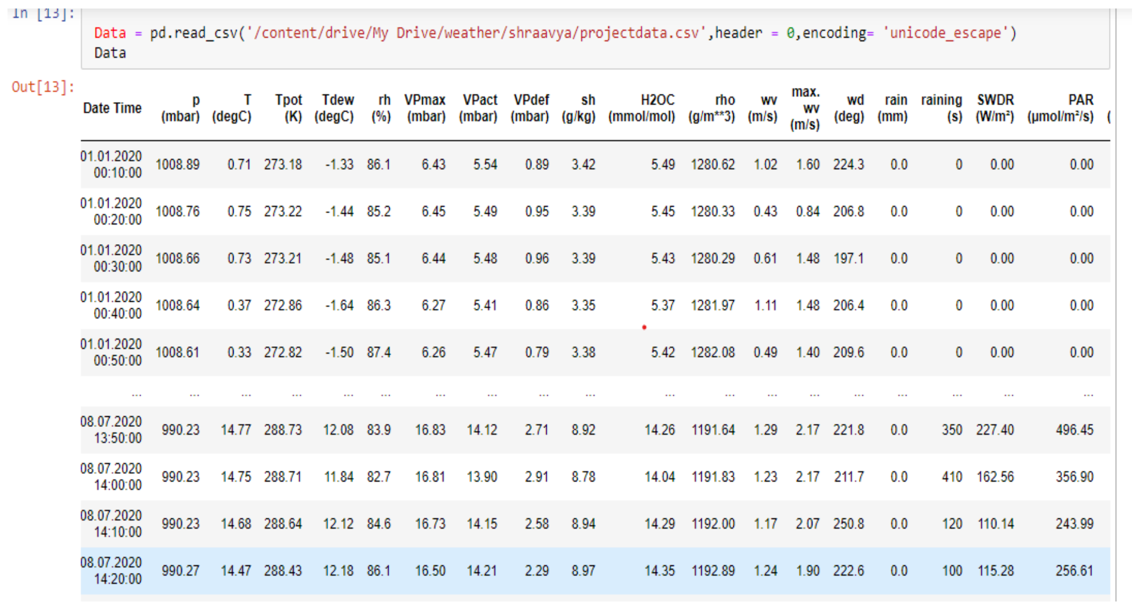


```
#importing packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
```

Fig 5.3.1: importing the libraries.

5.4 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 Data Frame.



```
In [13]:
Data = pd.read_csv('/content/drive/My Drive/weather/shraavya/projectdata.csv',header = 0,encoding= 'unicode_escape')
Data
```

Out[13]:

Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)	rain (mm)	raining (s)	SWDR (W/m²)	PAR (µmol/m²/s)
01.01.2020 00:10:00	1008.89	0.71	273.18	-1.33	86.1	6.43	5.54	0.89	3.42	5.49	1280.62	1.02	1.60	224.3	0.0	0	0.00	0.00
01.01.2020 00:20:00	1008.76	0.75	273.22	-1.44	85.2	6.45	5.49	0.95	3.39	5.45	1280.33	0.43	0.84	206.8	0.0	0	0.00	0.00
01.01.2020 00:30:00	1008.66	0.73	273.21	-1.48	85.1	6.44	5.48	0.96	3.39	5.43	1280.29	0.61	1.48	197.1	0.0	0	0.00	0.00
01.01.2020 00:40:00	1008.64	0.37	272.86	-1.64	86.3	6.27	5.41	0.86	3.35	5.37	1281.97	1.11	1.48	206.4	0.0	0	0.00	0.00
01.01.2020 00:50:00	1008.61	0.33	272.82	-1.50	87.4	6.26	5.47	0.79	3.38	5.42	1282.08	0.49	1.40	209.6	0.0	0	0.00	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
08.07.2020 13:50:00	990.23	14.77	288.73	12.08	83.9	16.83	14.12	2.71	8.92	14.26	1191.64	1.29	2.17	221.8	0.0	350	227.40	496.45
08.07.2020 14:00:00	990.23	14.75	288.71	11.84	82.7	16.81	13.90	2.91	8.78	14.04	1191.83	1.23	2.17	211.7	0.0	410	162.56	356.90
08.07.2020 14:10:00	990.23	14.68	288.64	12.12	84.6	16.73	14.15	2.58	8.94	14.29	1192.00	1.17	2.07	250.8	0.0	120	110.14	243.99
08.07.2020 14:20:00	990.27	14.47	288.43	12.18	86.1	16.50	14.21	2.29	8.97	14.35	1192.89	1.24	1.90	222.6	0.0	100	115.28	256.61

Fig 5.4.1: reading the data.

5.5 HANDLING MISSING VALUES:

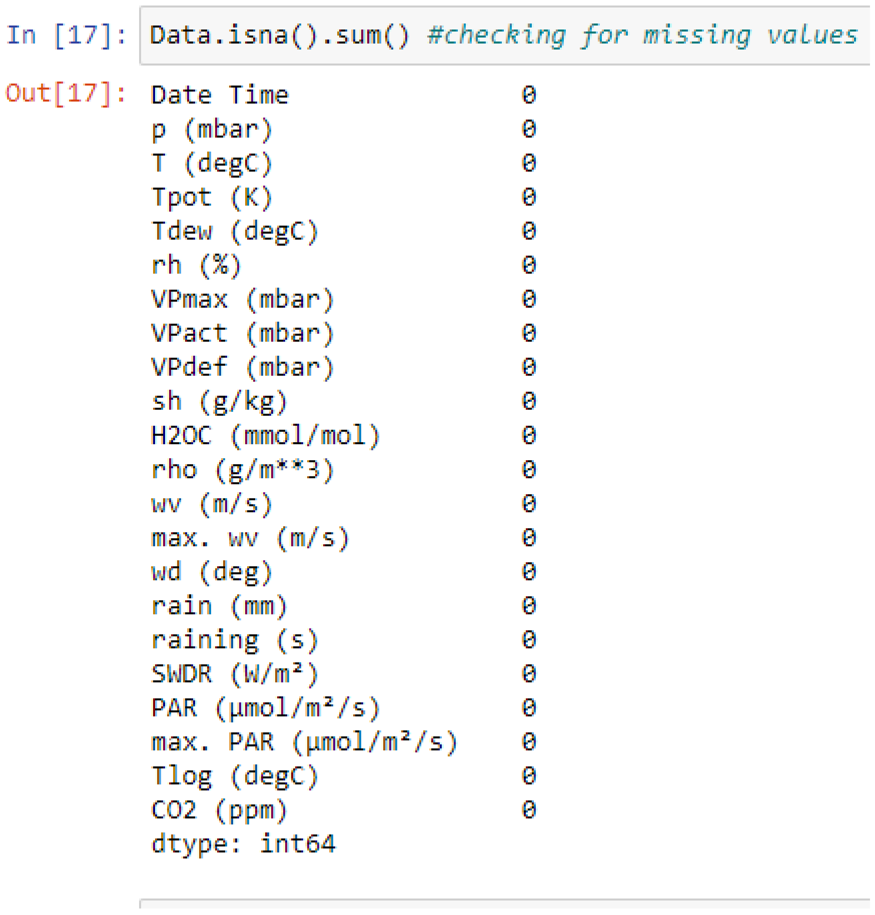
Missing values can be handled in many ways using some inbuilt methods:

(a)dropna ()

(b)fillna ()

(c)interpolate ()

(d)mean imputation and median imputation



```
In [17]: Data.isna().sum() #checking for missing values

Out[17]: Date Time                 0
         p (mbar)                  0
         T (degC)                  0
         Tpot (K)                  0
         Tdew (degC)               0
         rh (%)                    0
         VPmax (mbar)              0
         VPact (mbar)              0
         VPdef (mbar)              0
         sh (g/kg)                 0
         H2OC (mmol/mol)           0
         rho (g/m**3)              0
         wv (m/s)                  0
         max. wv (m/s)             0
         wd (deg)                  0
         rain (mm)                 0
         raining (s)               0
         SWDR (W/m²)               0
         PAR (µmol/m²/s)           0
         max. PAR (µmol/m²/s)      0
         Tlog (degC)               0
         CO2 (ppm)                 0
         dtype: int64
```

Fig 5.5.1: checking the missing values.

There are no missing values in the given data set.

5.6 CATEGORICAL DATA:

- Machine Learning models are based on equations; we need to replace the text by numbers. So that we can include the numbers in the equations.
- Categorical Variables are of two types: Nominal and Ordinal
- Nominal: The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour
- Ordinal: The categories have a numerical ordering in between them.

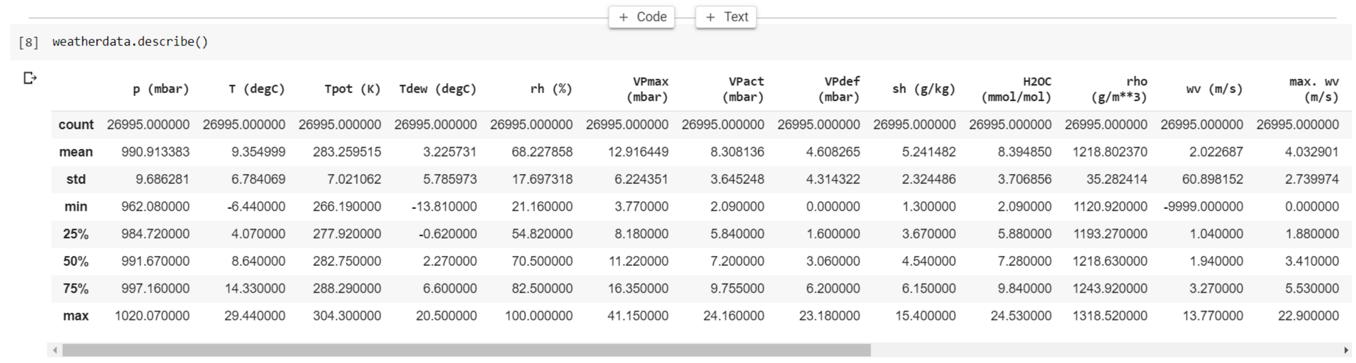
Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

- Categorical data can be handled by using dummy variables, which are also called as indicator variables.
- Handling categorical data using dummies: In panda's library we have a method called get dummies () which creates dummy variables for those categorical data in the form of 0's and 1's. Once these dummies got created we have to concept this dummy set to our data frame or we can add that dummy set to the data frame.

CHAPTER 6
DATA PREPROCESSING/FEATURE ENGINEERING AND EDA

6.1 Statistical data:

Describring the Data:



```
[8] weatherdata.describe()
```

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)
count	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000	26995.000000
mean	990.913383	9.354999	283.259515	3.225731	68.227858	12.916449	8.308136	4.608265	5.241482	8.394850	1218.802370	2.022687	4.032901
std	9.686281	6.784069	7.021062	5.785973	17.697318	6.224351	3.645248	4.314322	2.324486	3.706856	35.282414	60.898152	2.739974
min	962.080000	-6.440000	266.190000	-13.810000	21.160000	3.770000	2.090000	0.000000	1.300000	2.090000	1120.920000	-9999.000000	0.000000
25%	984.720000	4.070000	277.920000	-0.620000	54.820000	8.180000	5.840000	1.600000	3.670000	5.880000	1193.270000	1.040000	1.880000
50%	991.670000	8.640000	282.750000	2.270000	70.500000	11.220000	7.200000	3.060000	4.540000	7.280000	1218.630000	1.940000	3.410000
75%	997.160000	14.330000	288.290000	6.600000	82.500000	16.350000	9.755000	6.200000	6.150000	9.840000	1243.920000	3.270000	5.530000
max	1020.070000	29.440000	304.300000	20.500000	100.000000	41.150000	24.160000	23.180000	15.400000	24.530000	1318.520000	13.770000	22.900000

Fig 6.1.1: describing the data

6.2 Generating Plots:

Visualizations:

```
[9] #visualzing the value with heatmap :
    plt.subplots(figsize=(12,10))
    sns.heatmap(weatherdata.corr(),annot=True)
```

```
[9] <matplotlib.axes._subplots.AxesSubplot at 0x7fd203495b38>
```

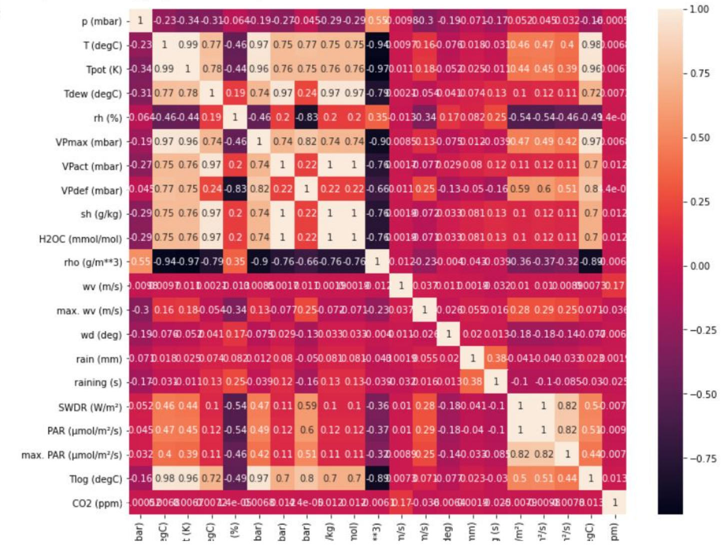


Fig 6.2.1: Visualization of the data using heat map

```
In [22]: Data.hist(figsize=(12,12))
```

```
dtype=object)
```

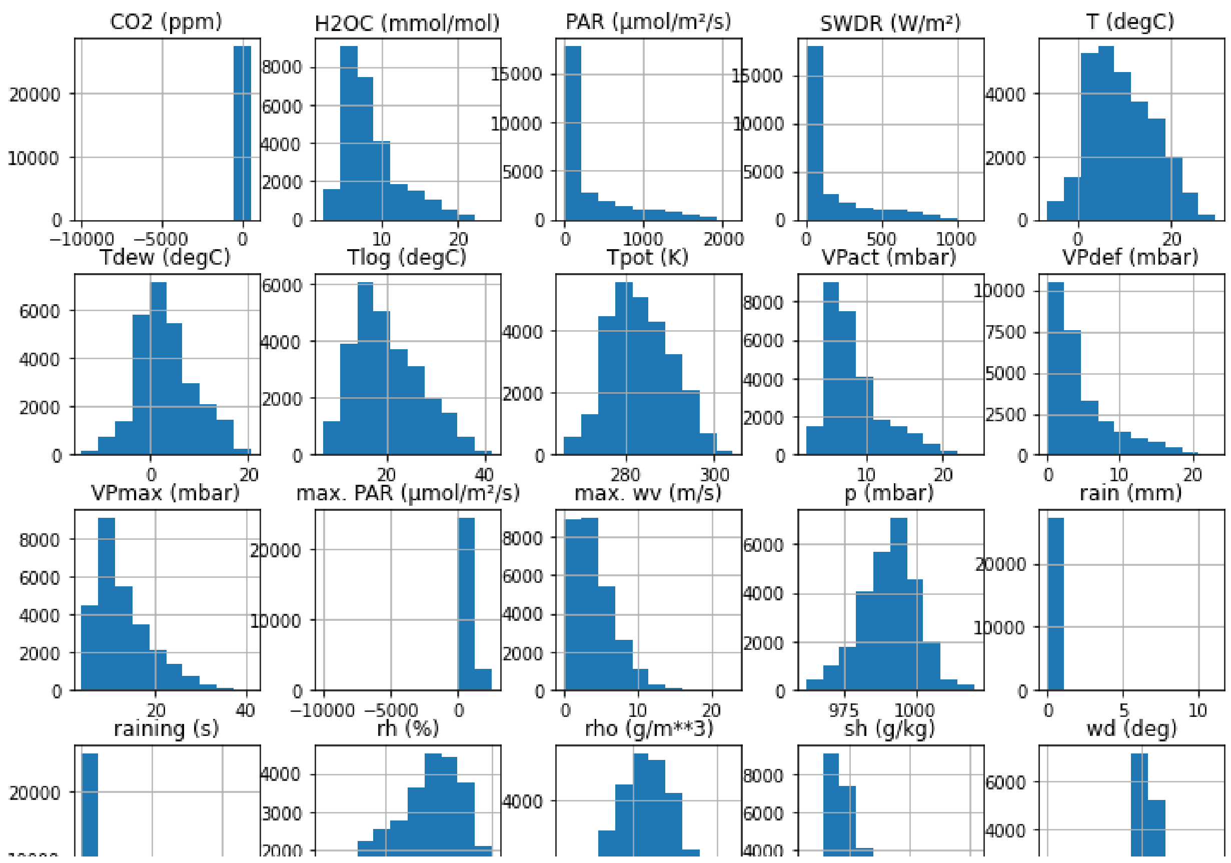
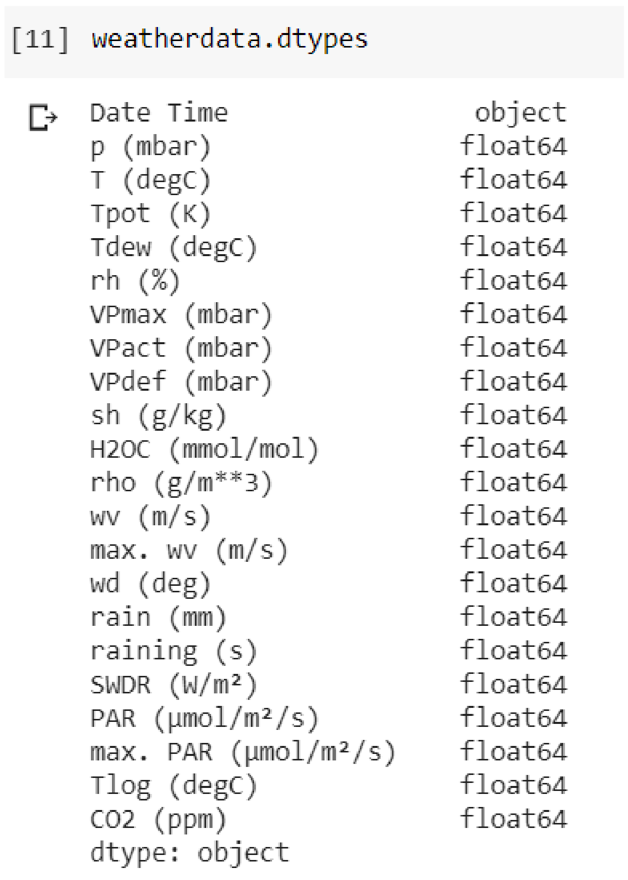


Fig 6.2.2: histogram of all the data in the column

6.3 Data Type Conversions:

In my dataset I have a column namely Date time of type string object and my target column (i.e., potential temperature) of type float and the rest are also of float.



```
[11] weatherdata.dtypes

Date Time                object
p (mbar)                float64
T (degC)                float64
Tpot (K)                float64
Tdew (degC)             float64
rh (%)                  float64
VPmax (mbar)            float64
VPact (mbar)            float64
VPdef (mbar)            float64
sh (g/kg)               float64
H2OC (mmol/mol)         float64
rho (g/m**3)            float64
wv (m/s)                float64
max. wv (m/s)           float64
wd (deg)                float64
rain (mm)               float64
raining (s)             float64
SWDR (W/m²)             float64
PAR (μmol/m²/s)         float64
max. PAR (μmol/m²/s)    float64
Tlog (degC)             float64
CO2 (ppm)               float64
dtype: object
```

Fig 6.3.1 checking the datatype of the columns

6.4 Detection of Outliers:

We don’t have any outliers in the given data.

6.5 Handling Missing Values:

We don’t have any missing values in the given data.

6.6 Encoding Categorical Data:

There are no categorical values in the given dataset. So, encoding categorical values can be skipped.

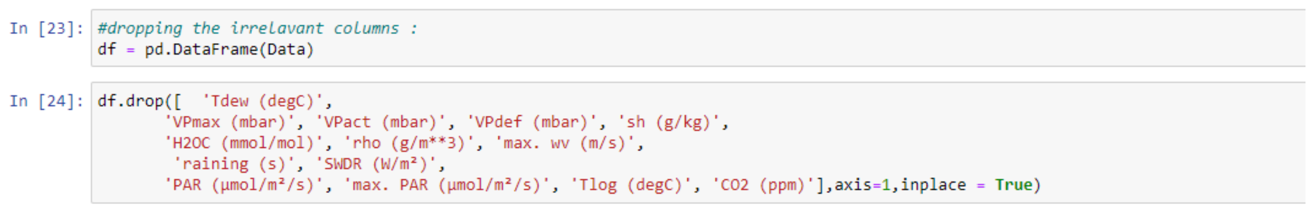
CHAPTER 7
FEATURE SELECTION

7.1 Select relevant features for the analysis:

The irrelevant features in my dataset are area code and state.

These features are considered does not affect my target column. Rest of the columns are all relevant to the target column.

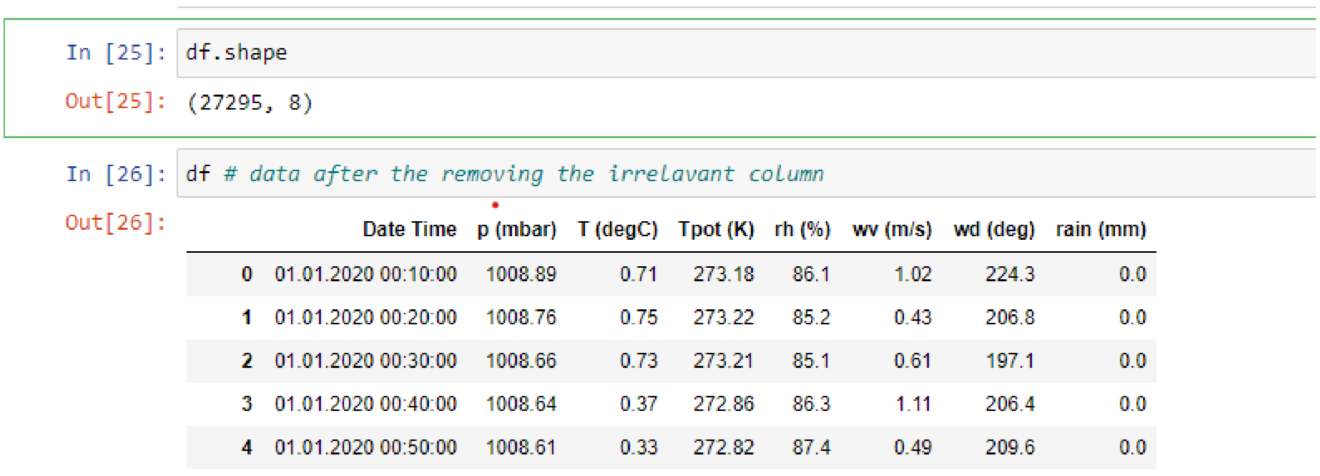
7.2 Drop irrelevant features:



```
In [23]: #dropping the irrelavant columns :
         df = pd.DataFrame(Data)

In [24]: df.drop([  'Tdew (degC)',
                'VPmax (mbar)', 'VPact (mbar)', 'VPdef (mbar)', 'sh (g/kg)',
                'H2OC (mmol/mol)', 'rho (g/m**3)', 'max. wv (m/s)',
                 'raining (s)', 'SWDR (W/m²)',
                'PAR (µmol/m²/s)', 'max. PAR (µmol/m²/s)', 'Tlog (degC)', 'CO2 (ppm)'],axis=1,inplace = True)
```

7.2.1: Dropping the irrelevant columns

In the above we have drop the columns which I felt that is nothing to do to find the weather forecasting of the given data.



```
In [25]: df.shape
Out[25]: (27295, 8)

In [26]: df # data after the removing the irrelavant column
Out[26]:
```

	Date Time	p (mbar)	T (degC)	Tpot (K)	rh (%)	wv (m/s)	wd (deg)	rain (mm)
0	01.01.2020 00:10:00	1008.89	0.71	273.18	86.1	1.02	224.3	0.0
1	01.01.2020 00:20:00	1008.76	0.75	273.22	85.2	0.43	206.8	0.0
2	01.01.2020 00:30:00	1008.66	0.73	273.21	85.1	0.61	197.1	0.0
3	01.01.2020 00:40:00	1008.64	0.37	272.86	86.3	1.11	206.4	0.0
4	01.01.2020 00:50:00	1008.61	0.33	272.82	87.4	0.49	209.6	0.0

Fig 7.2.2: The data after dropping the columns.

7.3 Train-Test-Split:

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.

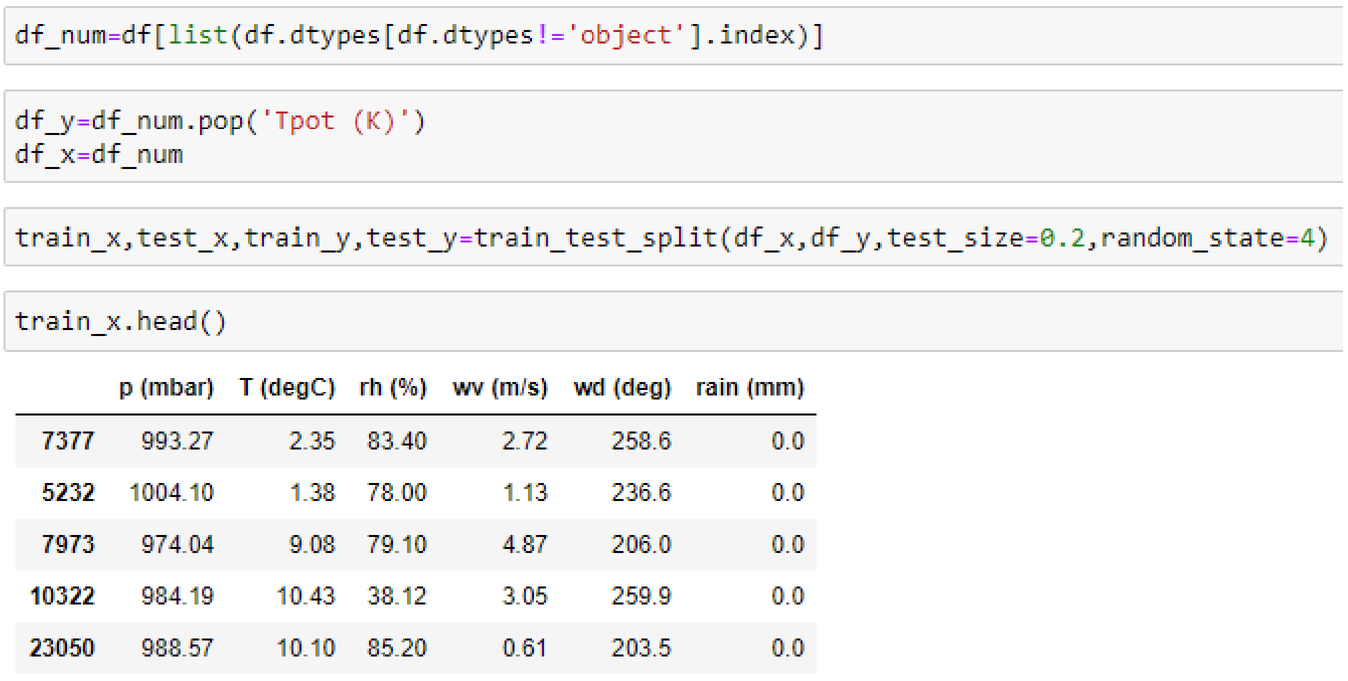
● training set - a subset to train a model. (Model learns patterns between Input and Output

● test set - a subset to test the trained model (To test whether the mode 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

● Here we have taken the test size as 0.2 indicating train data =80% and test-data=20%



```
df_num=df[list(df.dtypes[df.dtypes!='object'].index)]
```

```
df_y=df_num.pop('Tpot (K)')
df_x=df_num
```

```
train_x,test_x,train_y,test_y=train_test_split(df_x,df_y,test_size=0.2,random_state=4)
```

```
train_x.head()
```

	p (mbar)	T (degC)	rh (%)	wv (m/s)	wd (deg)	rain (mm)
7377	993.27	2.35	83.40	2.72	258.6	0.0
5232	1004.10	1.38	78.00	1.13	236.6	0.0
7973	974.04	9.08	79.10	4.87	206.0	0.0
10322	984.19	10.43	38.12	3.05	259.9	0.0
23050	988.57	10.10	85.20	0.61	203.5	0.0

Fig 7.3.1: importing train_test_split

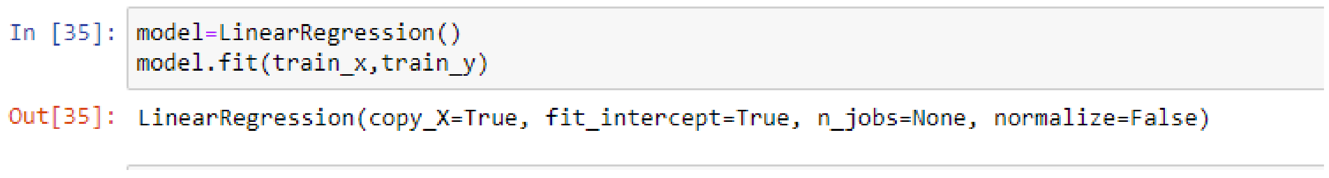
CHAPTER 8
MODEL BUILDING AND EVALUATION

8.1 Linear Regression:

8.1.1 Brief about the algorithms used.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this **regression** technique finds out a **linear** relationship between x (input) and y(output). Hence, the name is **Linear Regression**.

8.1.2 Train the Models

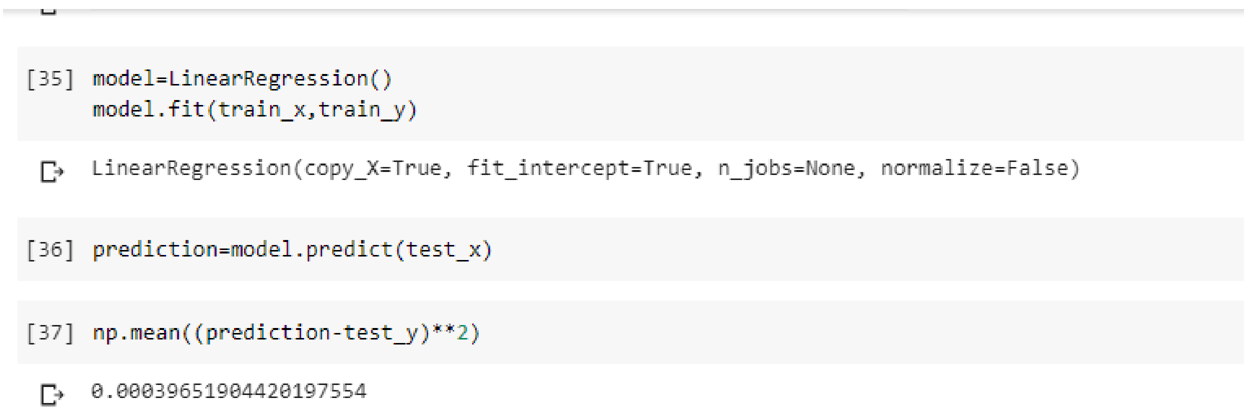


```
In [35]: model=LinearRegression()
         model.fit(train_x,train_y)

Out[35]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Fig 8.1.2.1: training the model.

8.1.3 Make Predictions:

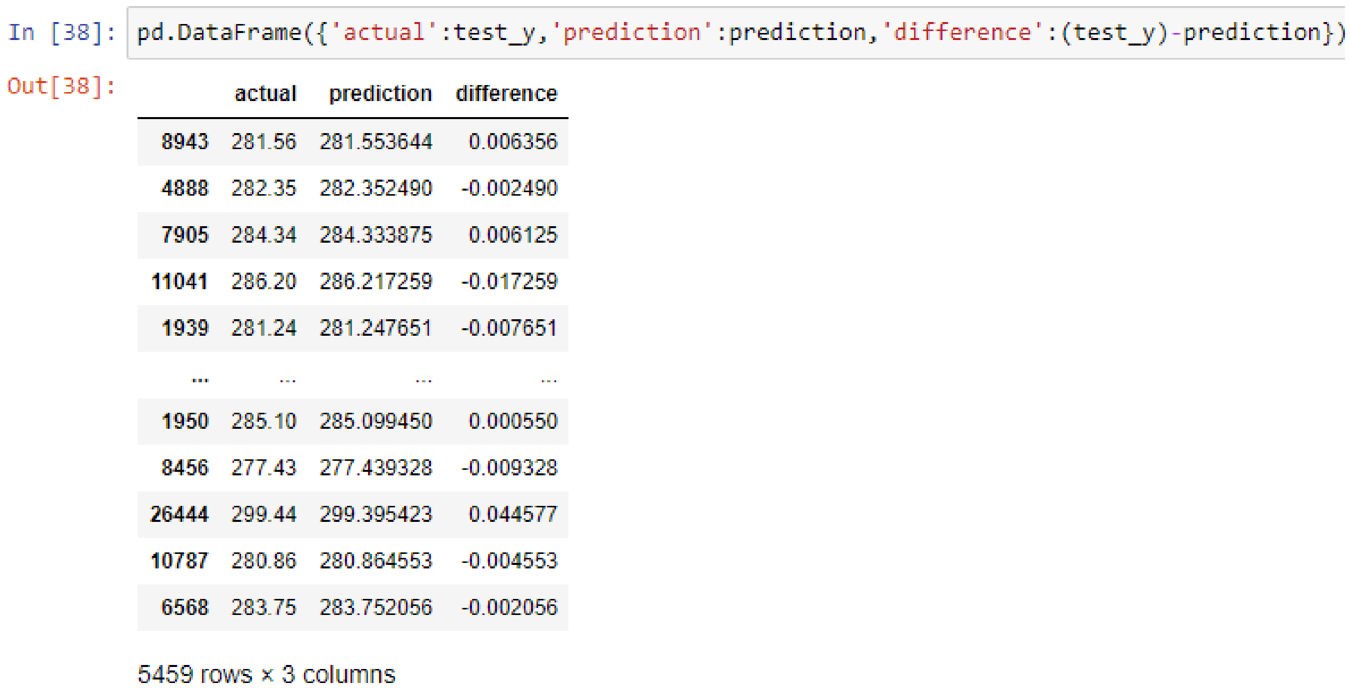
I have sent test_x (i.e., my testing dataset) to the predict () as argument and I have stored the results in prediction.



```
[35] model=LinearRegression()
     model.fit(train_x,train_y)

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

[36] prediction=model.predict(test_x)

[37] np.mean((prediction-test_y)**2)

     0.00039651904420197554
```

Fig 8.1.3.1: predicting the raw data.

8.1.4 Making the Prediction Table:



```
In [38]: pd.DataFrame({'actual':test_y,'prediction':prediction,'difference':(test_y)-prediction})
Out[38]:
```

	actual	prediction	difference
8943	281.56	281.553644	0.006356
4888	282.35	282.352490	-0.002490
7905	284.34	284.333875	0.006125
11041	286.20	286.217259	-0.017259
1939	281.24	281.247651	-0.007651
...	...	...	...
1950	285.10	285.099450	0.000550
8456	277.43	277.439328	-0.009328
26444	299.44	299.395423	0.044577
10787	280.86	280.864553	-0.004553
6568	283.75	283.752056	-0.002056

5459 rows × 3 columns

Fig 8.1.4.1: making the Data frame of prediction table

Making the data frame of Testy, Prediction, Difference of testy and prediction.

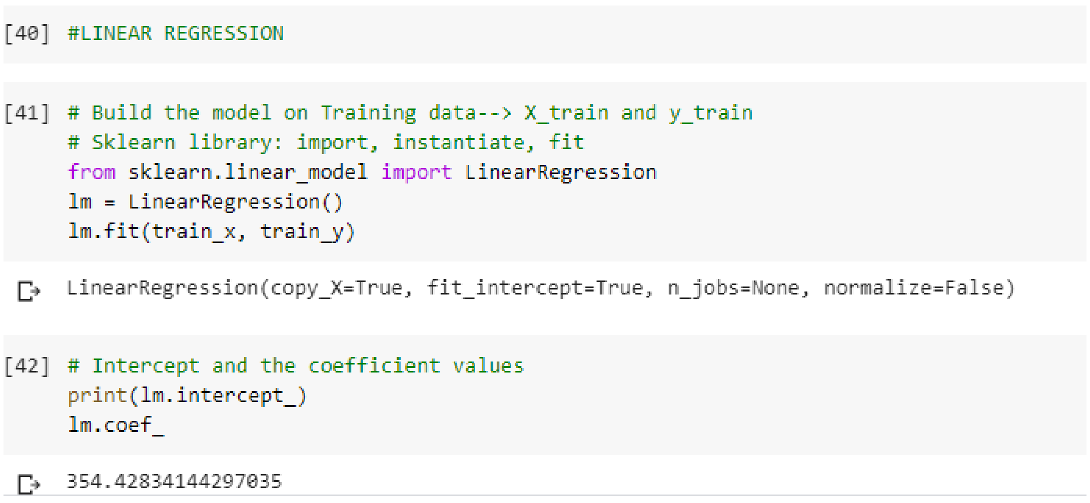
8.2 Linear regression steps:

8.2.1: Training the Data: First we import the ‘Linear regression’ method from the ‘Sklearn.linear_model’ package.

Then we create an object called Linear Regression () and store it in a variable called ‘lm’

Next we apply the algorithm to the data and fit the data using the syntax:

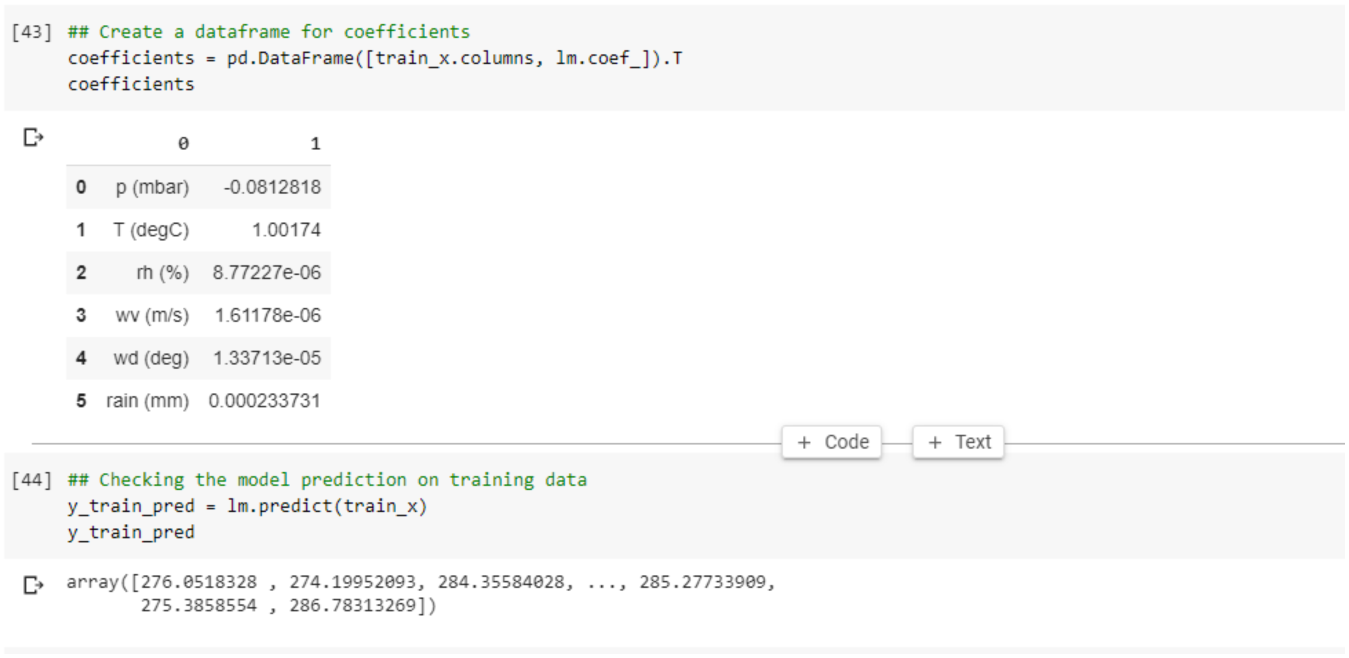
objectname.fit (input, output).



```
[40] #LINEAR REGRESSION

[41] # Build the model on Training data--> X_train and y_train
     # Sklearn library: import, instantiate, fit
     from sklearn.linear_model import LinearRegression
     lm = LinearRegression()
     lm.fit(train_x, train_y)

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

[42] # Intercept and the coefficient values
     print(lm.intercept_)
     lm.coef_

     354.42834144297035
```

Fig 8.2.1.1: importing the package and storing in ‘lm’ variable.

8.2.2: Creating the data frame for coefficients and checking the model prediction on training data of y train:



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

	0	1
0	p (mbar)	-0.0812818
1	T (degC)	1.00174
2	rh (%)	8.77227e-06
3	wv (m/s)	1.61178e-06
4	wd (deg)	1.33713e-05
5	rain (mm)	0.000233731

```
[44] ## Checking the model prediction on training data
     y_train_pred = lm.predict(train_x)
     y_train_pred

     array([276.0518328 , 274.19952093, 284.35584028, ..., 285.27733909,
            275.3858554 , 286.78313269])
```

Fig 8.2.2.1: creating coef and checking the model prediction.

8.2.3: we are comparing the y_train and the predicted value of y_train_pred:

```
## 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(train_y,y_train_pred))
print('Adjusted R^2:', 1- (1-r2_score(train_y, y_train_pred))*(len(train_x)-1)/(len(train_x)-train_x.shape[1]-1))
print('MAE:', mean_absolute_error(train_y, y_train_pred))
print('MSE:', mean_squared_error(train_y, y_train_pred))
print('RMSE', np.sqrt(mean_squared_error(train_y, y_train_pred)))
```

```
R^2: 0.999991913536648
Adjusted R^2: 0.9999919113139726
MAE: 0.014607275079599939
MSE: 0.00039655115185577763
RMSE 0.019913592138430886
```

Fig 8.2.3.1: predictions of y_train and y_train_pred.

Here we are comparing the actual values of y_train and the predicted values of y_train_pred.

Whereas:

MAE is mean absolute error

MSE is mean square error

We got the R^2 value as 0.99 approx.

8.2.4: plot indicating the actual values of y_train and y_train_pred:

```
## regplot(y_train--> actual values and y_train_pred--> predicted values)
sns.regplot(train_y, y_train_pred)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc9501a3438>
```

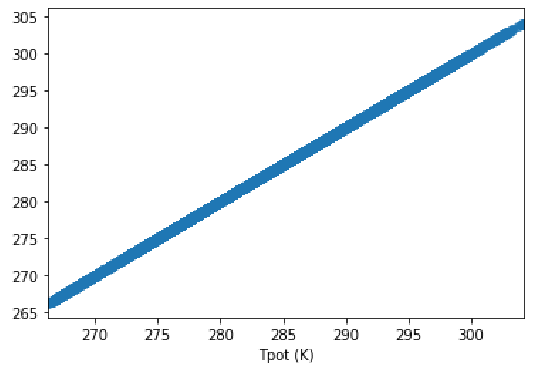
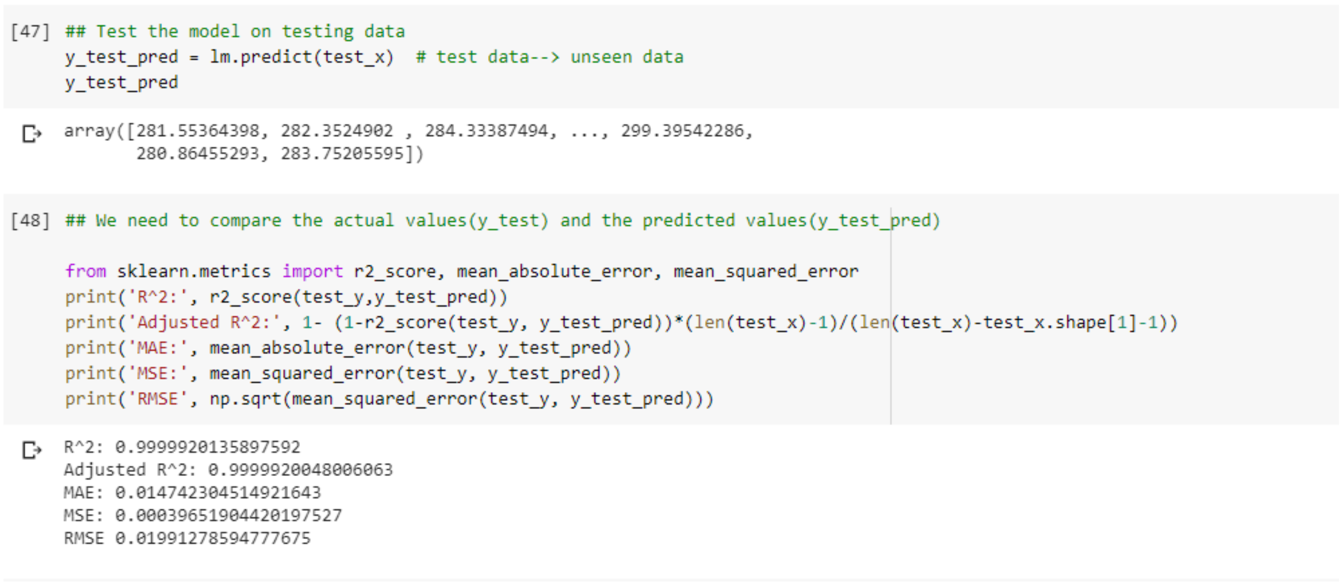


Fig 8.2.4.1 graph indicating the train_y and train_y_pred

The regression plot of the tpot(k) of train_y and y_train_pred has the straight line indicating towards upwards. Where x axis indicates the y_train and y axis indicates the train_y_pred of Tplot(k).

8.2.5: Test the model on testing data:



```
[47] ## Test the model on testing data
     y_test_pred = lm.predict(test_x)  # test data--> unseen data
     y_test_pred

     array([281.55364398, 282.3524902 , 284.33387494, ..., 299.39542286,
            280.86455293, 283.75205595])

[48] ## We need to compare the actual values(y_test) and the predicted values(y_test_pred)

     from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
     print('R^2:', r2_score(test_y,y_test_pred))
     print('Adjusted R^2:', 1- (1-r2_score(test_y, y_test_pred))*(len(test_x)-1)/(len(test_x)-test_x.shape[1]-1))
     print('MAE:', mean_absolute_error(test_y, y_test_pred))
     print('MSE:', mean_squared_error(test_y, y_test_pred))
     print('RMSE', np.sqrt(mean_squared_error(test_y, y_test_pred)))

     R^2: 0.9999920135897592
     Adjusted R^2: 0.9999920048006063
     MAE: 0.014742304514921643
     MSE: 0.00039651904420197527
     RMSE 0.01991278594777675
```

8.2.5.1: The prediction on testing data

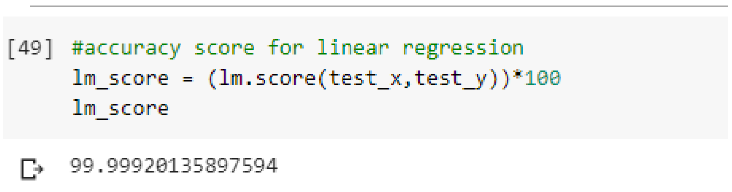
Before we have taken for the train data in the above we have followed the same method for finding the training data. Here we are taking for the testing data.

Here we need to compare the actual value of y_test and the predicted values of y_test_pred.

We have got the R^2 as 0.99 approx.

Here the training and testing data of R^2 values are almost same so that's the reason the mat plot graph is straight.

8.2.6: Accuracy score for linear regression:



```
[49] #accuracy score for linear regression
     lm_score = (lm.score(test_x,test_y))*100
     lm_score

     99.99920135897594
```

Fig 8.2.6.1: accuracy score.

The accuracy score for linear regression for test_x, test_y for 100 % we have got the accuracy score as 99.99.

8.3 RNN (RECURRENT NEURAL NETWORK):

8.3.1 Brief about the algorithms used

Recurrent neural networks (RNN) are a class of neural networks that is powerful for modelling sequence data such as time series or natural language.

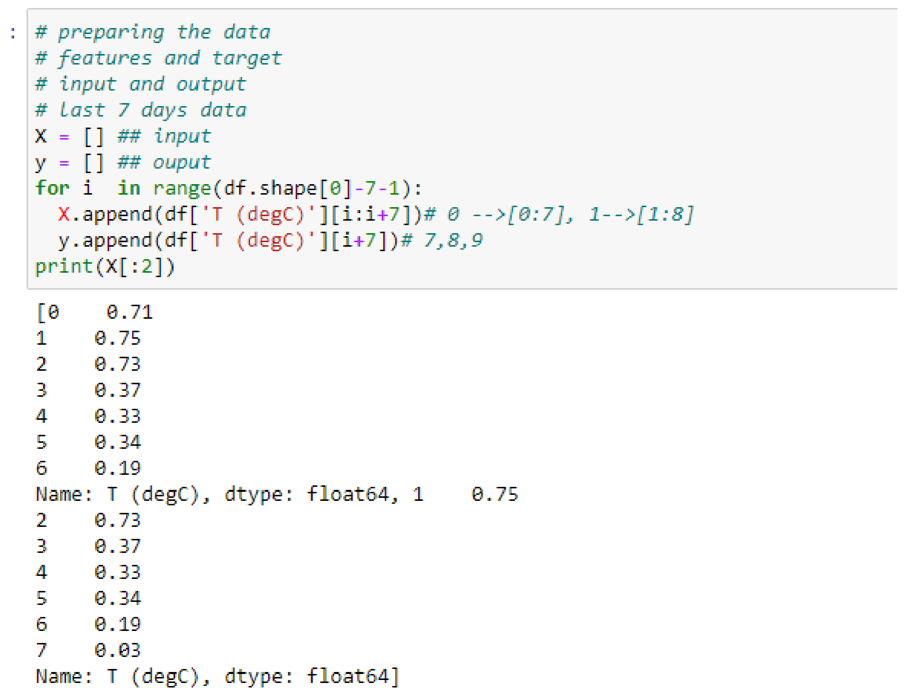
Schematically, a RNN layer uses a `for` loop to iterate over the time steps of a sequence, while maintaining an internal state that encodes information about the time steps it has seen so far.

The Keras RNN API is designed with a focus on:

- **Ease of use**: the built-in `keras.layers.RNN`, `keras.layers.LSTM`, `keras.layers.GRU` layers enable you to quickly build recurrent models without having to make difficult configuration choices.
- **Ease of customization**: You can also define your own RNN cell layer (the inner part of the `for` loop) with custom behaviour, and use it with the generic `keras.layers.RNN` layer (the `for` loop itself). This allows you to quickly prototype different research ideas in a flexible way with minimal code.

8.3.2 Train the Model:

In the below I have taken x as the input and y as the output. “i” is the range of shape. X is given 7 columns and Y is given as 8.



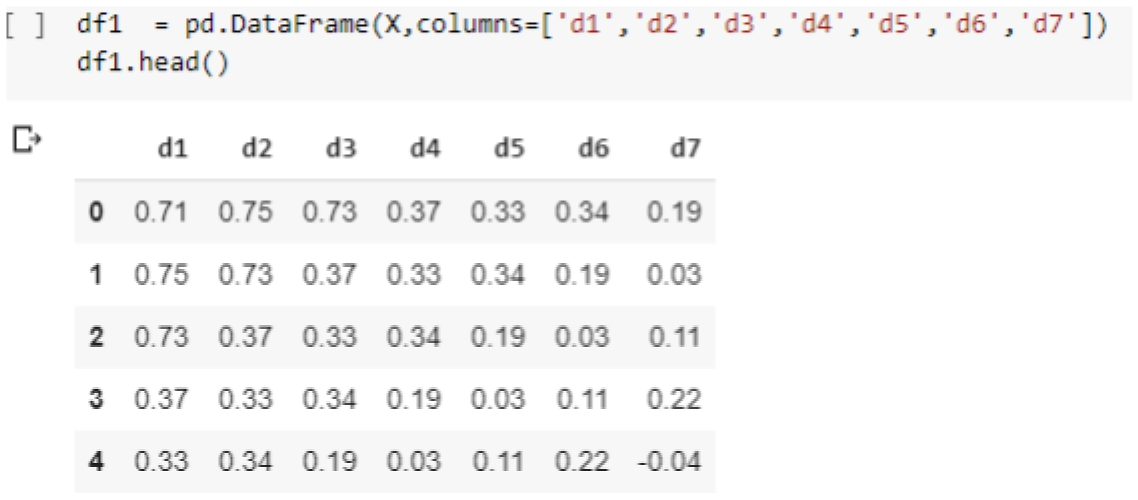
```
: # preparing the data
  # features and target
  # input and output
  # last 7 days data
  X = [] ## input
  y = [] ## ouput
  for i  in range(df.shape[0]-7-1):
    X.append(df['T (degC)'][i:i+7])# 0 -->[0:7], 1-->[1:8]
    y.append(df['T (degC)'][i+7])# 7,8,9
  print(X[:2])
```

```
[0    0.71
1    0.75
2    0.73
3    0.37
4    0.33
5    0.34
6    0.19
Name: T (degC), dtype: float64, 1    0.75
2    0.73
3    0.37
4    0.33
5    0.34
6    0.19
7    0.03
Name: T (degC), dtype: float64]
```

Fig 8.3.2.1:preparing the data.

8.3.3: Assiging the values (x):

Here we have taken 7 columns and the top 5 rows by using the head command.

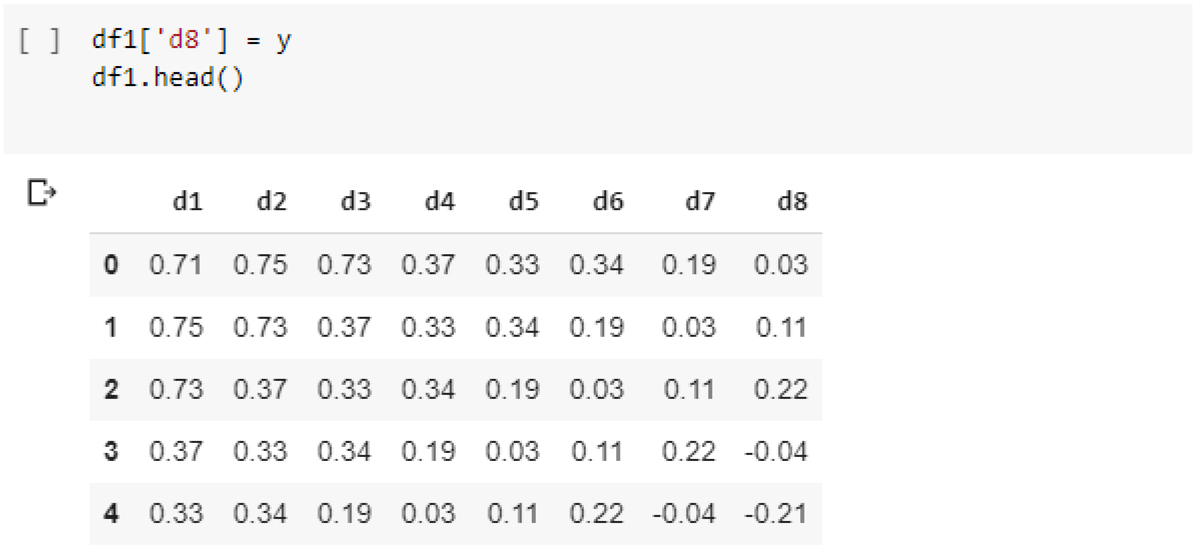


```
[ ]  df1  = pd.DataFrame(X,columns=['d1','d2','d3','d4','d5','d6','d7'])
     df1.head()
```

	d1	d2	d3	d4	d5	d6	d7
0	0.71	0.75	0.73	0.37	0.33	0.34	0.19
1	0.75	0.73	0.37	0.33	0.34	0.19	0.03
2	0.73	0.37	0.33	0.34	0.19	0.03	0.11
3	0.37	0.33	0.34	0.19	0.03	0.11	0.22
4	0.33	0.34	0.19	0.03	0.11	0.22	-0.04

Fig 8.3.3.1: describing the data.

8.3.4: Assiging the values (y):



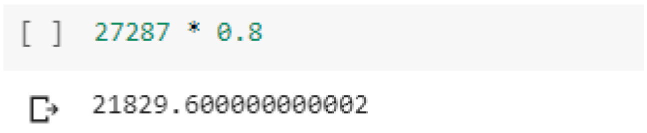
```
[ ]  df1['d8'] = y
     df1.head()
```

	d1	d2	d3	d4	d5	d6	d7	d8
0	0.71	0.75	0.73	0.37	0.33	0.34	0.19	0.03
1	0.75	0.73	0.37	0.33	0.34	0.19	0.03	0.11
2	0.73	0.37	0.33	0.34	0.19	0.03	0.11	0.22
3	0.37	0.33	0.34	0.19	0.03	0.11	0.22	-0.04
4	0.33	0.34	0.19	0.03	0.11	0.22	-0.04	-0.21

8.3.4.1: data of y

Here, y value is given as 8 columns, where x is given 7.

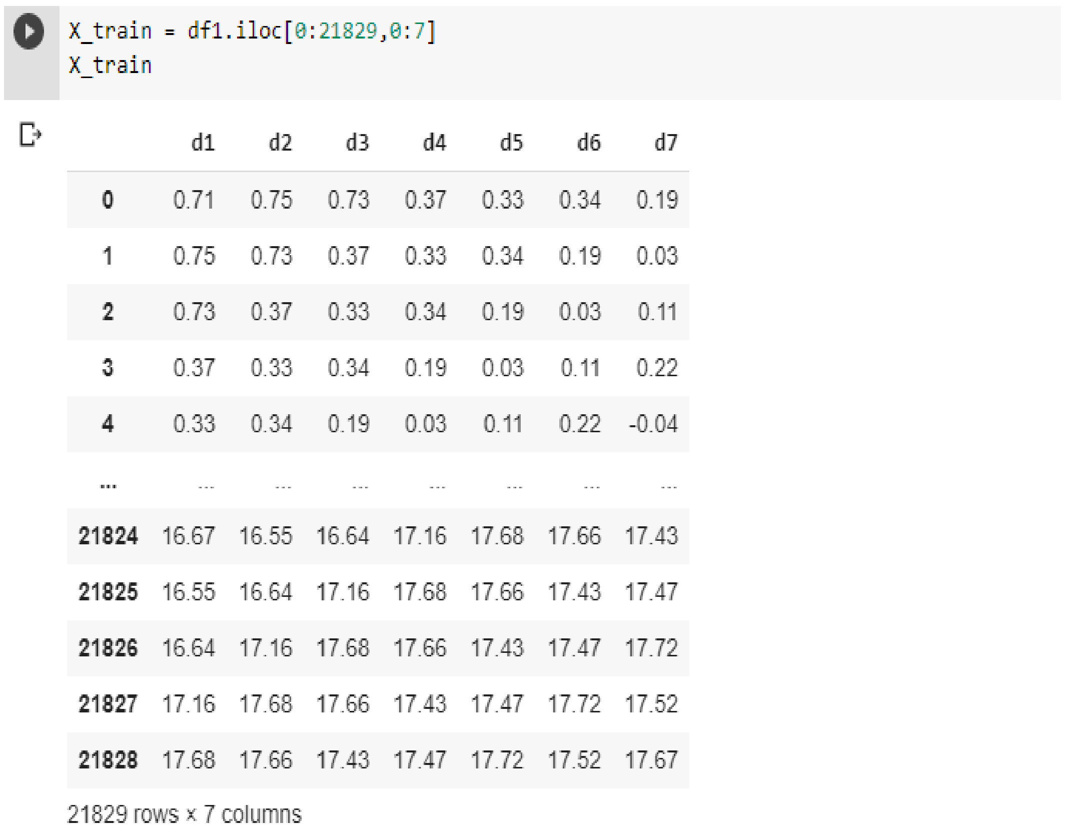
8.3.5: Train_Test_Split:



```
[ ]  27287 * 0.8

     21829.600000000002
```

Fig 8.3.5.1: training the data.

Here we are taking 0.8 because it has 80% for train and 20 % test.



```
X_train = df1.iloc[0:21829,0:7]
X_train
```

	d1	d2	d3	d4	d5	d6	d7
0	0.71	0.75	0.73	0.37	0.33	0.34	0.19
1	0.75	0.73	0.37	0.33	0.34	0.19	0.03
2	0.73	0.37	0.33	0.34	0.19	0.03	0.11
3	0.37	0.33	0.34	0.19	0.03	0.11	0.22
4	0.33	0.34	0.19	0.03	0.11	0.22	-0.04
...	...	...	...	...	...	...	...
21824	16.67	16.55	16.64	17.16	17.68	17.66	17.43
21825	16.55	16.64	17.16	17.68	17.66	17.43	17.47
21826	16.64	17.16	17.68	17.66	17.43	17.47	17.72
21827	17.16	17.68	17.66	17.43	17.47	17.72	17.52
21828	17.68	17.66	17.43	17.47	17.72	17.52	17.67

21829 rows × 7 columns

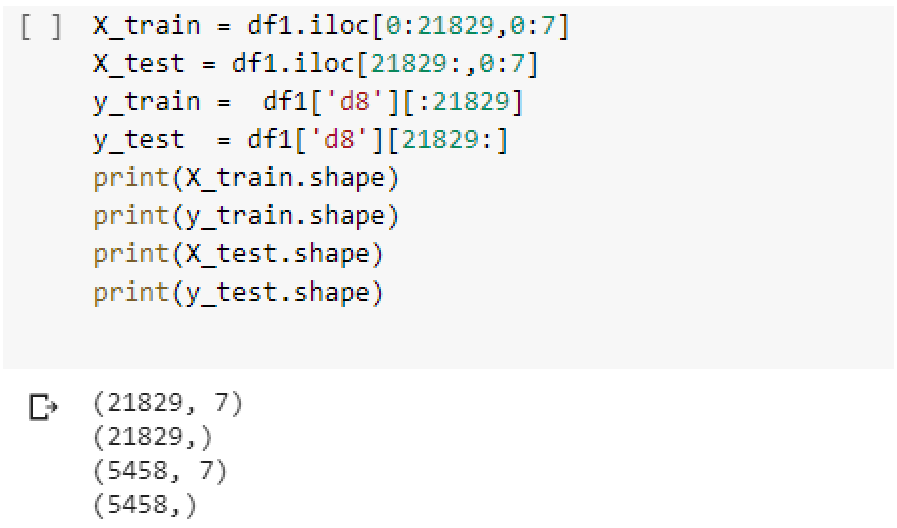
Fig 8.3.5.2 Describing the df1 data for x_train.

8.3.6: finding the shape of x and y:

In the above we have taken X_train, y_train and X_test and y_test and print those columns values and shape.

I have sent X_train (i.e., my training dataset) to the predict () as argument and I have stored the results.

I have sent X_test (i.e., my testing dataset) to the predict () as argument and I have stored the results.



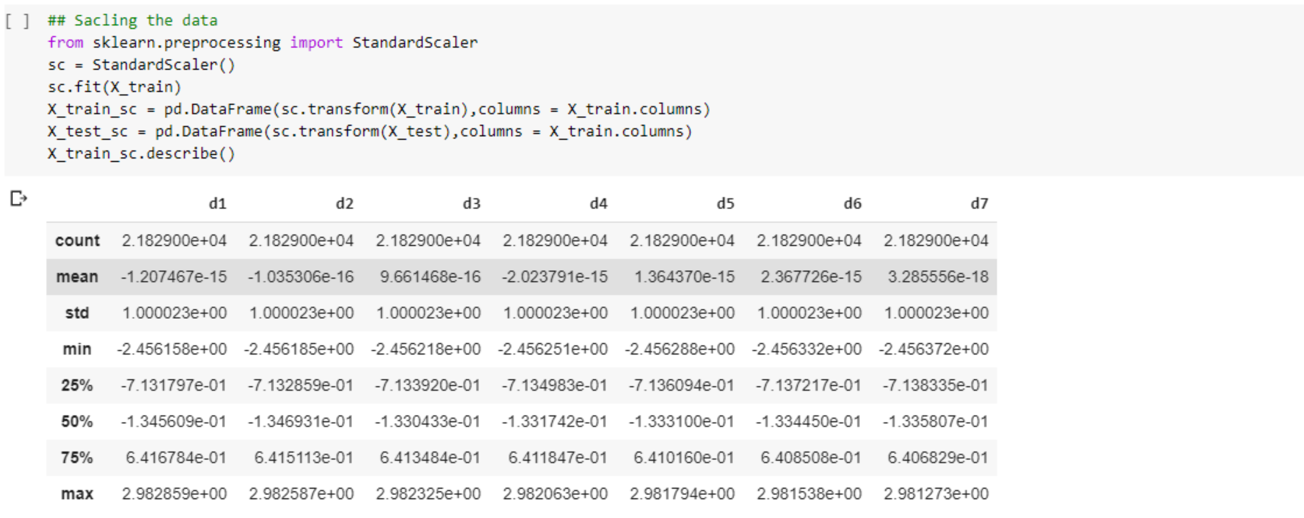
```
[ ] X_train = df1.iloc[0:21829,0:7]
    X_test = df1.iloc[21829:,0:7]
    y_train =  df1['d8'][:21829]
    y_test  = df1['d8'][21829:]
    print(X_train.shape)
    print(y_train.shape)
    print(X_test.shape)
    print(y_test.shape)
```

```
(21829, 7)
(21829,)
(5458, 7)
(5458,)
```

Fig 8.3.6.1: printing the shape.

8.3.7: Scaling the data:

Scaling the values in the columns by importing the standard scalar and made sc_fit(X_train) and making the necessary changes in the x train and x_test columns and at last describe the data.



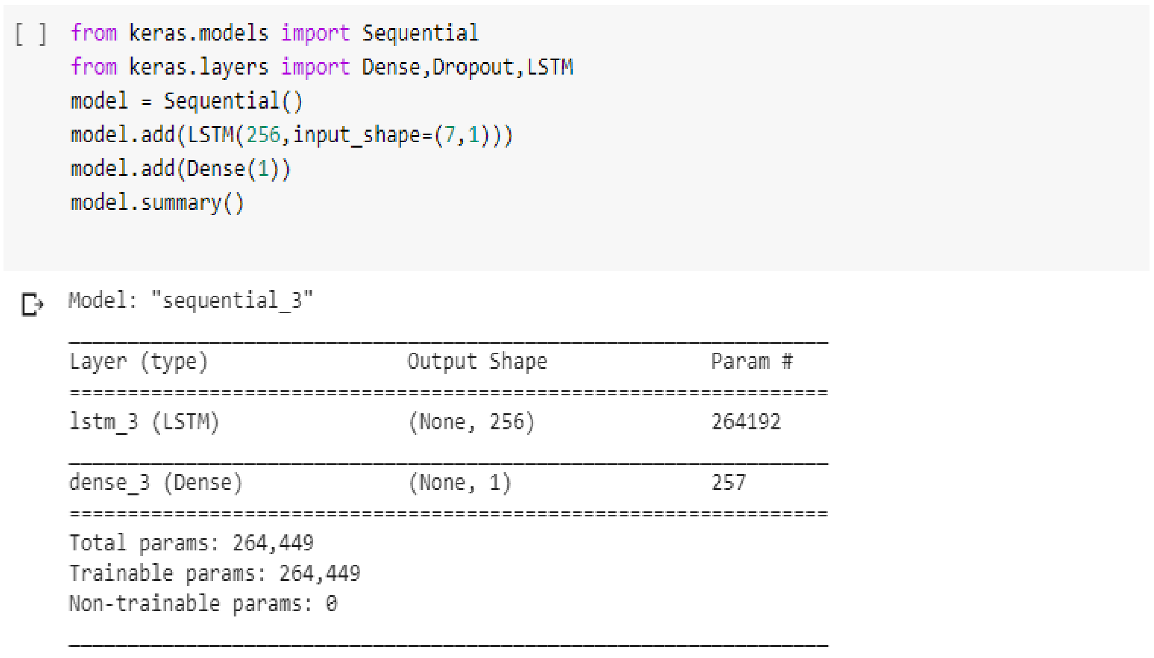
```
[ ] ## Sacling the data
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    sc.fit(X_train)
    X_train_sc = pd.DataFrame(sc.transform(X_train),columns = X_train.columns)
    X_test_sc = pd.DataFrame(sc.transform(X_test),columns = X_train.columns)
    X_train_sc.describe()
```

	d1	d2	d3	d4	d5	d6	d7
count	2.182900e+04	2.182900e+04	2.182900e+04	2.182900e+04	2.182900e+04	2.182900e+04	2.182900e+04
mean	-1.207467e-15	-1.035306e-16	9.661468e-16	-2.023791e-15	1.364370e-15	2.367726e-15	3.285556e-18
std	1.000023e+00	1.000023e+00	1.000023e+00	1.000023e+00	1.000023e+00	1.000023e+00	1.000023e+00
min	-2.456158e+00	-2.456185e+00	-2.456218e+00	-2.456251e+00	-2.456288e+00	-2.456332e+00	-2.456372e+00
25%	-7.131797e-01	-7.132859e-01	-7.133920e-01	-7.134983e-01	-7.136094e-01	-7.137217e-01	-7.138335e-01
50%	-1.345609e-01	-1.346931e-01	-1.330433e-01	-1.331742e-01	-1.333100e-01	-1.334450e-01	-1.335807e-01
75%	6.416784e-01	6.415113e-01	6.413484e-01	6.411847e-01	6.410160e-01	6.408508e-01	6.406829e-01
max	2.982859e+00	2.982587e+00	2.982325e+00	2.982063e+00	2.981794e+00	2.981538e+00	2.981273e+00

Fig 8.3.7.1: scaling the data.

8.3.8: importing the keras:

We are importing keras models and importing the LSTM, dropout, dense functions. By taking the shape of as 7,1 and adding the dense at 1.



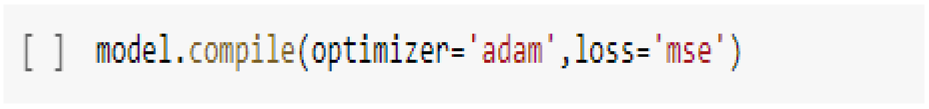
```
[ ] from keras.models import Sequential
    from keras.layers import Dense,Dropout,LSTM
    model = Sequential()
    model.add(LSTM(256,input_shape=(7,1)))
    model.add(Dense(1))
    model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 256)               264192
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 264,449
Trainable params: 264,449
Non-trainable params: 0
_________________________________________________________________
```

Fig 8.3.8.1: importing the keras.

8.3.9: Describing the model data:

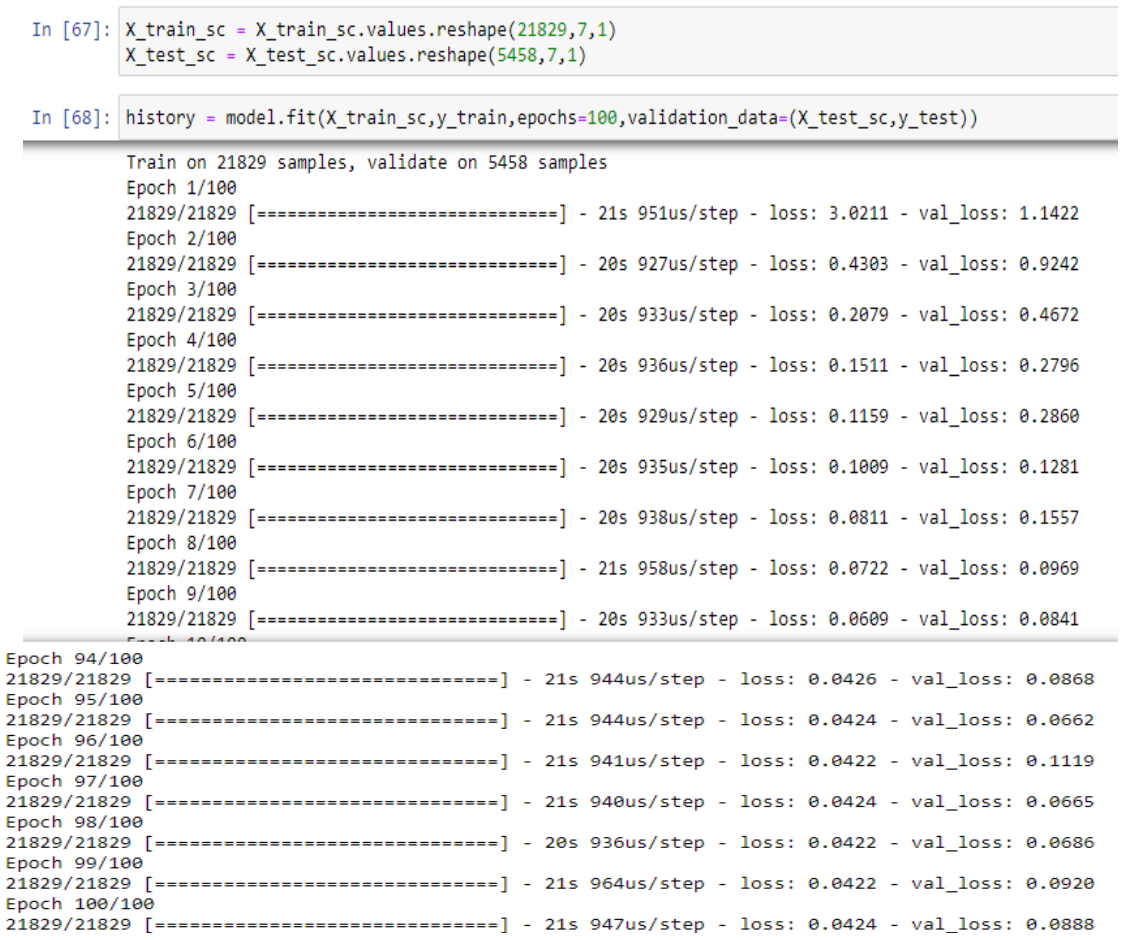
Describing a model the value of optimizer is default, whereas loss is MSE that is mean square error.



```
[ ] model.compile(optimizer='adam',loss='mse')
```

Fig 8.3.9.1 describing the data.

8.3.10: Reshaping and history of x_test_sc and y_test:



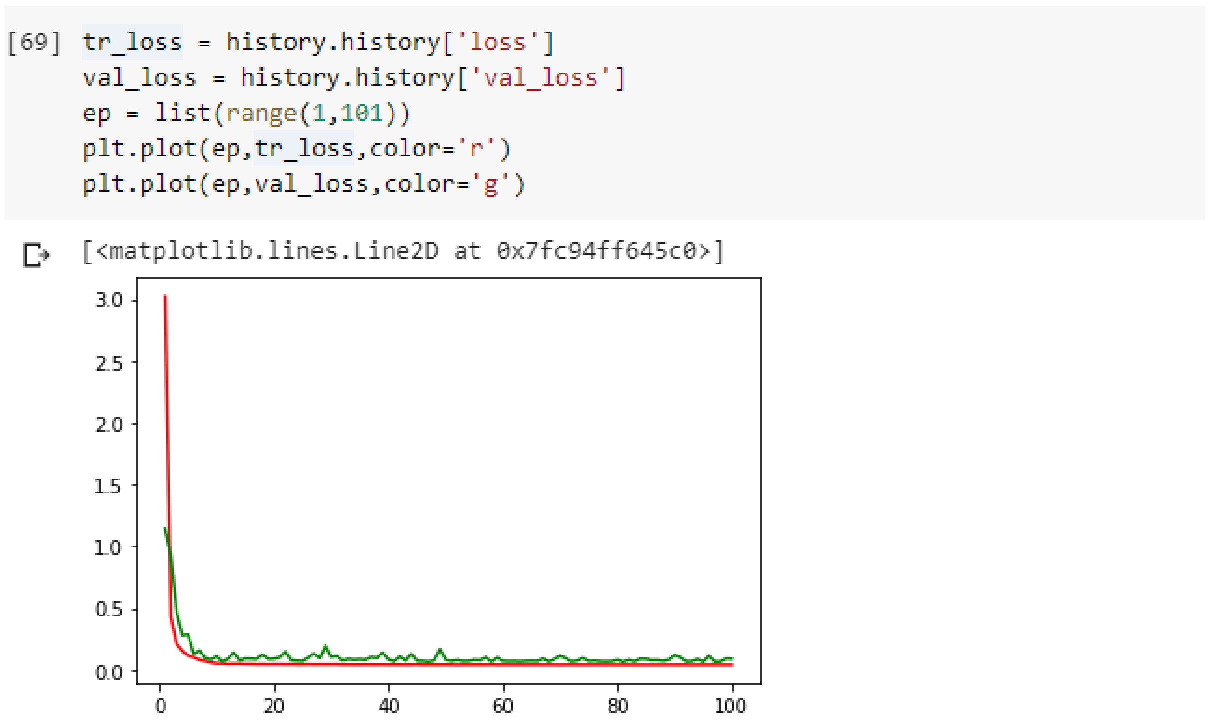
```
In [67]: X_train_sc = X_train_sc.values.reshape(21829,7,1)
         X_test_sc = X_test_sc.values.reshape(5458,7,1)

In [68]: history = model.fit(X_train_sc,y_train,epochs=100,validation_data=(X_test_sc,y_test))

         Train on 21829 samples, validate on 5458 samples
         Epoch 1/100
         21829/21829 [==============================] - 21s 951us/step - loss: 3.0211 - val_loss: 1.1422
         Epoch 2/100
         21829/21829 [==============================] - 20s 927us/step - loss: 0.4303 - val_loss: 0.9242
         Epoch 3/100
         21829/21829 [==============================] - 20s 933us/step - loss: 0.2079 - val_loss: 0.4672
         Epoch 4/100
         21829/21829 [==============================] - 20s 936us/step - loss: 0.1511 - val_loss: 0.2796
         Epoch 5/100
         21829/21829 [==============================] - 20s 929us/step - loss: 0.1159 - val_loss: 0.2860
         Epoch 6/100
         21829/21829 [==============================] - 20s 935us/step - loss: 0.1009 - val_loss: 0.1281
         Epoch 7/100
         21829/21829 [==============================] - 20s 938us/step - loss: 0.0811 - val_loss: 0.1557
         Epoch 8/100
         21829/21829 [==============================] - 21s 958us/step - loss: 0.0722 - val_loss: 0.0969
         Epoch 9/100
         21829/21829 [==============================] - 20s 933us/step - loss: 0.0609 - val_loss: 0.0841
```

```
Epoch 94/100
21829/21829 [==============================] - 21s 944us/step - loss: 0.0426 - val_loss: 0.0868
Epoch 95/100
21829/21829 [==============================] - 21s 944us/step - loss: 0.0424 - val_loss: 0.0662
Epoch 96/100
21829/21829 [==============================] - 21s 941us/step - loss: 0.0422 - val_loss: 0.1119
Epoch 97/100
21829/21829 [==============================] - 21s 940us/step - loss: 0.0424 - val_loss: 0.0665
Epoch 98/100
21829/21829 [==============================] - 20s 936us/step - loss: 0.0422 - val_loss: 0.0686
Epoch 99/100
21829/21829 [==============================] - 21s 964us/step - loss: 0.0422 - val_loss: 0.0920
Epoch 100/100
21829/21829 [==============================] - 21s 947us/step - loss: 0.0424 - val_loss: 0.0888
```

Fig 8.3.10.1: epoch till 1 to 100

The value of epoch is decreasing from top to bottom of loss and validation loss. Here we are taking epoch values till 100 only. At the epoch of 1/100 the values of loss at 0.04 and the value of validation loss was at 0.07. At the end of the epoch 100/100 its decreased to 0.04 and validation loss at 0.08.

8.3.11: Graph representation of loss and val_loss:



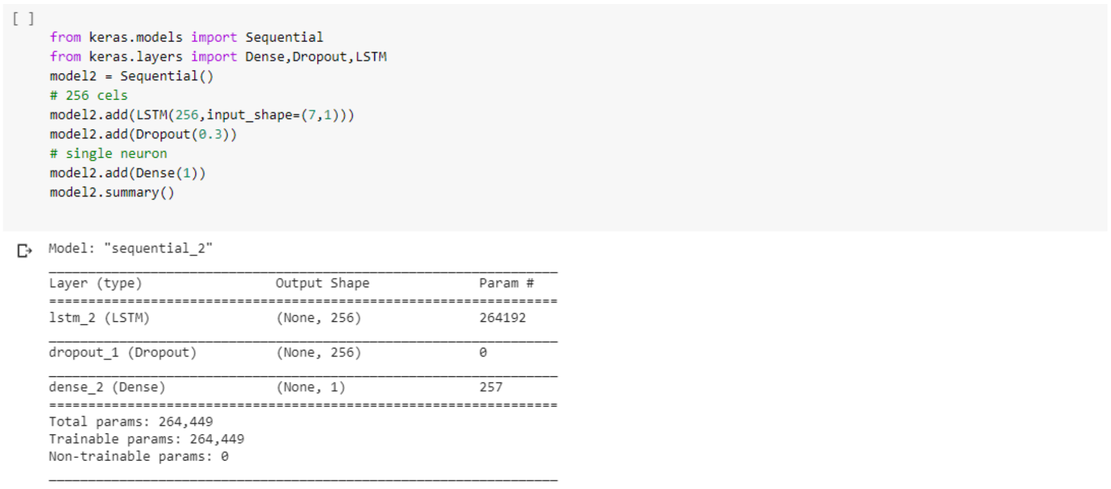
```
[69] tr_loss = history.history['loss']
     val_loss = history.history['val_loss']
     ep = list(range(1,101))
     plt.plot(ep,tr_loss,color='r')
     plt.plot(ep,val_loss,color='g')
```

```
[<matplotlib.lines.Line2D at 0x7fc94ff645c0>]
```

Fig 8.3.11.1: variation between loss and val_loss.

Here we are taking the history of loss and history of validation loss where in x axis we have validation _loss and in y axis we have loss and the above plot represents slight changes between 0-40.

8.3.12: Importing the packages (for model2):



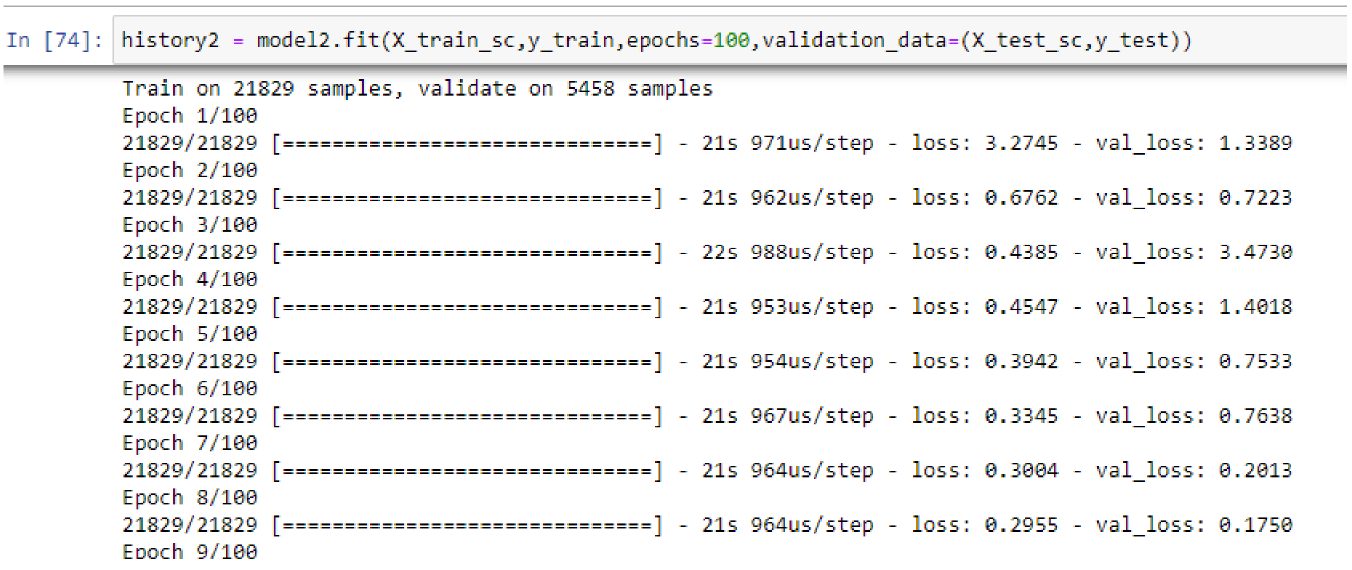
```
[ ]
    from keras.models import Sequential
    from keras.layers import Dense,Dropout,LSTM
    model2 = Sequential()
    # 256 cels
    model2.add(LSTM(256,input_shape=(7,1)))
    model2.add(Dropout(0.3))
    # single neuron
    model2.add(Dense(1))
    model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 256)               264192    
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257       
=================================================================
Total params: 264,449
Trainable params: 264,449
Non-trainable params: 0
_________________________________________________________________
```

Fig 8.3.12.1: importing the keras

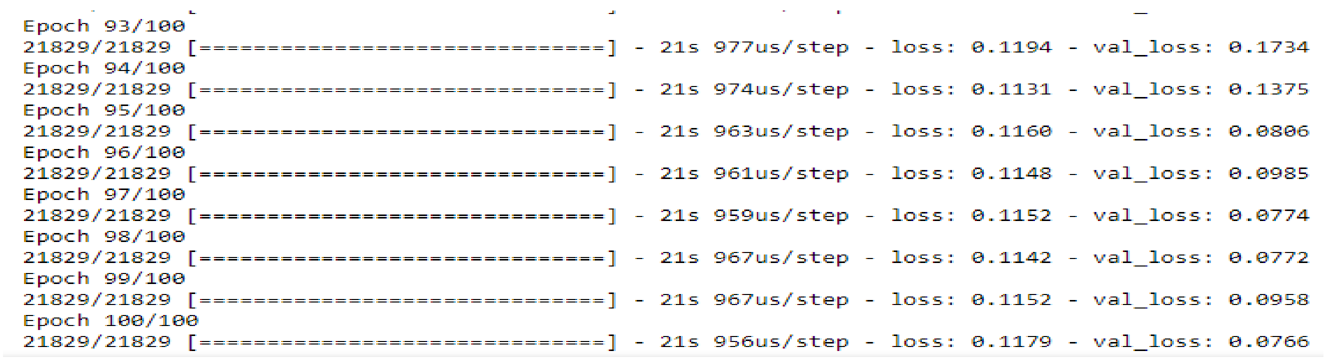
In the above we have imported the packages and drop the columns for further better plotting the data.

8.3.13: history of model2:



```
In [74]: history2 = model2.fit(X_train_sc,y_train,epochs=100,validation_data=(X_test_sc,y_test))
```

```
Train on 21829 samples, validate on 5458 samples
Epoch 1/100
21829/21829 [==============================] - 21s 971us/step - loss: 3.2745 - val_loss: 1.3389
Epoch 2/100
21829/21829 [==============================] - 21s 962us/step - loss: 0.6762 - val_loss: 0.7223
Epoch 3/100
21829/21829 [==============================] - 22s 988us/step - loss: 0.4385 - val_loss: 3.4730
Epoch 4/100
21829/21829 [==============================] - 21s 953us/step - loss: 0.4547 - val_loss: 1.4018
Epoch 5/100
21829/21829 [==============================] - 21s 954us/step - loss: 0.3942 - val_loss: 0.7533
Epoch 6/100
21829/21829 [==============================] - 21s 967us/step - loss: 0.3345 - val_loss: 0.7638
Epoch 7/100
21829/21829 [==============================] - 21s 964us/step - loss: 0.3004 - val_loss: 0.2013
Epoch 8/100
21829/21829 [==============================] - 21s 964us/step - loss: 0.2955 - val_loss: 0.1750
Epoch 9/100
```

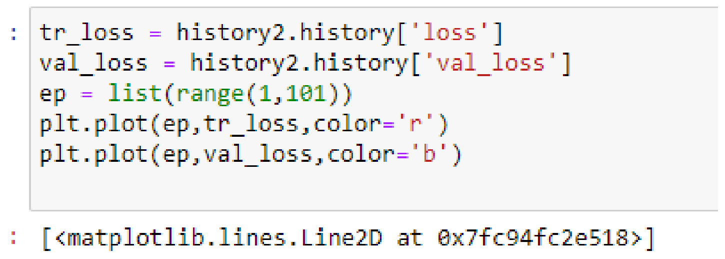


```
Epoch 93/100
21829/21829 [==============================] - 21s 977us/step - loss: 0.1194 - val_loss: 0.1734
Epoch 94/100
21829/21829 [==============================] - 21s 974us/step - loss: 0.1131 - val_loss: 0.1375
Epoch 95/100
21829/21829 [==============================] - 21s 963us/step - loss: 0.1160 - val_loss: 0.0806
Epoch 96/100
21829/21829 [==============================] - 21s 961us/step - loss: 0.1148 - val_loss: 0.0985
Epoch 97/100
21829/21829 [==============================] - 21s 959us/step - loss: 0.1152 - val_loss: 0.0774
Epoch 98/100
21829/21829 [==============================] - 21s 967us/step - loss: 0.1142 - val_loss: 0.0772
Epoch 99/100
21829/21829 [==============================] - 21s 967us/step - loss: 0.1152 - val_loss: 0.0958
Epoch 100/100
21829/21829 [==============================] - 21s 956us/step - loss: 0.1179 - val_loss: 0.0766
```

Fig 8.3.13.1 epoch till 1 to 100

The value of epoch is decreasing from top to bottom of loss and validation loss. Here we are taking epoch values till 100 only. At the epoch of 1/100 the values of loss at 3.2 and the value of validation loss was at 1.3 At the end of the epoch 100/100 its decreased to 0.11 and validation loss at 0.07. The values of epoch of history 1 and history 2 is completely different.

8.3.14: Graph represents the loss and val_loss:



```
: tr_loss = history2.history['loss']
  val_loss = history2.history['val_loss']
  ep = list(range(1,101))
  plt.plot(ep,tr_loss,color='r')
  plt.plot(ep,val_loss,color='b')

: [<matplotlib.lines.Line2D at 0x7fc94fc2e518>]
```

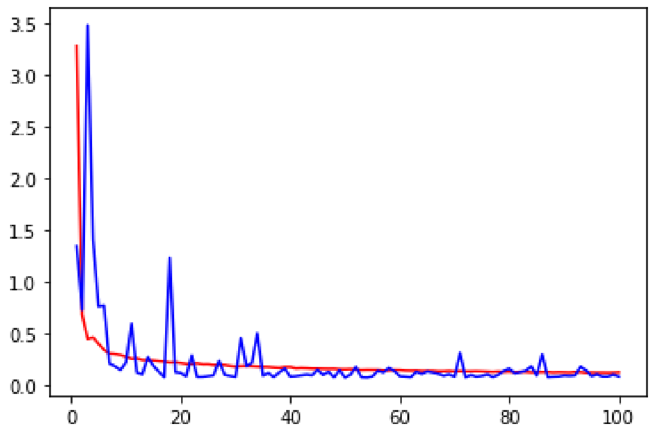
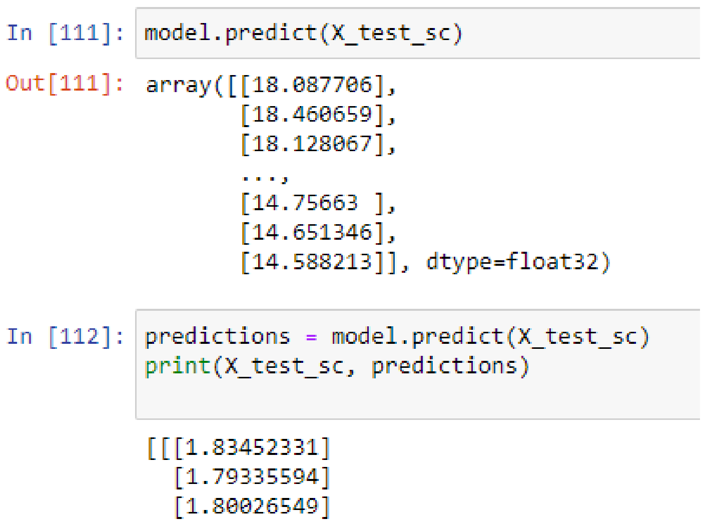


Fig 8.3.14.1: variation between loss and val_loss

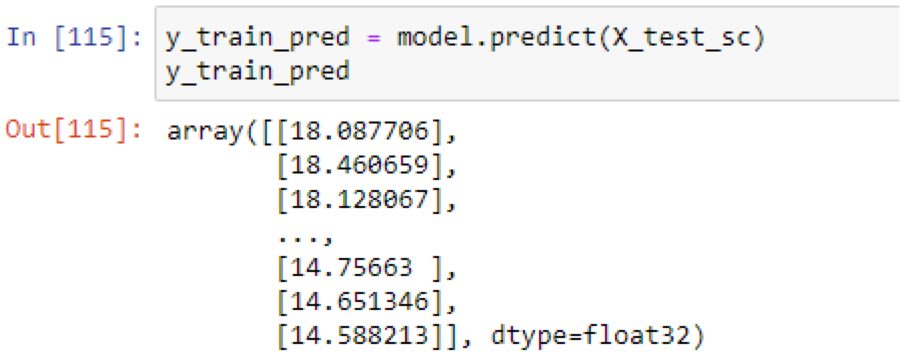
Here we are taking the history of loss and history of validation loss where in x axis we have validation _loss and in y axis we have loss and the above plot represents very vast changes between 0-80.

8.3.15: Make Prediction:



```
In [111]: model.predict(X_test_sc)

Out[111]: array([[18.087706],
                 [18.460659],
                 [18.128067],
                 ...,
                 [14.75663 ],
                 [14.651346],
                 [14.588213]], dtype=float32)

In [112]: predictions = model.predict(X_test_sc)
          print(X_test_sc, predictions)

          [[[1.83452331]
            [1.79335594]
            [1.80026549]
```

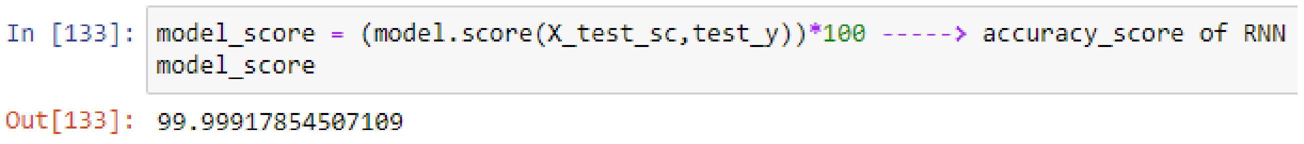
Fig 8.3.15.1: Predicting the x_test



```
In [115]: y_train_pred = model.predict(X_test_sc)
          y_train_pred

Out[115]: array([[18.087706],
                 [18.460659],
                 [18.128067],
                 ...,
                 [14.75663 ],
                 [14.651346],
                 [14.588213]], dtype=float32)
```

Fig 8.3.15.2: testing the predicted model.

8.3.16: Accuracy score:



```
In [133]: model_score = (model.score(X_test_sc,test_y))*100 -----> accuracy_score of RNN
          model_score

Out[133]: 99.99917854507109
```

Fig 8.3.16.1 accuracy score of testing data.

We have got the accuracy score for RNN as 99.9

8.3.17: COMPARIOSN OF RNN:

CONCLUSION: When compared to the history2 math plot graph the history1 math plot graph Is much accurate as there in no drastic changes when the loss and validation loss lines. In the history2 there is changes happening between 0-40 of total epoch of 100.

```
[<matplotlib.lines.Line2D at 0x7ff76b46ddd8>]
```

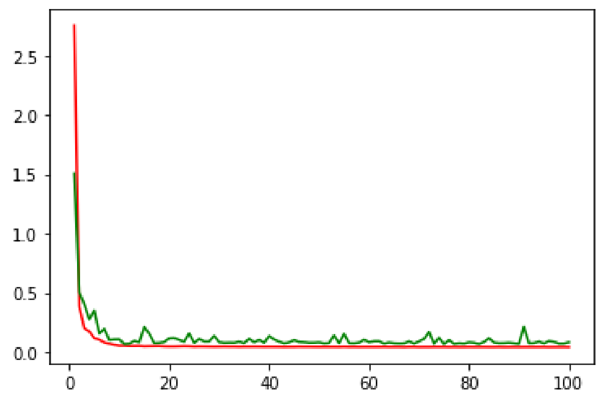


Fig 8.3.17.1 history1 graph

```
[<matplotlib.lines.Line2D at 0x7ff76adcc7b8>]
```

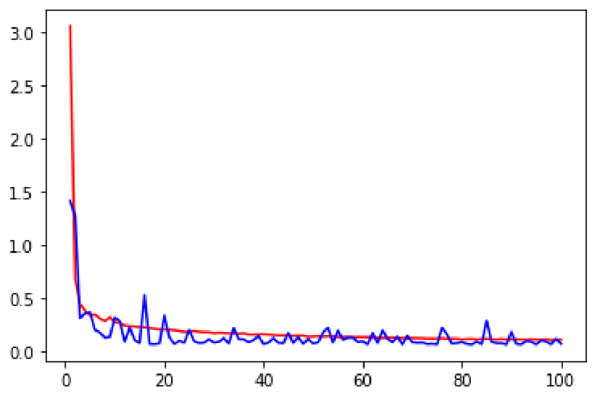


Fig 8.3.17.2 history2 graph

CHAPTER 9

Comparing the performances of all the models:

On comparing the accuracy score with respect to training dataset of all models it has been observed that Linear regression has better performance than RNN.

On comparing the accuracy score with respect to testing dataset of all models it has been observed that Linear has better score than RNN.

CHAPTER 10
REFERENCES :

https://www.bgc-jena.mpg.de/wetter/

https://www.kaggle.com/davidbnn92/weather-data

https://www.geeksforgeeks.org/python-find-current-weather-of-any-city-using-openweathermap-api/

GITHUB LINK : https://github.com/shraavyakunch/Weather-forcasting-project-