# Mini-Project Report

On

**Weather forecasting**

### **Submitted as partial fulfillment for the award of**

#### **BACHELOR OF TECHNOLOGY DEGREE**

**Session 2021-22**

in

###### **Computer Science & Engineering**

**Submitted By:**

**NAVEEN DIWEDI (1900970100073)**

**NIKHIL GUPTA (1900970100075)**

**NIKHIL VARSHNEY (1900970100076)**

**PRANSHU GOYAL (1900970100082)**

Under the guidance of:

**Mayank Dixit**

**Designation**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS COLLEGE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

Logo

Description automatically generated

**AFFILIATED TO**

**DR. A.P.J ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW, UP**

**(Formerly UPTU)**

|  |  |
| --- | --- |
| **GEI logo** | **GALGOTIAS COLLEGE OF ENGINEERING & TECHNOLOGY**  **GREATER NOIDA, UTTER PRADESH, INDIA- 201306.** |

# CERTIFICATE

This is to certify that the mini-project report entitled “**WEATHER FORECASTING**” submitted by Mr./Ms. < **NAVEEN DIWEDI > <1900970100073> OF STUDENT 1 Roll No.,** Mr./Ms. < **NIKHIL GUPTA > <1900970100075> OF STUDENT 2,** Mr./Ms. < **NIKHIL VARSHNEY > <1900970100076> OF STUDENT 3,**Mr./Ms.< **PRANSHU GOYAL** > <**1900970100082**> **OF STUDENT 4** to the Galgotias College of Engineering & Technology, Greater Noida, Utter Pradesh, affiliated to Dr. A.P.J. Abdul Kalam Technical University Lucknow, Uttar Pradesh in partial fulfillment for the award of Degree of Bachelor of Technology in Computer science & Engineering is a bonafide record of the project work carried out by them under my supervision during the year 2021-2022.

|  |  |
| --- | --- |
| **Name** **(Project Coordinator)**  **Designation**  **Deptt. of CSE** | **Dr. Vishnu Sharma**  **Professor and Head**  **Deptt. of CSE** |

**ABSTRACT**

In this poster, We are helping Rural Farmers of India by building Machine learning Model to predict the amount of rainfall in mili meter(mm) in upcoming months of monsoon so that it all farmers to grow their crops productively and get a maximum profit. we train the model and predict accurate on recent data with ML techniques .

Predicting rainfall is an important step in generating data for climate impact studies. Rainfall predictions are a key process for providing climate impact assessments with inputs. A consistent rainfall pattern is typically good for normal plants; nevertheless, too much or too little rainfall can be disastrous to crops, even deadly. Drought can damage plants and lead to erosion, while heavy rainfall can encourage the growth of destructive fungi. Machine Learning (ML) can be helpful in overcoming such issues; for example, ML can be used to predict rainfall and apply it to foresee crop health and yield. Predictive analysis is a subset of data mining that forecasts future probabilities and patterns. Various sectors like the Agricultural Produce Markets Committee (APMC), Kisaan call centre, etc., can use proposed method, enabling the sector and farmers to obtain information on future precipitation, crop yields and crop health.

As we know agriculture was the predominant of our country and economy. While a regular rain pattern is usually played vital for healthy agriculture but too much rainfall or too little rainfall can be harmful, even it led to devastating of crops. This paper discusses the rate of rainfall in previous years according to various crops seasons like rabi, Kharif, zaid and predicts the rainfall in future seasons. The paper also measures the different categories of data by linear regression method in metrics for effective understanding of agriculture in India. We have selected a real dataset which consists of past year’s rainfall rate according to various seasons. Results of this application help farmers to make a correct decision to harvest a particular crop accordingly to crops seasons. Linear regression helps to find

**Table of Contents**

|  |  |
| --- | --- |
| **Sr. No.** | **Content** |
| **1.** | **Introduction**   * 1. motivation   2. Description of Theoretical Concept |
| **2.** | **Literature review**  2.1Related Literature review |
| **3.** | **Problem statement** |
| **4.** | **Proposed work and system design**  4.1 Proposed work  4.2 System Design |
| **5.** | **IMPLEMENTATION**   * 1. Experimental Setup:   5.2 Dataset Description: |
| **6.** | **RESULT AND DEMONSTRATION**  6.1 Performance measure  6.2 Results Analysis |
| **7.** | 7.1 OBJECTIVE AND RELEVENCE OF PROJECT:  7.2 TECHNICAL NOVELITY :  7.3 Expected Outcomes: |
| **8.** | 8.1 Conclusion  8.2 Limitations:  8.3 Future scope: | |
| **9.** | References | |

**Chapter 1**

**Introduction**

Today climate change is a global Issue as we have seen that many geographical and catastrophe due to climate change. Because we don't have accurate models for predicting rainfall quantity in future.in order to solve this problem so this model will not only restrict to help farmers,

In the current situation, rainfall is regarded as the one of the sole causes of most important traits. In India farming is seen as one of the key factors in deciding the country's economy, and agriculture depends solely on precipitation. The prediction of rainfall is important since heavy or irregular precipitation leads to crop devastation and destruction of property. The proposed prediction method and the prediction for rainfall outcomes can help farmers increase crop yield and detect crop health. It can also assist farmers in making more efficient use of water resources, resulting in increased crop productivity and crop health. Apart from that, knowing the amount of rainfall in a certain location is critical in coastal locations all over the world. To establish a rainwater harvester in some of the places where there is a water scarcity, rainfall forecasting should be done ahead of time. Predicting rainfall is a difficult task, and the findings must be correct. The goal of this project is to use Machine Learning (ML) to predict rainfall. The proposed project will focus on a comparison of different machine learning (ML) algorithms for rainfall prediction. Because various methods provide various levels of accuracy, it is critical to select the appropriate method and model it according to the needs for reliable rainfall forecast. Pre-processing of the data obtained for rainfall prediction is necessary, which includes data cleaning and normalizing of weather factors. The processed data (meteorological parameters) will then be used to predict rainfall. Prediction results can then be applied to various aspects of agriculture. The agricultural aspects that can be derived from the prediction results include crop yield estimation under current rainfall predictions, crop health identification in the future, and soil conditions for the predicted rainfall.

**1.1 Motivation**

In this poster , We are helping Rural Farmers of India by building Machine learning Model to predict the amount of rainfall in milimeter(mm) in upcoming months of monsoon so that it all farmers to grow their crops productively and get a maximum profit.we train the model and predict accurate on recent data with ML techniques .

it can also help in resolving several major issue like disaster management , agriculture areas and weather forecasting etc.

**1.2 Description of Theoretical Concept**

**NUMPY:**

# What is NumPy?

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
* NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
* A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today’s scientific/mathematical Python-based software, just knowing how to use Python’s built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

The points about sequence size and speed are particularly important in scientific computing. As a simple example, consider the case of multiplying each element in a 1-D sequence with the corresponding element in another sequence of the same length. If the data are stored in two Python lists, a and b, we could iterate over each element:

c **=** **[]**

**for** i **in** range**(**len**(**a**)):**

c**.**append**(**a**[**i**]\***b**[**i**])**

This produces the correct answer, but if a and b each contain millions of numbers, we will pay the price for the inefficiencies of looping in Python. We could accomplish the same task much more quickly in C by writing (for clarity we neglect variable declarations and initializations, memory allocation, etc.)

**for** **(**i **=** **0;** i **<** rows**;** i**++):** **{**

c**[**i**]** **=** a**[**i**]\***b**[**i**];**

**}**

This saves all the overhead involved in interpreting the Python code and manipulating Python objects, but at the expense of the benefits gained from coding in Python. Furthermore, the coding work required increases with the dimensionality of our data. In the case of a 2-D array, for example, the C code (abridged as before) expands to

**for** **(**i **=** **0;** i **<** rows**;** i**++):** **{**

**for** **(**j **=** **0;** j **<** columns**;** j**++):** **{**

c**[**i**][**j**]** **=** a**[**i**][**j**]\***b**[**i**][**j**];**

**}**

**}**

NumPy gives us the best of both worlds: element-by-element operations are the “default mode” when an ndarray is involved, but the element-by-element operation is speedily executed by pre-compiled C code. In NumPy

c **=** a **\*** b

does what the earlier examples do, at near-C speeds, but with the code simplicity we expect from something based on Python. Indeed, the NumPy idiom is even simpler! This last example illustrates two of NumPy’s features which are the basis of much of its power: vectorization and broadcasting.

## Why is NumPy Fast?

Vectorization describes the absence of any explicit looping, indexing, etc., in the code - these things are taking place, of course, just “behind the scenes” in optimized, pre-compiled C code. Vectorized code has many advantages, among which are:

* vectorized code is more concise and easier to read
* fewer lines of code generally means fewer bugs
* the code more closely resembles standard mathematical notation (making it easier, typically, to correctly code mathematical constructs)
* vectorization results in more “Pythonic” code. Without vectorization, our code would be littered with inefficient and difficult to read for loops.

Broadcasting is the term used to describe the implicit element-by-element behavior of operations; generally speaking, in NumPy all operations, not just arithmetic operations, but logical, bit-wise, functional, etc., behave in this implicit element-by-element fashion, i.e., they broadcast. Moreover, in the example above, a and b could be multidimensional arrays of the same shape, or a scalar and an array, or even two arrays of with different shapes, provided that the smaller array is “expandable” to the shape of the larger in such a way that the resulting broadcast is unambiguous. For detailed “rules” of broadcasting see [Broadcasting](https://numpy.org/devdocs/user/basics.broadcasting.html#basics-broadcasting).

**PANDAS:**

# What kind of data does pandas handle?

* I want to start using pandas
* **In [1]: import** **pandas** **as** **pd**

To load the pandas package and start working with it, import the package. The community agreed alias for pandas is pd, so loading pandas as pd is assumed standard practice for all of the pandas documentation.

## pandas data table representation

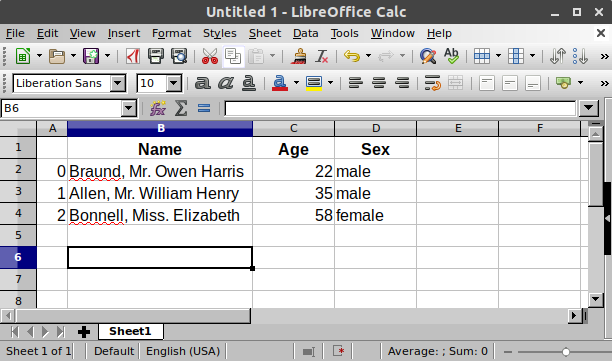
* I want to store passenger data of the Titanic. For a number of passengers, I know the name (characters), age (integers) and sex (male/female) data.
* **In [2]:** df = pd.DataFrame(
* **...:**  {
* **...:**  "Name": [
* **...:**  "Braund, Mr. Owen Harris",
* **...:**  "Allen, Mr. William Henry",
* **...:**  "Bonnell, Miss. Elizabeth",
* **...:**  ],
* **...:**  "Age": [22, 35, 58],
* **...:**  "Sex": ["male", "male", "female"],
* **...:**  }
* **...:** )
* **...:**
* **In [3]:** df
* **Out[3]:**
* Name Age Sex
* 0 Braund, Mr. Owen Harris 22 male
* 1 Allen, Mr. William Henry 35 male
* 2 Bonnell, Miss. Elizabeth 58 female

To manually store data in a table, create a DataFrame. When using a Python dictionary of lists, the dictionary keys will be used as column headers and the values in each list as columns of the DataFrame.

A [**DataFrame**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame) is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, categorical data and more) in columns. It is similar to a spreadsheet, a SQL table or the data.frame in R.

* The table has 3 columns, each of them with a column label. The column labels are respectively Name, Age and Sex.
* The column Name consists of textual data with each value a string, the column Age are numbers and the column Sex is textual data.

In spreadsheet software, the table representation of our data would look very similar:



## Each column in a DataFrame is a Series

* I’m just interested in working with the data in the column Age
* **In [4]:** df["Age"]
* **Out[4]:**
* 0 22
* 1 35
* 2 58
* Name: Age, dtype: int64

When selecting a single column of a pandas **[DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)**, the result is a pandas [**Series**](https://pandas.pydata.org/docs/reference/api/pandas.Series.html#pandas.Series). To select the column, use the column label in between square brackets [].

**Note**

If you are familiar to Python [dictionaries](https://docs.python.org/3/tutorial/datastructures.html#tut-dictionaries), the selection of a single column is very similar to selection of dictionary values based on the key.

You can create a Series from scratch as well:

**In [5]:** ages = pd.Series([22, 35, 58], name="Age")

**In [6]:** ages

**Out[6]:**

0 22

1 35

2 58

Name: Age, dtype: int64

A pandas Series has no column labels, as it is just a single column of a DataFrame. A Series does have row labels.

## Do something with a DataFrame or Series

* I want to know the maximum Age of the passengers

We can do this on the DataFrame by selecting the Age column and applying max():

**In [7]:** df["Age"].max()

**Out[7]:** 58

Or to the Series:

**In [8]:** ages.max()

**Out[8]:** 58

As illustrated by the max() method, you can do things with a DataFrame or Series. pandas provides a lot of functionalities, each of them a method you can apply to a DataFrame or Series. As methods are functions, do not forget to use parentheses ().

* I’m interested in some basic statistics of the numerical data of my data table
* **In [9]:** df.describe()
* **Out[9]:**
* Age
* count 3.000000
* mean 38.333333
* std 18.230012
* min 22.000000
* 25% 28.500000
* 50% 35.000000
* 75% 46.500000
* max 58.000000

The [**describe()**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html#pandas.DataFrame.describe) method provides a quick overview of the numerical data in a DataFrame. As the Name and Sex columns are textual data, these are by default not taken into account by the [**describe()**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html#pandas.DataFrame.describe) method.

Many pandas operations return a DataFrame or a Series. The [**describe()**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html#pandas.DataFrame.describe) method is an example of a pandas operation returning a pandas Series or a pandas DataFrame.

# How do I read and write tabular data?

* I want to analyze the Titanic passenger data, available as a CSV file.
* **In [2]:** titanic = pd.read\_csv("data/titanic.csv")

pandas provides the **[read\_csv()](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html" \l "pandas.read_csv" \o "pandas.read_csv)** function to read data stored as a csv file into a pandas DataFrame. pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, …), each of them with the prefix read\_\*.

Make sure to always have a check on the data after reading in the data. When displaying a DataFrame, the first and last 5 rows will be shown by default:

**In [3]:** titanic

**Out[3]:**

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

.. ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

[891 rows x 12 columns]

* I want to see the first 8 rows of a pandas DataFrame.
* **In [4]:** titanic.head(8)
* **Out[4]:**
* PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked
* 0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S
* 1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C
* 2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S
* 3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S
* 4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S
* 5 6 0 3 Moran, Mr. James male NaN 0 0 330877 8.4583 NaN Q
* 6 7 0 1 McCarthy, Mr. Timothy J male 54.0 0 0 17463 51.8625 E46 S
* 7 8 0 3 Palsson, Master. Gosta Leonard male 2.0 3 1 349909 21.0750 NaN S

To see the first N rows of a DataFrame, use the [**head()**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html#pandas.DataFrame.head) method with the required number of rows (in this case 8) as argument.

**Note**

Interested in the last N rows instead? pandas also provides a [**tail()**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.tail.html#pandas.DataFrame.tail) method. For example, titanic.tail(10) will return the last 10 rows of the DataFrame.

A check on how pandas interpreted each of the column data types can be done by requesting the pandas dtypes attribute:

**In [5]:** titanic.dtypes

**Out[5]:**

PassengerId int64

Survived int64

Pclass int64

Name object

Sex object

Age float64

SibSp int64

Parch int64

Ticket object

Fare float64

Cabin object

Embarked object

dtype: object

For each of the columns, the used data type is enlisted. The data types in this DataFrame are integers (int64), floats (float64) and strings (object).

**Note**

When asking for the dtypes, no brackets are used! dtypes is an attribute of a DataFrame and Series. Attributes of DataFrame or Series do not need brackets. Attributes represent a characteristic of a DataFrame/Series, whereas a method (which requires brackets) do something with the DataFrame/Series as introduced in the [first tutorial](https://pandas.pydata.org/docs/getting_started/intro_tutorials/01_table_oriented.html#min-tut-01-tableoriented).

* My colleague requested the Titanic data as a spreadsheet.
* **In [6]:** titanic.to\_excel("titanic.xlsx", sheet\_name="passengers", index=**False**)

Whereas read\_\* functions are used to read data to pandas, the to\_\* methods are used to store data. The **[to\_excel()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html" \l "pandas.DataFrame.to_excel" \o "pandas.DataFrame.to_excel)** method stores the data as an excel file. In the example here, the sheet\_name is named passengers instead of the default Sheet1. By setting index=False the row index labels are not saved in the spreadsheet.

The equivalent read function **read\_excel()** will reload the data to a DataFrame:

**In [7]:** titanic = pd.read\_excel("titanic.xlsx", sheet\_name="passengers")

**In [8]:** titanic.head()

**Out[8]:**

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

* I’m interested in a technical summary of a DataFrame
* **In [9]:** titanic.info()
* <class 'pandas.core.frame.DataFrame'>
* RangeIndex: 891 entries, 0 to 890
* Data columns (total 12 columns):
* # Column Non-Null Count Dtype
* --- ------ -------------- -----
* 0 PassengerId 891 non-null int64
* 1 Survived 891 non-null int64
* 2 Pclass 891 non-null int64
* 3 Name 891 non-null object
* 4 Sex 891 non-null object
* 5 Age 714 non-null float64
* 6 SibSp 891 non-null int64
* 7 Parch 891 non-null int64
* 8 Ticket 891 non-null object
* 9 Fare 891 non-null float64
* 10 Cabin 204 non-null object
* 11 Embarked 889 non-null object
* dtypes: float64(2), int64(5), object(5)
* memory usage: 83.7+ KB

The method [**info()**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html#pandas.DataFrame.info) provides technical information about a DataFrame, so let’s explain the output in more detail:

* + It is indeed a **[DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)**.
  + There are 891 entries, i.e. 891 rows.
  + Each row has a row label (aka the index) with values ranging from 0 to 890.
  + The table has 12 columns. Most columns have a value for each of the rows (all 891 values are non-null). Some columns do have missing values and less than 891 non-null values.
  + The columns Name, Sex, Cabin and Embarked consists of textual data (strings, aka object). The other columns are numerical data with some of them whole numbers (aka integer) and others are real numbers (aka float).
  + The kind of data (characters, integers,…) in the different columns are summarized by listing the dtypes.
  + The approximate amount of RAM used to hold the DataFrame is provided as well.

**Chapter 2**

**Literature review**

**2.1 Related Literature review**

Literature review has 5 objectives:

1. Motivate your work by explaining its social value.
2. Provide background context for your readers.
3. your findings.
4. Justify your experiment method.
5. your project’s scientific novelty.

In this mini project , We are building Machine learning Model to predict the amount of rainfall in milimeter(mm) in upcoming months of monsoon so that it all farmers to grow their crops productively and get a maximum profit.we train the model and predict accurate on recent data with ML techniques.

Model Predict Predicting the rainfall in mm happening in september month on the basis of previous months of monsoons like June , July , August. Model can be used :to build platform for various sector reserach and development purpose to deploy web application for end users like farmer and weather forecast department.

pred **=** reg**.**predict(X\_test)

In [57]:

lst**=** pred**.**tolist()

In [58]:

lst

Out[58]:

[[195.44375751678064],

[180.75858505109832],

[185.27796481331825],

[162.89446099067428],

[148.02597135434692],

[203.97969176276843],

[170.73949623462318],

[182.90828096438977],

[175.1271451745628],

[181.5267908454426],

[163.23329366482062],

[200.9515275254538],

[165.55755618749245],

[209.16777945276868],

[173.78805332373395],

[148.73270871548473],

[109.76867809177611],

[160.7276092007735],

[201.33316158774855],

[192.00196022750222],

[197.1908489711251],

[162.40849358749625]]

df **=** pd**.**DataFrame(lst)

df

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[59]:   |  | **0** | | --- | --- | | **0** | 195.443758 | | **1** | 180.758585 | | **2** | 185.277965 | | **3** | 162.894461 | | **4** | 148.025971 | | **5** | 203.979692 | | **6** | 170.739496 | | **7** | 182.908281 | | **8** | 175.127145 | | **9** | 181.526791 | | **10** | 163.233294 | | **11** | 200.951528 | | **12** | 165.557556 | | **13** | 209.167779 | | **14** | 173.788053 | | **15** | 148.732709 | | **16** | 109.768678 | | **17** | 160.727609 | | **18** | 201.333162 | | **19** | 192.001960 | | **20** | 197.190849 | | **21** | 162.408494 |   **C:\Users\naveen\Pictures\Screenshots\Screenshot (3961).png** |  |
| **C:\Users\naveen\Pictures\Screenshots\Screenshot (3962).png** |  |
|  |  |

It can use in weather forecasting .

It can use in Disaster management.

It can use in Agriculture areas.

**CHAPTER 3**

**Problem statement**

In this mini project we building a Machine learning Model Predict Predicting the rainfall in mm happening in september month on the basis of previous months of monsoons like June , July , August.

**Depiction of problem statement**

**1.input**

Recent weather Datasets used from data.gov.in .It contain rainfall in mm during 3 months June,July, august .

Graphical user interface, application, table, Excel

Description automatically generated Graphical user interface, application, table, Excel

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated Graphical user interface, application, table, Excel

Description automatically generated

**2.Output**

Graphical user interface, application, table, Excel

Description automatically generated Graphical user interface, application, table, Excel

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

**Process**

* Collect and prepare data
* choose algorithm
* train algo. using trainig data
* evaluate model performance

**Chapter 4**

**Proposed work and system design**

**4.1 Proposed work:**

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

**Parameters**

**fit\_intercept*bool, default=True***

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

**normalize*bool, default=False***

This parameter is ignored when fit\_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use **[StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler" \o "sklearn.preprocessing.StandardScaler)** before calling fit on an estimator with normalize=False.

*Deprecated since version 1.0:*normalize was deprecated in version 1.0 and will be removed in 1.2.

**copy\_X*bool, default=True***

If True, X will be copied; else, it may be overwritten.

**n\_jobs*int, default=None***

The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly n\_targets > 1 and secondly X is sparse or if positive is set to True. None means 1 unless in a **[joblib.parallel\_backend](https://joblib.readthedocs.io/en/latest/parallel.html" \l "joblib.parallel_backend" \o "(in joblib v1.1.0))** context. -1 means using all processors. See [Glossary](https://scikit-learn.org/stable/glossary.html#term-n_jobs) for more details.

**positive*bool, default=False***

When set to True, forces the coefficients to be positive. This option is only supported for dense arrays.

*New in version 0.24.*

**Attributes**

**coef\_*array of shape (n\_features, ) or (n\_targets, n\_features)***

Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n\_targets, n\_features), while if only one target is passed, this is a 1D array of length n\_features.

**rank\_*int***

Rank of matrix X. Only available when X is dense.

**singular\_*array of shape (min(X, y),)***

Singular values of X. Only available when X is dense.

**intercept\_*float or array of shape (n\_targets,)***

Independent term in the linear model. Set to 0.0 if fit\_intercept = False.

**n\_features\_in\_*int***

Number of features seen during [fit](https://scikit-learn.org/stable/glossary.html#term-fit).

*New in version 0.24.*

**feature\_names\_in\_*ndarray of shape (n\_features\_in\_,)***

Names of features seen during [fit](https://scikit-learn.org/stable/glossary.html#term-fit). Defined only when X has feature names that are all strings.

**Methods**

|  |  |
| --- | --- |
| [**fit**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.fit)(X, y[, sample\_weight]) | Fit linear model. |
| [**get\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.get_params)([deep]) | Get parameters for this estimator. |
| [**predict**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.predict)(X) | Predict using the linear model. |
| [**score**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.score)(X, y[, sample\_weight]) | Return the coefficient of determination of the prediction. |
| [**set\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.set_params)(\*\*params) | Set the parameters of this estimator. |

fit(*X*, *y*, *sample\_weight=None*)[[source]](https://github.com/scikit-learn/scikit-learn/blob/0d378913b/sklearn/linear_model/_base.py#L630)

Fit linear model.

**Parameters**

**X*{array-like, sparse matrix} of shape (n\_samples, n\_features)***

Training data.

**y*array-like of shape (n\_samples,) or (n\_samples, n\_targets)***

Target values. Will be cast to X’s dtype if necessary.

**sample\_weight*array-like of shape (n\_samples,), default=None***

Individual weights for each sample.

*New in version 0.17:*parameter *sample\_weight* support to LinearRegression.

**Returns**

**self*object***

Fitted Estimator.

get\_params(*deep=True*)[[source]](https://github.com/scikit-learn/scikit-learn/blob/0d378913b/sklearn/base.py#L188)

Get parameters for this estimator.

**Parameters**

**deep*bool, default=True***

If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns**

**params*dict***

Parameter names mapped to their values.

predict(*X*)[[source]](https://github.com/scikit-learn/scikit-learn/blob/0d378913b/sklearn/linear_model/_base.py#L348)

Predict using the linear model.

**Parameters**

**X*array-like or sparse matrix, shape (n\_samples, n\_features)***

Samples.

**Returns**

**C*array, shape (n\_samples,)***

Returns predicted values.

score(*X*, *y*, *sample\_weight=None*)[[source]](https://github.com/scikit-learn/scikit-learn/blob/0d378913b/sklearn/base.py#L657)

Return the coefficient of determination of the prediction.

The coefficient of determination R2 is defined as (1−uv), where u is the residual sum of squares ((y\_true - y\_pred)\*\* 2).sum() and v is the total sum of squares ((y\_true - y\_true.mean()) \*\* 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

**Parameters**

**X*array-like of shape (n\_samples, n\_features)***

Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n\_samples, n\_samples\_fitted), where n\_samples\_fitted is the number of samples used in the fitting for the estimator.

**y*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)***

True values for X.

**sample\_weight*array-like of shape (n\_samples,), default=None***

Sample weights.

**Returns**

**score*float***

R2 of self.predict(X) wrt. y.

**Notes**

The R2 score used when calling score on a regressor uses multioutput='uniform\_average' from version 0.23 to keep consistent with default value of [**r2\_score**](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). This influences the score method of all the multioutput regressors (except for **[MultiOutputRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html" \l "sklearn.multioutput.MultiOutputRegressor" \o "sklearn.multioutput.MultiOutputRegressor)**).

set\_params(*\*\*params*)[[source]](https://github.com/scikit-learn/scikit-learn/blob/0d378913b/sklearn/base.py#L212)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as [**Pipeline**](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline)). The latter have parameters of the form <component>\_\_<parameter> so that it’s possible to update each component of a nested object.

**Parameters**

**\*\*params*dict***

Estimator parameters.

**Returns**

**self*estimator instance***

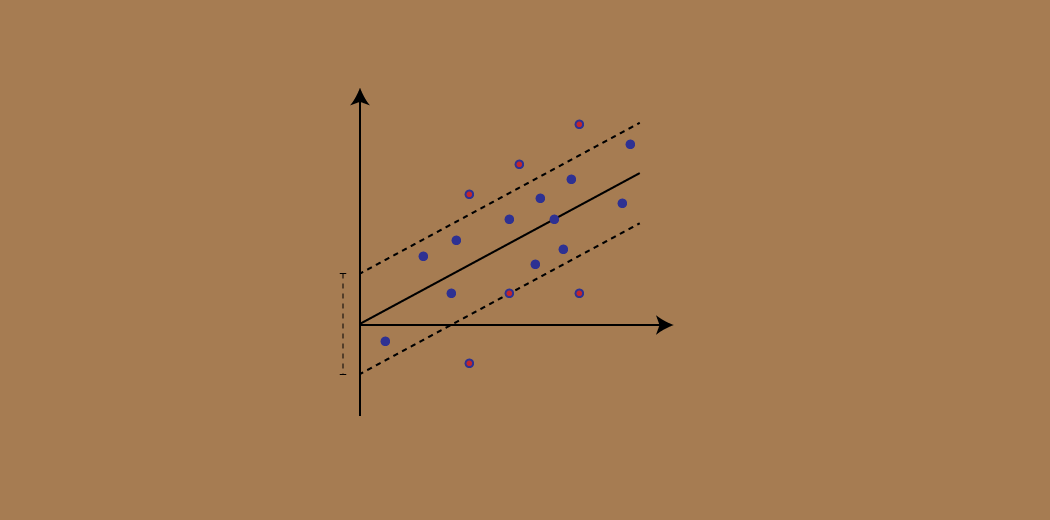
Estimator instance.

**SVM:**

## Support Vector Regression Algorithm

Support Vector Machines (SVM) are popularly and widely used for classification problems in machine learning. I’ve often relied on this not just in machine learning projects but when I want a quick result in a hackathon.

But SVM for regression analysis? I hadn’t even considered the possibility for a while! And even now when I bring up “Support Vector Regression” in front of machine learning beginners, I often get a bemused expression. I understand – most courses and experts don’t even mention Support Vector Regression (SVR) as a machine learning algorithm.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Support-Vector-Regression.gif)

But SVR has its uses as you’ll see in this tutorial. We will first quickly understand what SVM is, before diving into the world of Support Vector Regression and how to implement it in Python!

Note: You can learn about Support Vector Machines and Regression problems in course format here (it’s free!):

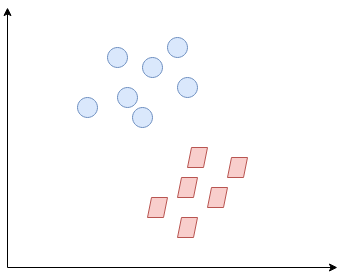
* [*Support Vector Machine (SVM) in Python and R*](https://courses.analyticsvidhya.com/courses/support-vector-machine-svm-in-python-and-r?utm_source=blog&utm_medium=understaing-support-vector-machine-example-code?utm_source=blog&utm_medium=support-vector-regression-tutorial-for-machine-learning)
* [*Fundamentals of Regression Analysis*](https://courses.analyticsvidhya.com/courses/Fundamentals-of-Regression-Analysis?utm_source=blog&utm_medium=support-vector-regression-tutorial-for-machine-learning)

### **Here’s what we’ll cover in this Support Vector Regression tutorial:**

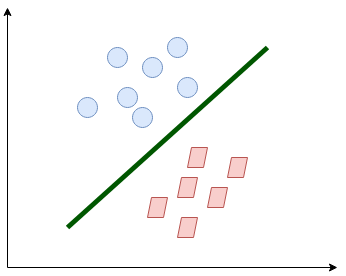
* What is a Support Vector Machine (SVM)?
* Hyperparameters of the Support Vector Machine Algorithm
* Introduction to Support Vector Regression (SVR)
* Implementing Support Vector Regression in Python

## What is a Support Vector Machine (SVM)?

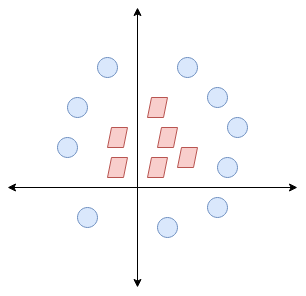
So what exactly is Support Vector Machine (SVM)? We’ll start by understanding SVM in simple terms. Let’s say we have a plot of two label classes as shown in the figure below:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR2.png)

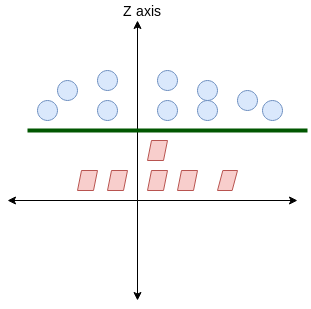
Can you decide what the separating line will be? You might have come up with this:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR3.png)

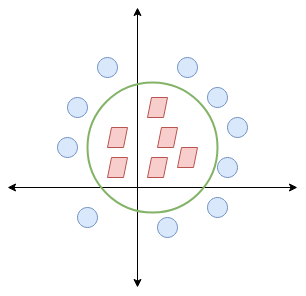
The line fairly separates the classes. This is what SVM essentially does – **simple class separation.**Now, what is the data was like this:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR4.png)

Here, we don’t have a simple line separating these two classes. So we’ll extend our dimension and introduce a new dimension along the z-axis. We can now separate these two classes:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR5.png)

When we transform this line back to the original plane, it maps to the circular boundary as I’ve shown here:

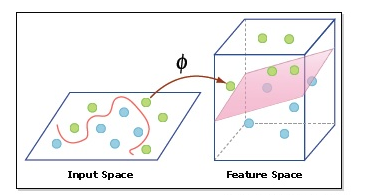
[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR6.png)

This is exactly what SVM does! It tries to find a line/hyperplane (in multidimensional space) that separates these two classes. Then it classifies the new point depending on whether it lies on the positive or negative side of the hyperplane depending on the classes to predict.

## Hyperparameters of the Support Vector Machine (SVM) Algorithm

There are a few important parameters of SVM that you should be aware of before proceeding further:

* **Kernel:** A kernel helps us find a hyperplane in the higher dimensional space without increasing the computational cost. Usually, the computational cost will increase if the dimension of the data increases. This increase in dimension is required when we are unable to find a separating hyperplane in a given dimension and are required to move in a higher dimension:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/hyperplane.png)

* **Hyperplane:**This is basically a separating line between two data classes in SVM. But in Support Vector Regression, this is the line that will be used to predict the continuous output
* **Decision Boundary**: A decision boundary can be thought of as a demarcation line (for simplification) on one side of which lie positive examples and on the other side lie the negative examples. On this very line, the examples may be classified as either positive or negative. This same concept of SVM will be applied in Support Vector Regression as well

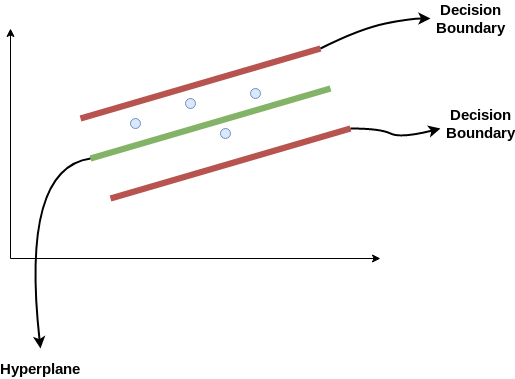
**To understand SVM from scratch, I recommend this tutorial:**[**Understanding Support Vector Machine(SVM) algorithm from examples**](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?utm_source=blog&utm_medium=support-vector-regression-tutorial-for-machine-learning)**.**

## Introduction to Support Vector Regression (SVR)

Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. Let’s spend a few minutes understanding the idea behind SVR.

### **The Idea Behind Support Vector Regression**

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. So let’s now dive deep and understand how SVR works actually.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR1.png)

Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we’ll understand is what is the decision boundary (the danger red line above!). Consider these lines as being at any distance, say ‘a’, from the hyperplane. So, these are the lines that we draw at distance ‘+a’ and ‘-a’ from the hyperplane. This ‘a’ in the text is basically referred to as epsilon.

Assuming that the equation of the hyperplane is as follows:

Y = wx+b (equation of hyperplane)

Then the equations of decision boundary become:

wx+b= +a

wx+b= -a

Thus, any hyperplane that satisfies our SVR should satisfy:

**-a < Y- wx+b < +a**

Our main aim here is to decide a decision boundary at ‘a’ distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

Hence, we are going to take only those points that are within the decision boundary and have the least error rate, or are within the Margin of Tolerance. This gives us a better fitting model.

## Implementing Support Vector Regression (SVR) in Python

Time to put on our coding hats! In this section, we’ll understand the use of Support Vector Regression with the help of a dataset. Here, we have to predict the salary of an employee given a few independent variables. A classic HR analytics project!

**4.2 SYSTEM DESIGN:**

**System requirement :**

4 GB

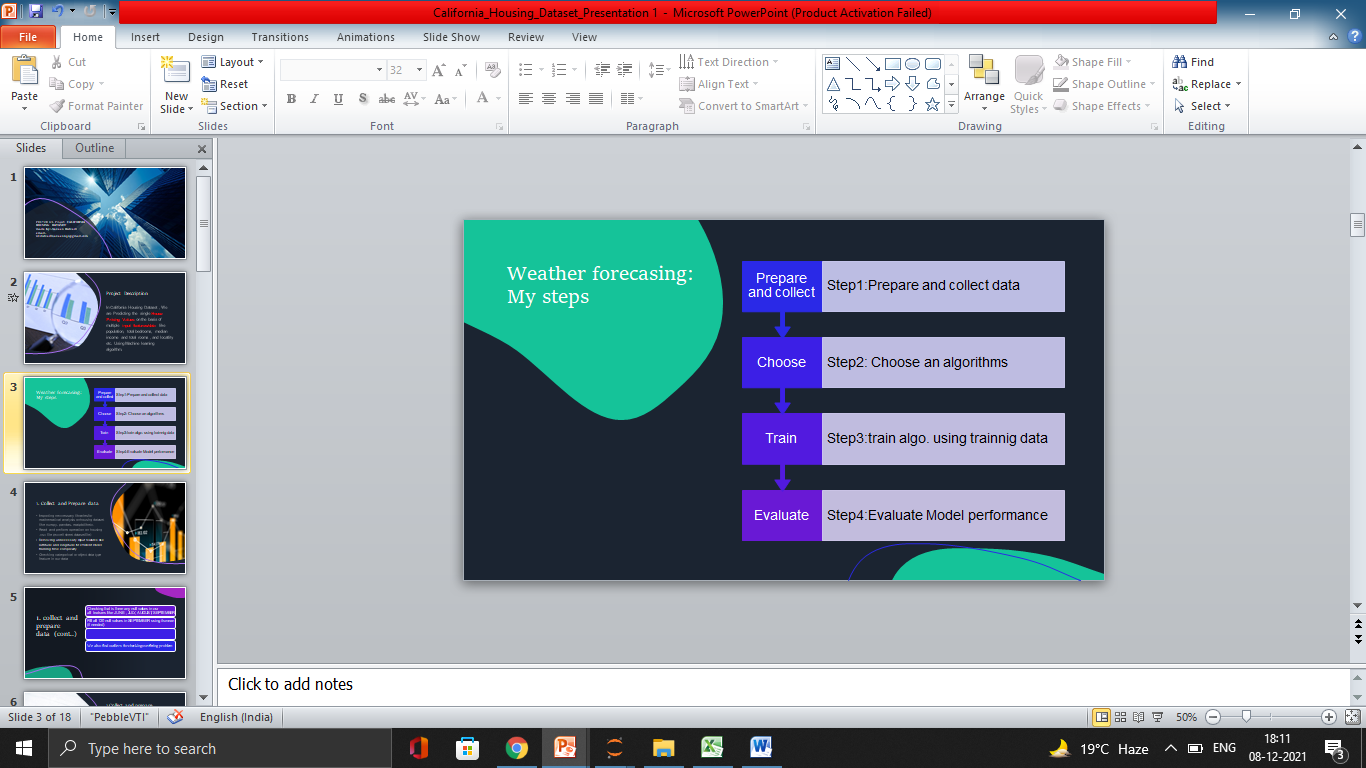
OS Window

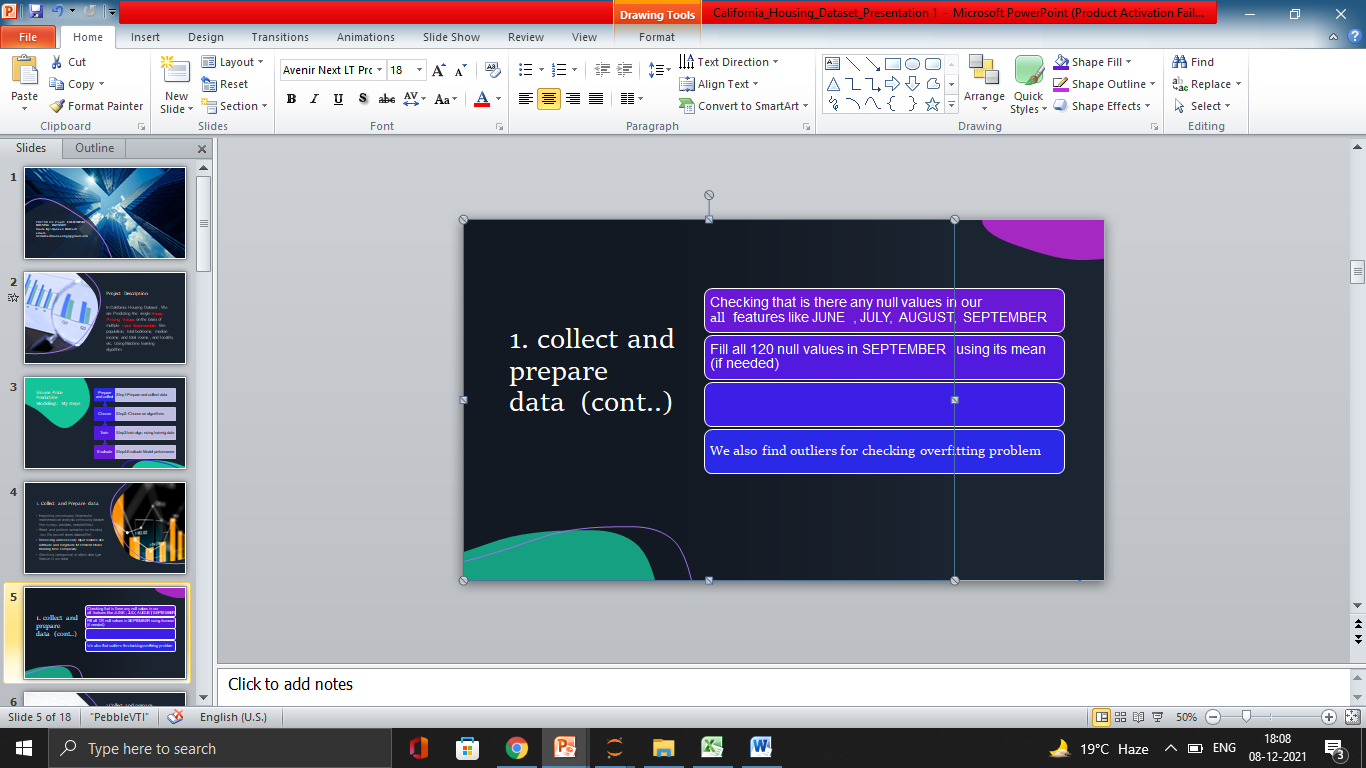
Jupyter

Notebook

Machine learning using s-klearn

**DIAGRAM:**





**CHAPTER 5**

**IMPLEMENTATION**

* 1. **Experimental Setup:**

1. #Import libraries
2. In [1]:
3. **import** pandas **as** pd
4. In [2]:
5. **import** numpy **as** np
6. In [3]:
7. **import** matplotlib.pyplot **as** plt
8. In [4]:
9. df **=** pd**.**read\_csv('weather\_data.csv')
10. In [5]:
11. df**.**head()
12. y**.**reshape(**-**1, 1)
13. **from** sklearn.pipeline **import** make\_pipeline
14. In [12]:
15. **from** sklearn.preprocessing **import** StandardScaler
16. sc\_X **=** StandardScaler()
17. sc\_y **=** StandardScaler()
18. In [13]:
19. X **=** sc\_X**.**fit\_transform(X)
20. In [14]:
21. y **=** sc\_y**.**fit\_transform(y**.**reshape(**-**1,1))
22. In [24]:
23. **from** sklearn.model\_selection **import** train\_test\_split
24. In [32]:
25. X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.22)
26. In [33]:
27. **from** sklearn.svm **import** SVR
28. In [34]:
29. regressor **=** SVR(kernel **=** 'rbf')
30. In [35]:
31. regressor**.**fit(X,y)
32. C:\Users\naveen\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples, ), for example using ravel().
33. return f(\*\*kwargs)
34. Out[35]:
35. SVR()
36. In [36]:
37. y\_pred **=** regressor**.**predict(X\_test)
38. In [37]:
39. y\_pred **=** sc\_y**.**inverse\_transform(y\_pred)
40. In [38]:
41. df **=** pd**.**DataFrame({'Real Values':sc\_y**.**inverse\_transform(y\_test**.**reshape(**-**1)), 'Predicted Values':y\_pred})
42. In [39]:
43. df

**5.2 Dataset Description:**

1. For linear regression:

Test size data 18%

Train size data 82%

2. For SVR :

Test size data 22%

Train size data 78%

**CHAPTER 6**

**RESULT AND DEMONSTRATION**

**6.1 Performance measure**

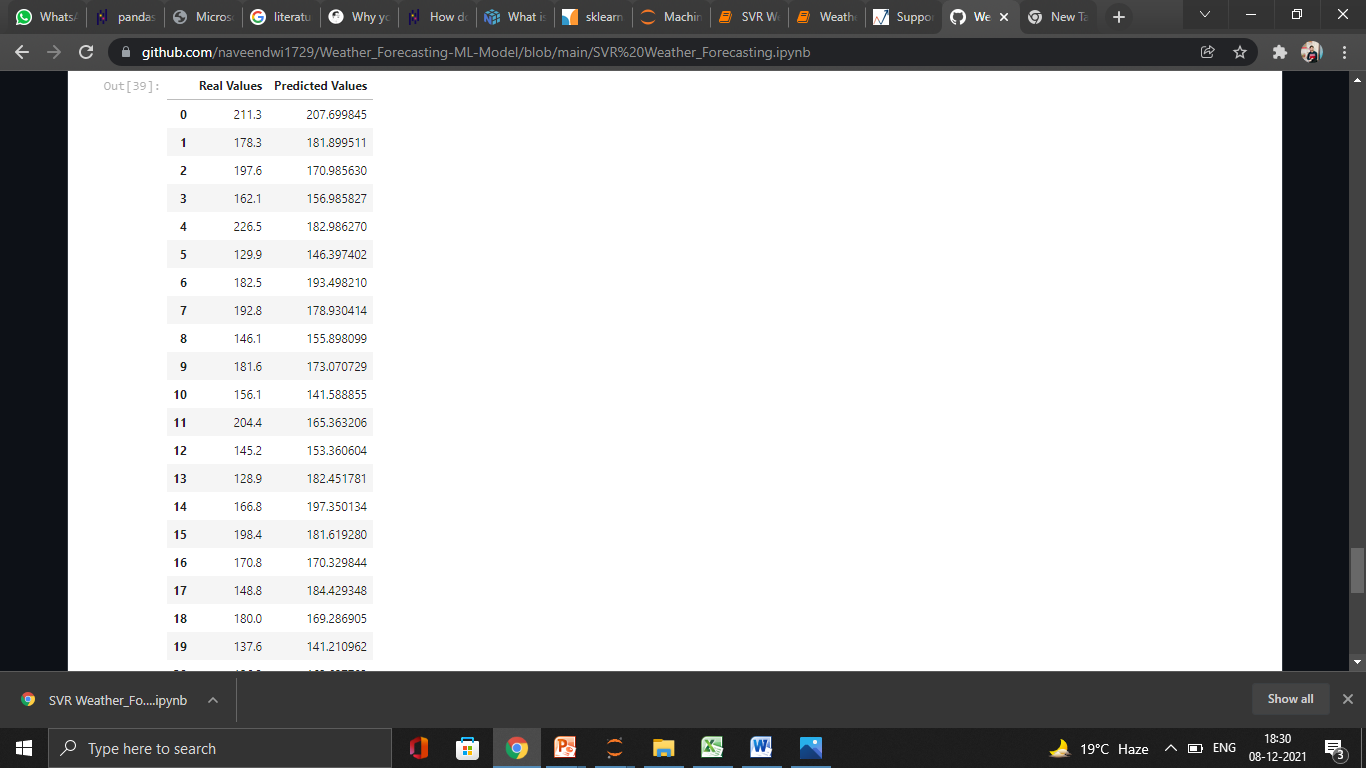
from sklearn.metrics import r2\_score

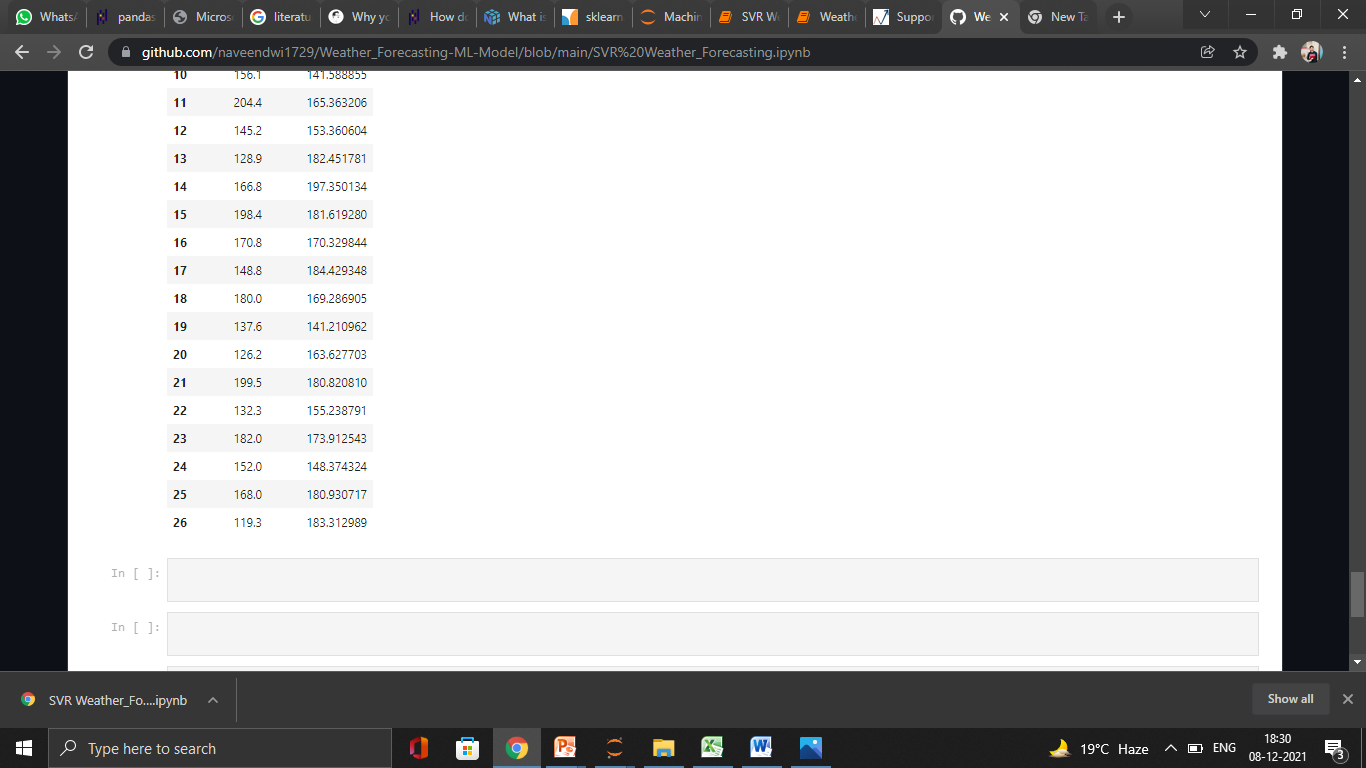
z=r2\_score(y\_test, pred)

print("Accuracy score:",z\*z\*100)

Accuracy score: 94.80055055544602

**6.2 Result Analysis**

****

****

**CHAPTER 7**

**7.1 OBJECTIVE AND RELEVENCE OF PROJECT:**

Model Predict Predicting the rainfall in mm happening in september month on the basis of previous months of monsoons like June , July , August. Model can be used :to build platform for various sector reserach and development purpose to deploy web application for end users like farmer and weather forecast department.

**7.2 TECHNICAL NOVELITY :**

We are helping Rural Farmers of India by building Machine learning Model to predict the amount of rainfall in milimeter(mm) in upcoming months of monsoon so that it all farmers to grow their crops productively and get a maximum profit.we train the model and predict accurate on recent data with ML techniques .

We can use this model to develop an application which can be further used by farmers disaster management unitand weather forecasting.it can be use by student for research purpose.

**7.3 Expected Outcomes:**

Accuracy of our model is 94% which can be used better rainfall quantity in given geography.

**CHAPTER 8**

**8.1 Conclusion:**

It can also help in resolving several major issue like disaster management , agriculture areas and weather forecasting etc.

**8.2 limitation :**

The limitation of this model be have to train our model in every area where we want to predict rainfall.

**8.3 Future scope:**

It can be use in IOT device to build better monitoring system for various sector like agriculture ,weather forecasting and other etc.

Reference :

<https://data.gov.in/resources/rainfall-all-india-and-its-departure-normal-during-monsoon-session-june-sept-1901-2019>

<https://github.com/naveendwi1729/Weather_Forecasting-ML-Model>

sklearn documentaion.