MACHINE LEARNING

(Predict the Forest Fires)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Gajjala Maansi

221710310017

Under the Guidance of

Assistant Professor

VISAKHAPATNAM HYDERABAD BENGALURU

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 **"PREDICT THE FOREST FIRES**" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science Engineering**". I declare that it was carried out independently by me under the guidance of **,**Asst. Professor, 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 Gajjala Maansi

Date: 221710310017

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **“PREDICT THE FOREST FIRES”** is being submitted by GAJJALA MAANSI (221710310017) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 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.

Dr. S. Phani Kumar

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

I would like to thank the respected faculties 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.

Gajjala Maansi

221710310017

ABSTRACT

Forest fires are a major environmental issue, creating economical and ecological damage while endangering human lives. Fast detection is a key element for controlling such phenomenon. To achieve this, one alternative is to use automatic tools based on local sensors, such as provided by meteorological stations. In effect, meteorological conditions (e.g. temperature, wind) are known to influence forest fires and several fire indexes, such as the forest Fire Weather Index (FWI), use such data.

In this work, we explore a Data Mining (DM) approach and Linear Regression to predict the burned area of forest fires. Five different DM techniques, e.g. Random Forests and Neural Networks and four distinct feature selection setups (using spatial, temporal, FWI components and weather attributes), were tested on recent real-world data collected from the northeast region of Portugal. It is capable of predicting the burned area of small fires, which are more frequent. Such knowledge is particularly useful for improving firefighting resource management (e.g. prioritizing targets for air tankers and ground crews).

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

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

1.2 IMPORTANCE OF MACHINE LEARNING:

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

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

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

The process flow depicted here represents how machine learning works

Figure 1.2 : The Process Flow

1.3 USES OF MACHINE LEARNING:

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

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

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

1.4 TYPES OF LEARNING ALGORITHMS:

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

1.4.1 Supervised Learning :

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

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

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

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

1.4.2 Unsupervised Learning:

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

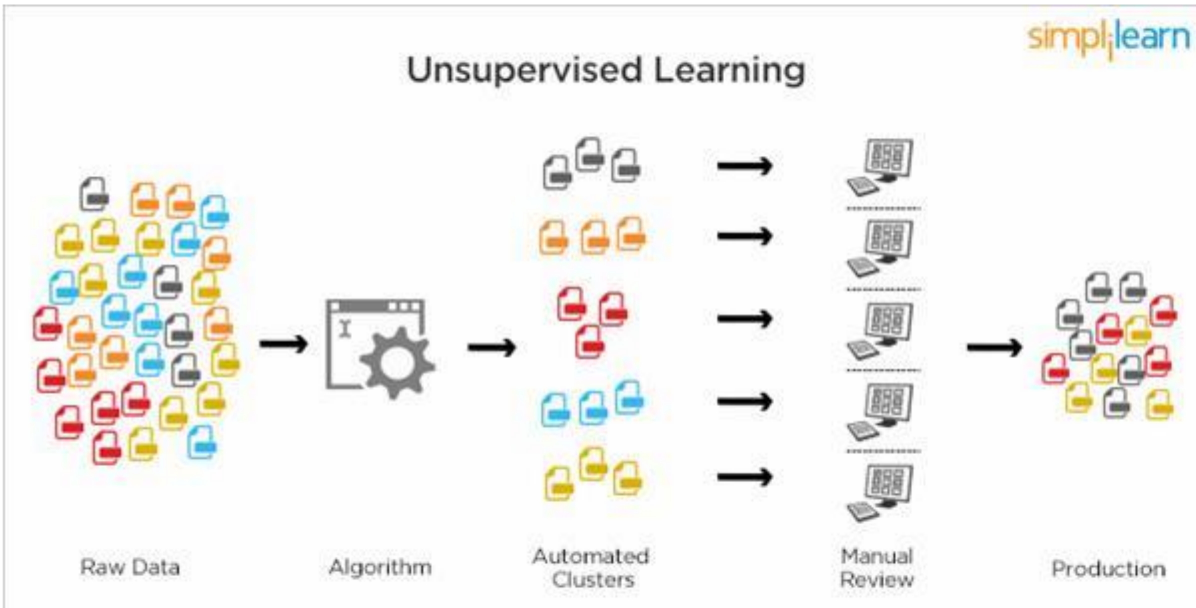


Figure 1.4.2 : Unsupervised Learning

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

1.4.3 Semi Supervised Learning:

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

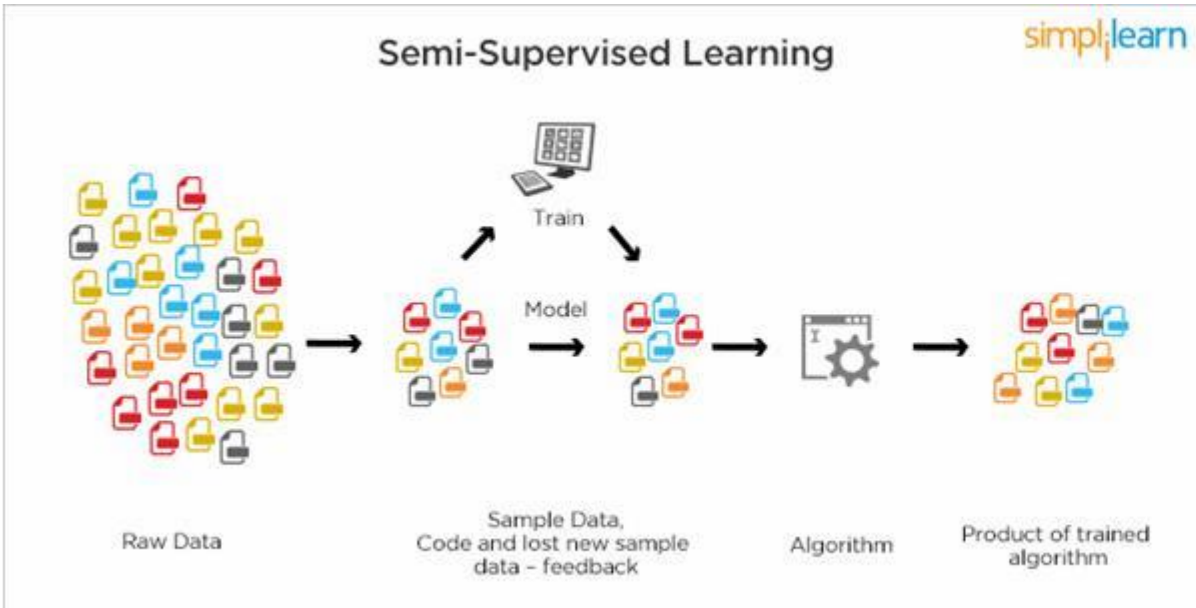


Figure 1.4.3 : Semi Supervised Learning

1.5 DEEP LEARNING

At a very basic level, deep learning is a machine learning technique. It teaches a computer to filter inputs through layers to learn how to predict and classify information. Observations can be in the form of images, text, or sound. The inspiration for deep learning is the way that the human brain filters information.

1.5.1 Introduction to Deep Learning:

Imagine you work for a loan company, and you need to build a model for predicting, whether a user (borrower) should get a loan or not? You have the features for each customer like age, bank balance, salary per annum, whether retired or not and so on.

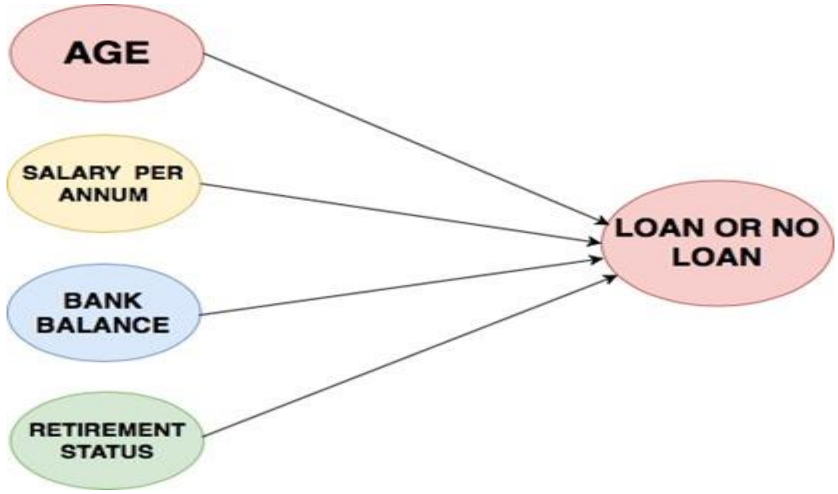


Figure 1.5.1.1 : Model for loan

Consider if you want to solve this problem using a linear regression model, then the linear regression will assume that the outcome (whether a customer's loan should be sanctioned or not) will be the sum of all the features. It will take into account the effect of age, salary, bank balance, retirement status and so. So the linear regression model is not taking into account the interaction between these features or how they affect the overall loan process.

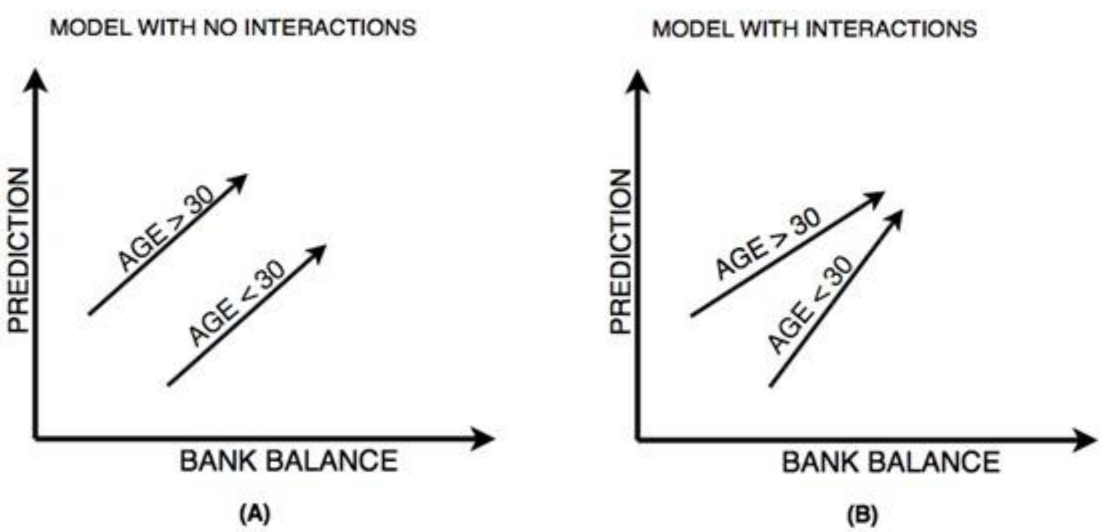


Figure 1.5.1.2: Bank balance Prediction

The above figure left (A) shows prediction from a linear regression model with absolutely no interactions in which it simply adds up the effect of age (30 > age > 30) and bank balance, you can observe from figure (A) that the lack of interaction is reflected by both lines being parallel that is what the linear regression model assumes.

On the other hand, figure right (B) shows predictions from a model that allows interactions in which the lines do not have to parallel. Neural Networks is a pretty good modeling approach that allows interactions like the one in figure (B) very well and from these neural networks evolves a term known as Deep Learning which uses these powerful neural networks. Because the neural network takes into account these type of interactions so well it can perform quite well on a plethora of prediction problems you have seen till now or possibly not heard.

Since neural networks are capable of handling such complex interactions gives them the power to solve challenging problems and do amazing things with

- Image
- Text
- Audio
- Video

This list is merely a subset of what neural networks are capable of solving, almost anything you can think of in data science field can be solved with neural networks.

1.5.2 INTERACTIONS IN NEURAL NETWORK :

The neural network architecture looks something similar to the above figure. On the far left you have the input layer that consists of the features like age, salary per annum, bank balance, etc. and on the far right, you have the output layer that outputs the prediction from the model which in your case is whether a customer should get a loan or not.

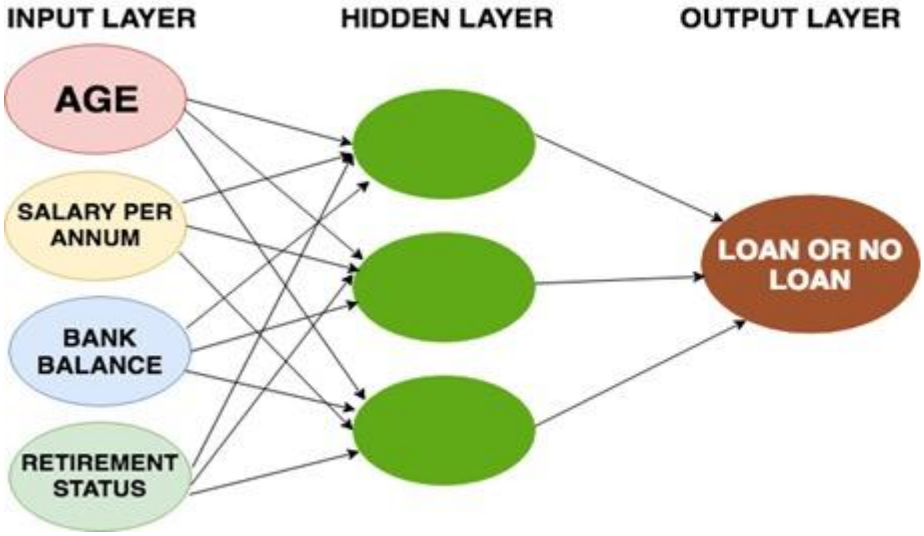


Figure 1.5.2.1: Interactions in neural network

The layers apart from the input and the output layers are called the hidden layers.

Technically, each node in the hidden layer represents an aggregation of information from the input data; hence each node adds to the model's capability to capture interactions between the data. The more the nodes, the more interactions can be achieved from the data.

1.5.3 FORWARD PROPAGATION :

To understand the concept of forward propagation let's revisit the example of a loan company. For simplification, let's consider only two features as an input namely age and retirement status, the retirement status being a binary (0 - not retired and 1 - retired) number based on which you will make predictions.

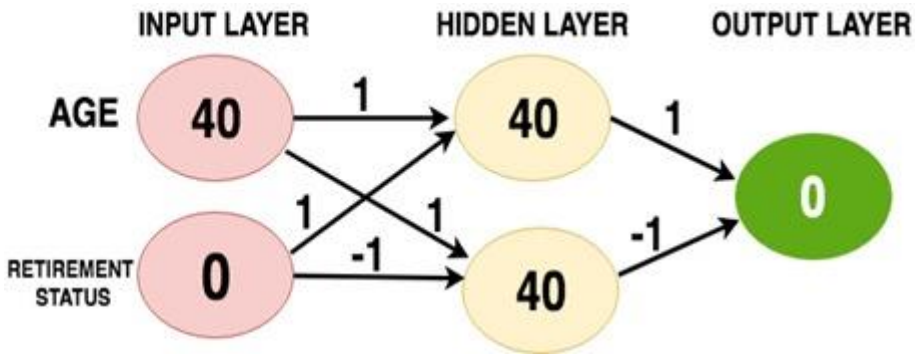


Figure 1.5.3.1: Layers

The above figure shows a customer with age 40 and is not retired. The forward propagation algorithm will pass this information through the network/model to predict the output layer. The lines connect each node of the input to every other node of the hidden layer. Each line has a weight associated with it which indicates how strongly that feature affects the hidden node connected to that specific line.

There are total four weights between input and hidden layer. The first set of weights are connected from the top node of the input layer to the first and second node of the hidden layer; likewise, the second set of weight are connected from the bottom node of the input to the first and second node of the hidden layer.

Remember these weights are the key in deep learning which you train or update when you fit a neural network to the data. These weights are commonly known as parameters.

To make a prediction for the top node of the hidden layer, you consider each node in the input layer multiply it by the weights connected to that top node and finally sum up all the values resulting in a value 40 (40 * 1 + 0 * 1 = 40) as shown in above figure. You repeat the same process for the bottom node of the hidden layer resulting in a value 40. Finally, for the output layer you follow the same process and obtain a value 0 (40 * 1 + 40 * (-1) = 0). This output layer predicts a value zero.

That's pretty much what happens in forward propagation. You start from the input layer move to the hidden layer and then to the output layer which then gives you a prediction score. You pretty much always use the multiple-add process, in linear algebra this operation is a dot product operation. In general, a forward propagation is done for a single data point at a time.

1.6 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 discovers 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.

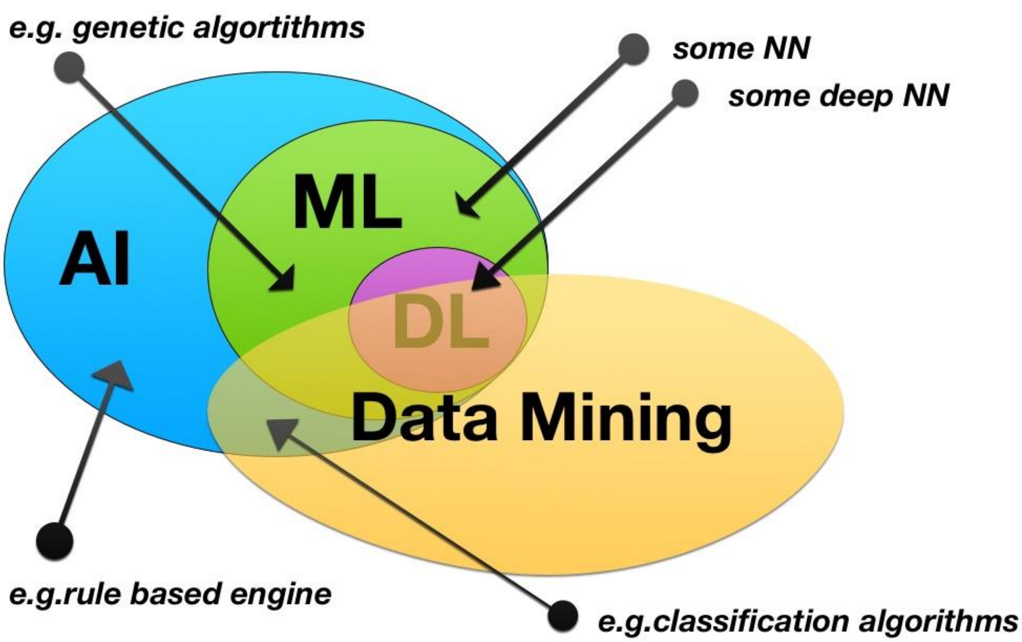


Figure 1.6 : Relation Between Data mining, Machine learning, Deep learning

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYHTON:

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

2.2 HISTORY OF PYTHON:

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

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax,

This allows the student to pick up the language quickly.

- Easy-to-read: Python code is more clearly defined and visible to the eyes.

- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.

- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

- Databases: Python provides interfaces to all major commercial databases.

- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1Installation(usingpythonIDLE)

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.

- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.

- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

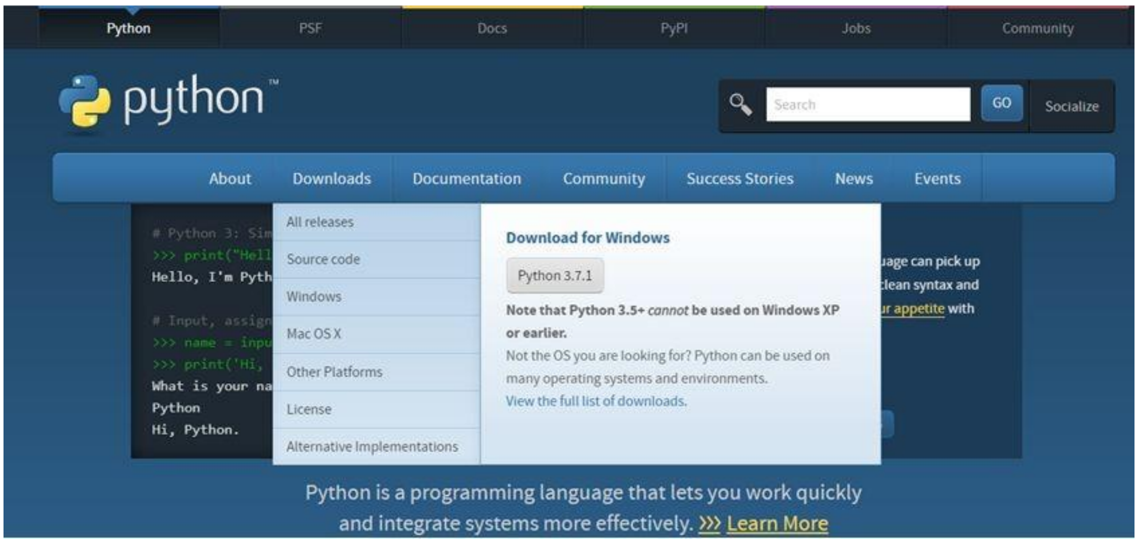


Figure 2.4.1 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.

- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.

- Conda is a package manager quickly installs and manages packages.

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

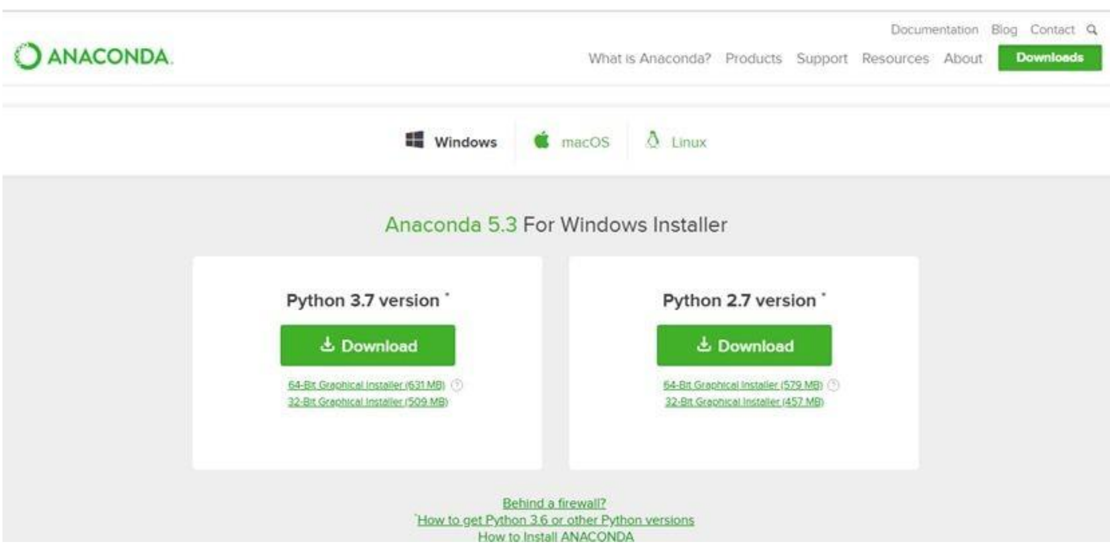


Figure 2.4.2.1 : Anaconda download

Figure 2.4.2.2 : Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

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

- Tuples
- Dictionary

2.5 1 Python Numbers:

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

2.5.2 Python Strings:

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

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

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

2.5.5 Python Dictionary:

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

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

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

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

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

2.6.2 Calling a Function:

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

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

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

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

- Defining a Class:

    - We define a class in a very similar way how we define a function.

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

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

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

Figure 2.7.1 : Defining a Class

2.7.2 __init__method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: init ().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

This is a difficult regression task, where the aim is to predict the burned area of forest fires, in the northeast region of Portugal.

3.2 DATA SET:

The given data set consists of the following parameters:

1.X - x-axis spatial coordinate within the Montesinho park map: 1 to 9

2.Y - y-axis spatial coordinate within the Montesinho park map: 2 to 9

3.month - month of the year: 'jan' to 'dec'

4.day - day of the week: 'mon' to 'sun'

5.FFMC - FFMC index from the FWI system: 18.7 to 96.20

6.DMC - DMC index from the FWI system: 1.1 to 291.3

7.DC - DC index from the FWI system: 7.9 to 860.6

8.ISI - ISI index from the FWI system: 0.0 to 56.10

9.temp - temperature in Celsius degrees: 2.2 to 33.30

10.RH - relative humidity in %: 15.0 to 100

11.wind - wind speed in km/h: 0.40 to 9.40

12.rain - outside rain in mm/m2 : 0.0 to 6.4

13.area - the burned area of the forest (in ha): 0.00 to 1090.84

3.3 OBJECTIVE OF THE CASE STUDY:

This is a regression problem with clear outliers which cannot be predicted using any reasonable method. A comparision of the three methods has been done :

(a) Random Forest Regressor,

(b) Neural Network,

(c) Linear Regression

The output 'area' was first transformed with a ln(x+1) function.
One regression metric was measured: RMSE and r2 score is obtained. An analysis to the regression error curve(REC) shows that the RFR model predicts more examples within a lower admitted error. In effect, the RFR model predicts better small fires and r2 score is obtained by using Linear Regression.

CHAPTER 4

MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

we can get the data from client. we can get the data from database.

https://archive.ics.uci.edu/ml/datasets/forest+fires

4.1.2 IMPORTING THE LIBRARIES:

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

```
[1]  import numpy as np      #Importing numpy package
     import pandas as pd     #Importing pandas package
     import seaborn as sns   #Importing seaborn package
     import sklearn          #Importing sklearn package
     import matplotlib.pyplot as plt   #Importing matplotlib package
     %matplotlib inline
```

Figure 4.1.2 : Importing Libraries

4.1.3 IMPORTING THE DATA-SET:

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

```
[2] data = pd.read_csv("/content/drive/My Drive/Summer Internship Project /Predict the ForestFires/forestfires.csv")     #Reading the dataset from drive
    data    #Displaying the dataframe
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.00
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...
512	4	3	aug	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44
513	2	4	aug	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29
514	7	4	aug	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16
515	1	4	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00
516	6	3	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00

Figure 4.1.3: Reading the dataset

4.1.4 HANDLING MISSING VALUES:

Missing values

```
[ ]  data.isna().sum()     ### checking missing values
```

```
X          0
Y          0
month      0
day        0
FFMC       0
DMC        0
DC         0
ISI        0
temp       0
RH         0
wind       0
rain       0
area       0
dtype: int64
```

Figure 4.1.4: Missing values

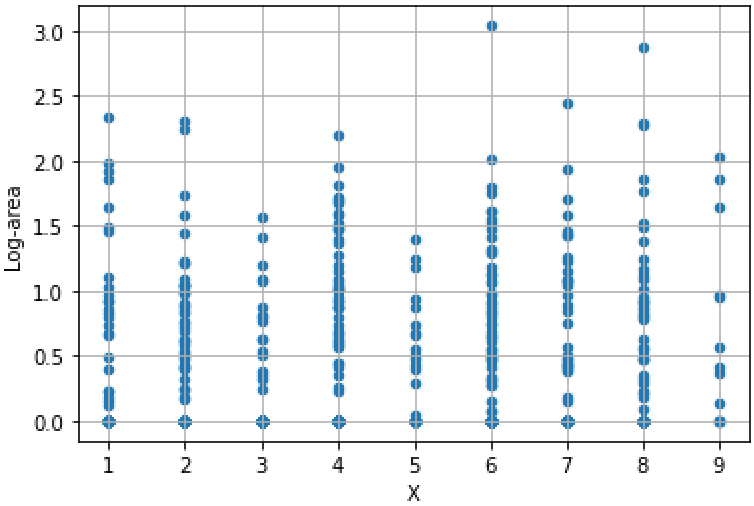
OBSERVATION:

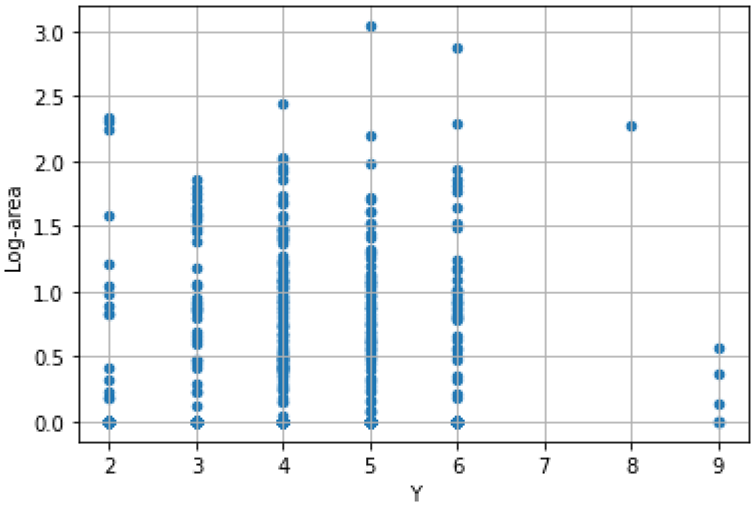
As we can see there are no missing values in the given dataset of forest fires

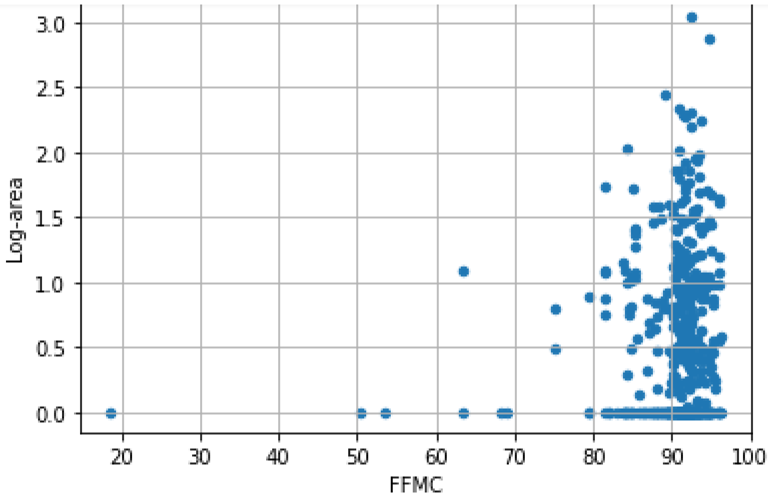
4.1.5 DATA VISUALIZATION:

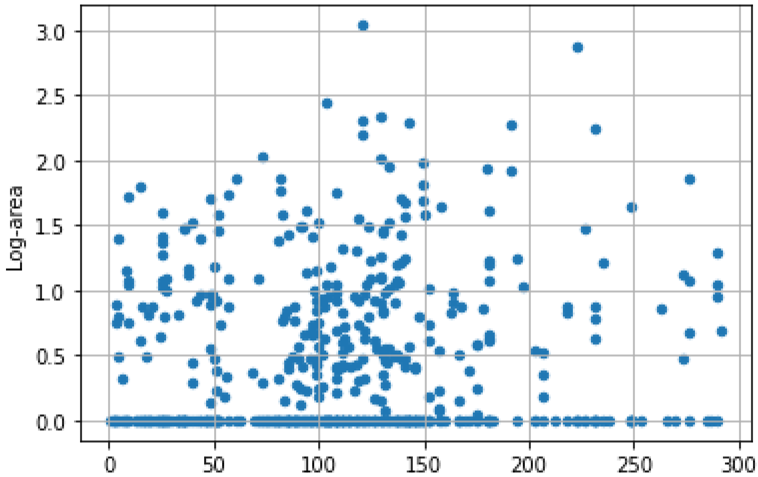
```
[19] data['Log-area']=np.log10(data['area']+1)     #To find the log of the column area
```

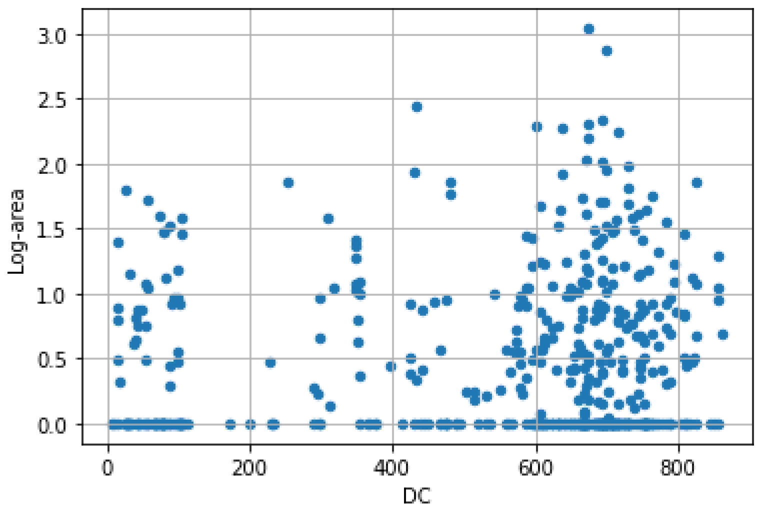
```
[20] for i in data.describe().columns[:-2]:
         data.plot.scatter(i,'Log-area',grid=True)     #Display the scatter plot with x-axis as columns and y-axis as log area column
```

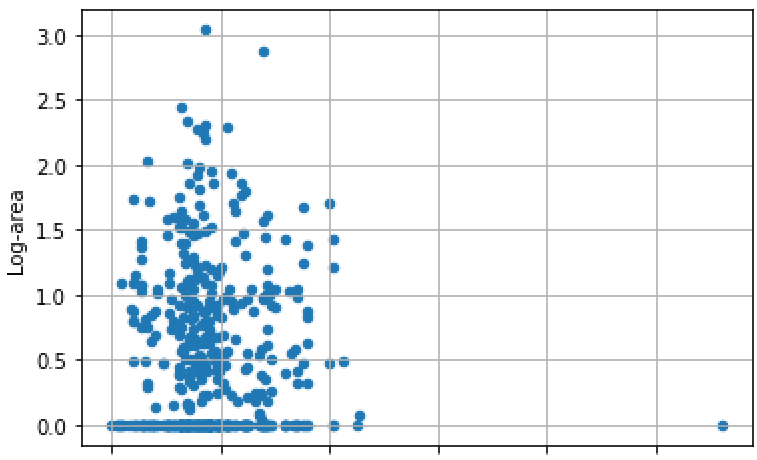


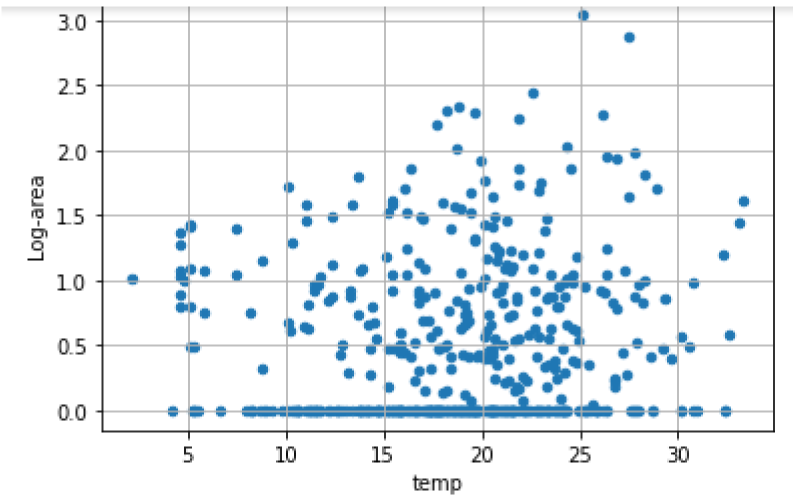



Log-area
FFMC

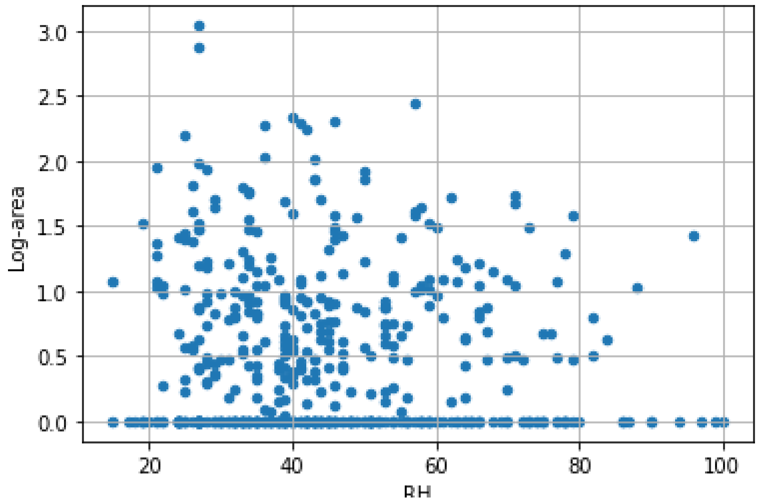

Log-area


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Log-area
0
200
400
600
800
DC


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Log-area



Log-area
temp
0.0
0.5
1.0
1.5
2.0
2.5
3.0
5
10
15
20
25
30



Log-area
RH
0.0
0.5
1.0
1.5
2.0
2.5
3.0
20
40
60
80
100

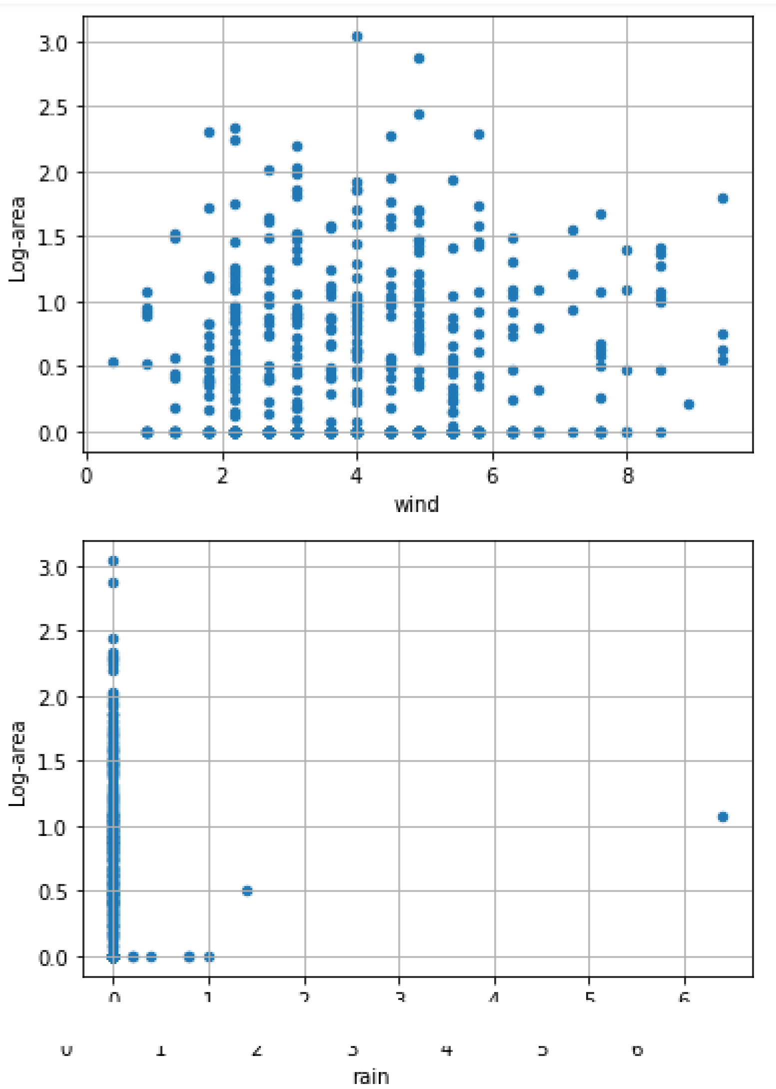


Figure 4.1.5.1: scatterplots and distributions of numerical features to see how they may affect the output 'area'

```
[21] data.boxplot(column='Log-area',by='day')      #Boxplot which shows the how categorical column "day" affect the outcome
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb4a8e17080>
```

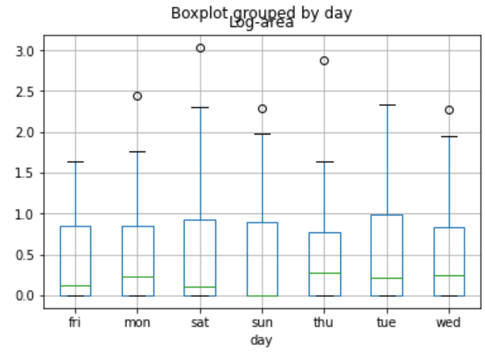


Figure 4.1.5.2: Boxplot of how categorical column day affect the outcome

```
[22] data.boxplot(column='Log-area',by='month')      #Boxplot which shows the how categorical column "month" affect the outcome
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb4a8d8c128>
```

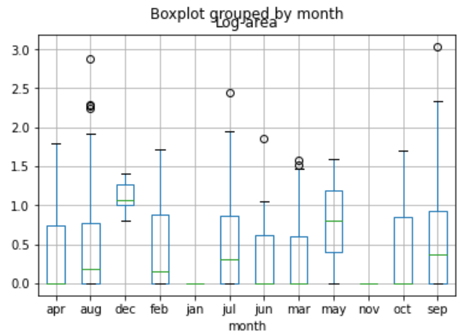


Figure 4.1.5.3:Boxplot of how categorical column month affect the outcome

4.1.6 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 pandas 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 concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

```
[24] data['month']    #Encoding the categorical column 'month'

0      mar
1      oct
2      oct
3      mar
4      mar
      ...
512    aug
513    aug
514    aug
515    aug
516    nov
Name: month, Length: 517, dtype: object
```

Figure 4.1.6.1: Categorical data-column 'month'

```
[25] ## Coverting the categorical data into numerical columns using get_dummies
     dummy_set = pd.get_dummies(data.month)
     dummy_set    #Display the dummy_set
```

	apr	aug	dec	feb	jan	jul	jun	mar	may	nov	oct	sep
0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
512	0	1	0	0	0	0	0	0	0	0	0	0
513	0	1	0	0	0	0	0	0	0	0	0	0
514	0	1	0	0	0	0	0	0	0	0	0	0
515	0	1	0	0	0	0	0	0	0	0	0	0
516	0	0	0	0	0	0	0	0	0	1	0	0

517 rows × 12 columns

Figure 4.1.6.2: dummy set for column 'month'

```
[26] data['day']    #Encoding the categorical column 'day'
```

```
0      fri
1      tue
2      sat
3      fri
4      sun
      ...
512    sun
513    sun
514    sun
515    sat
516    tue
Name: day, Length: 517, dtype: object
```

Figure 4.1.6.3: Categorical column-'day'

```
[27] dummy_set1 = pd.get_dummies(data.day)    ## Coverting the categorical data into numerical columns using get_dummies
     dummy_set1 #Display the dummy_set
```

	fri	mon	sat	sun	thu	tue	wed
0	1	0	0	0	0	0	0
1	0	0	0	0	0	1	0
2	0	0	1	0	0	0	0
3	1	0	0	0	0	0	0
4	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...
512	0	0	0	1	0	0	0
513	0	0	0	1	0	0	0
514	0	0	0	1	0	0	0
515	0	0	1	0	0	0	0
516	0	0	0	0	0	1	0

517 rows × 7 columns

Figure 4.1.6.4: dummy set for column 'day'

```
[28] # Concatenating the original dataframe and dataframe of dummy columns
     merged_data = pd.concat([data, dummy_set, dummy_set1 ], axis=1)
     merged_data    #Displaying the merged data
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	Log-area	apr	aug	dec	feb	jan	jul	jun	mar	may	nov	oct	sep	fri	mon	sat
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00	0.000000	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00	0.000000	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00	0.000000	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.00	0.000000	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.00	0.000000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
512	4	3	aug	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44	0.871573	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
513	2	4	aug	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29	1.742647	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
514	7	4	aug	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16	1.084934	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
515	1	4	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00	0.000000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
516	6	3	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00	0.000000	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

517 rows × 33 columns

Figure 4.1.6.5: Concatenating dummy sets to dataframe

- Getting dummies using label encoder from scikit learn package
- We have a method called label encoder in scikit learn package .we need to import the label encoder method from scikitlearn package and after that we have to fit and transform the data frame to make the categorical data into dummies.
- If we use this method to get dummies then in place of categorical data we get the numerical values (0,1,2….)

```
[30] from sklearn.model_selection import train_test_split     #Importing train_test_split from sklearn model selection
     from sklearn.preprocessing import OneHotEncoder           #Importing OneHotEncoder from sklearn preprocessing
     from sklearn.preprocessing import LabelEncoder            #Importing LabelEncoder from sklearn preprocessing
```

Figure 4.1.6.6: importing label encoder and one hot encoder

```
enc.classes_     #Encoding classes of month column

array(['apr', 'aug', 'dec', 'feb', 'jan', 'jul', 'jun', 'mar', 'may',
       'nov', 'oct', 'sep'], dtype=object)
```

```
data['month_encoded']=enc.transform(data['month'])     #Transforming the encoded month column
data.head()
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	Log-area	month_encoded
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0	0.0	7
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0	0.0	10
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0	0.0	10
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0	0.0	7
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0	0.0	7

Figure 4.1.6.7: Handling categorical data of column month

```
enc.classes_     #Encoding classes of day column

array(['fri', 'mon', 'sat', 'sun', 'thu', 'tue', 'wed'], dtype=object)
```

```
data['day_encoded']=enc.transform(data['day'])      #Transforming the encoded day column
data.head(10)
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	Log-area	month_encoded	day_encoded
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0	0.0	7	0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0	0.0	10	5
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0	0.0	10	2
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0	0.0	7	0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0	0.0	7	3
5	8	6	aug	sun	92.3	85.3	488.0	14.7	22.2	29	5.4	0.0	0.0	0.0	1	3
6	8	6	aug	mon	92.3	88.9	495.6	8.5	24.1	27	3.1	0.0	0.0	0.0	1	1
7	8	6	aug	mon	91.5	145.4	608.2	10.7	8.0	86	2.2	0.0	0.0	0.0	1	1

Figure 4.1.6.8: Handling categorical data of column day

4.2 TRAINING THE MODEL:

- Splitting the data : after the preprocessing is done then the data is split into train and test sets

- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.

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

- test set - a subset to test the trained model.(To test whether the model has correctly learnt)

- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75%

  , test data =25% or train data = 80% , test data= 20%)
- First we need to identify the input and output variables and we need to separate the input set and output set

- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method

- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables)

```
[40] # Preparing Training and Testing Data
     from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                          random_state= 156)
```

```
[41] print(X_train.shape)    #To display the X_train shape
     print(X_test.shape)     #To display the X_test shape
     print(y_train.shape)    #To display the y_train shape
     print(y_test.shape)     #To display the y_test shape
     X_train      #Displaying the X_train
```

```
(310, 12)
(207, 12)
(310,)
(207,)
```

	X	Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	month_encoded	day_encoded
293	7	6	93.1	180.4	430.8	11.0	26.9	28	5.4	0.0	5	5
390	7	4	84.7	9.5	58.3	4.1	7.5	71	6.3	0.0	3	1
237	1	2	91.0	129.5	692.6	7.0	18.8	40	2.2	0.0	11	5
337	6	3	91.6	108.4	764.0	6.2	23.0	34	2.2	0.0	11	1
453	4	5	89.4	266.2	803.3	5.6	17.4	54	3.1	0.0	1	4

Figure 4.2: Importing train_test_split

4.3 EVALUATING THE CASE STUDY:

4.3.1 Building the model (using splitting):

- First we have to retrieve the input and output sets from the given dataset

```
X = data.drop(['area','Log-area','month','day'], axis=1)   #Splitting the dataset to X by dropping the columns area,logarea,month,da
X.head()
```

	X	Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	month_encoded	day_encoded
0	7	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	7	0
1	7	4	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	10	5
2	7	4	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	10	2
3	8	6	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	7	0
4	8	6	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	7	3

Figure 4.3.1.1: Retrieving the input columns

```
y = data['Log-area']     #Splitting the dataset to y by dropping the columns logarea
y.head()
```

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Log-area, dtype: float64
```

Figure 4.3.1.2: Retrieving output column

4.3.2 MODEL BUILDING:

```
def rec(m,n,tol):      #Defining rec
    if type(m)!='numpy.ndarray':
        m=np.array(m)
    if type(n)!='numpy.ndarray':
        n=np.array(n)
    l=m.size       #Assigning m.size to l
    percent = 0
    for i in range(l):
        if np.abs(10**m[i]-10**n[i])<=tol:
            percent+=1
    return 100*(percent/l)   #Returning (100*percent/l) value
```

```
tol_max=20    # Assigning Maxvalue
```

Figure 4.3.2: Defining Regression Error Characteristic (REC)

4.3.2.1 Regression Error Characteristic (REC) estimation:

Receiver Operating Characteristic (ROC) curves provide a powerful tool for visualizing and comparing classification results. Regression Error Characteristic (REC) curves generalize ROC curves to regression. REC curves plot the error tolerance on the versus the percentage of points predicted within the tolerance on the . The resulting curve estimates the cumulative distribution function of the error. The REC curve visually presents commonly-useds tatistics. The area-over-the-curve (AOC) is a biased estimate of the expected error.

The value can be estimated using the ratio of the AOC for a given model to the AOC for the nul-model. Users can quickly assess the relative merits of many regression functions by examining the relative position of their REC curves. The shape of the curve reveals additional information that can be used to guide modeling.

4.4 RANDOM FOREST REGRESSOR:

Random forest algorithm can be used for both classifications and regression task. It provides higher accuracy. Random forest classifier will handle the missing values and maintain the accuracy of a large proportion of data. If there are more trees, it won't allow overfitting trees in the model.

Random Forest is a flexible, easy to use machine learning algorithm that produces great results most of the time with minimum time spent on hyper-parameter tuning. It has gained popularity due to its simplicity and the fact that it can be used for both classification and regression tasks.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

RANDOM FOREST REGRESSOR

```
[ ] from sklearn.ensemble import RandomForestRegressor      #Importing Random Forest Regressor package from sklearn ensemble
    from sklearn.model_selection import GridSearchCV        #Importing GridSearchCV package from sklearn.model_selection
```

Figure 4.4.1: Importing random forest regressor and gridsearchCV

Gridsearch

Finding the right parameters for machine learning models is a tricky task. But luckily, Scikit-learn has the functionality of trying a bunch of combinations and see what works best, built in with GridSearchCV.The CV stands for cross-validation.

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

```
# Parameter grid for the Grid Search
param_grid = {'C': [0.01,0.1,1, 10], 'epsilon': [10,1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
```

```
param_grid = {'max_depth': [5,10,15,20,50], 'max_leaf_nodes': [2,5,10], 'min_samples_leaf': [2,5,10],
             'min_samples_split':[2,5,10]}
grid_RF = GridSearchCV(RandomForestRegressor(),param_grid,refit=True,verbose=0,cv=5)
grid_RF.fit(X_train,y_train)     #Fitting the X_train and y_train in the model
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [5, 10, 15, 20, 50],
                         'max_leaf_nodes': [2, 5, 10],
                         'min_samples_leaf': [2, 5, 10],
                         'min_samples_split': [2, 5, 10]},
```

Figure 4.4.2: parameter grid for grid search and rfr

```
print("Best parameters obtained by Grid Search:",grid_RF.best_params_)      #Displaying the best parameters obtained by Grid Search
```

```
Best parameters obtained by Grid Search: {'max_depth': 10, 'max_leaf_nodes': 2, 'min_samples_leaf': 5, 'min_samples_split': 10}
```

Figure 4.4.3: best parameters obtained by grid search

```
a=grid_RF.predict(X_test)     #Predicting the X_test by grid rf
rmse_rf=np.sqrt(np.mean((y_test-a)**2))     #RMSE formula
print("RMSE for Random Forest:",rmse_rf)    #Printing the Rmse value for random forest
```

```
RMSE for Random Forest: 0.6285079753296434
```

Figure 4.4.4: RMSE for random forest

```
##Scatter plot to show actual area burned and error
plt.xlabel("Actual area burned")      # To print xlabel as actual area burned
plt.ylabel("Error")          # To print ylabel as error
plt.grid(True)      #plotting as a grid
plt.scatter(10**(y_test),10**(a)-10**(y_test))     ##plotting a Scatterplot
```

```
<matplotlib.collections.PathCollection at 0x7fb4a90df048>
```

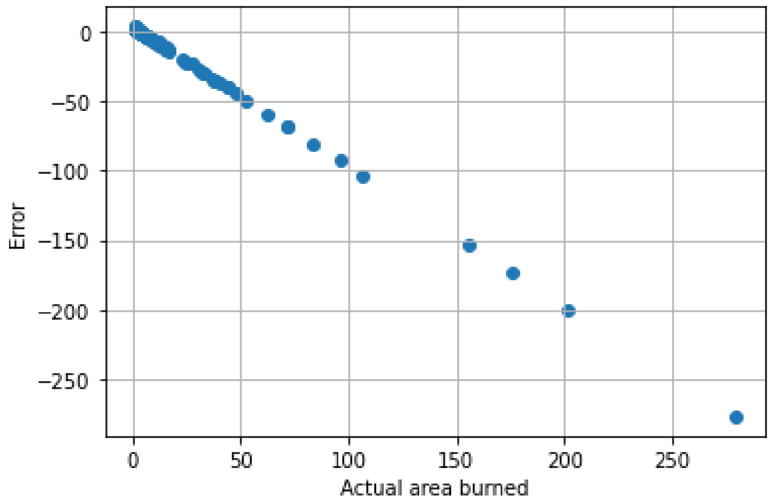


Figure 4.4.5: scatter plot
x-axis: actual area burned
y-axis: error

```
plt.title("Histogram of prediction errors\n",fontsize=18)     ##Title to the plot
plt.xlabel("Prediction error ($ha$)",fontsize=14)    ## xlabel as prediction error
plt.grid(True)    ##plotting as a grid
plt.hist(10**(a.reshape(a.size,))-10**(y_test),bins=50)   ##plotting a histogram
```

```
(array([  1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   1.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,
          1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   1.,
          0.,   1.,   0.,   0.,   2.,   1.,   0.,   1.,   1.,   5.,   3.,
          3.,   4.,   2.,  16.,  26., 136.]),
 array([-276.60848446, -271.00192186, -265.39535926, -259.78879667,
        -254.18223407, -248.57567148, -242.96910888, -237.36254629,
        -231.75598369, -226.14942109, -220.5428585 , -214.9362959 ,
        -209.32973331, -203.72317071, -198.11660812, -192.51004552,
        -186.90348293, -181.29692033, -175.69035773, -170.08379514,
        -164.47723254, -158.87066995, -153.26410735, -147.65754476,
        -142.05098216, -136.44441956, -130.83785697, -125.23129437,
        -119.62473178, -114.01816918, -108.41160659, -102.80504399,
         -97.19848139,  -91.5919188 ,  -85.9853562 ,  -80.37879361,
         -74.77223101,  -69.16566842,  -63.55910582,  -57.95254322,
         -52.34598063,  -46.73941803,  -41.13285544,  -35.52629284,
         -29.91973025,  -24.31316765,  -18.70660505,  -13.10004246,
          -7.49347986,   -1.88691727,    3.71964533]),
 <a list of 50 Patch objects>)
```

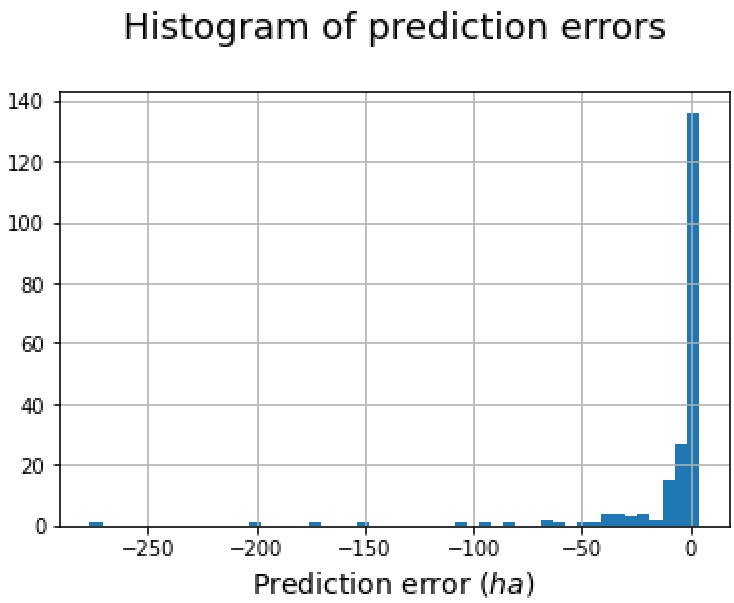


Figure 4.4.6: Histogram of prediction errors of rfr

```python
rec_RF=[]
for i in range(tol_max):
    rec_RF.append(rec(a,y_test,i))

plt.figure(figsize=(5,5))    #Assessing a size to the plot
plt.title("REC curve for the Random Forest\n",fontsize=15)   #title of the plot
plt.xlabel("Absolute error (tolerance) in prediction ($ha$)")   #assessing x label as absolute error
plt.ylabel("Percentage of correct prediction")   # assessing ylabel as percentage of correct prediction
plt.xticks([i for i in range(0,tol_max+1,5)])
plt.ylim(-10,100)    # giving a y limit as -10 to 100
plt.yticks([i*20 for i in range(6)])
plt.grid(True)    #plotting as a grid
plt.plot(range(tol_max),rec_RF)
```

```
[<matplotlib.lines.Line2D at 0x7f5a3bfc5ac8>]
```

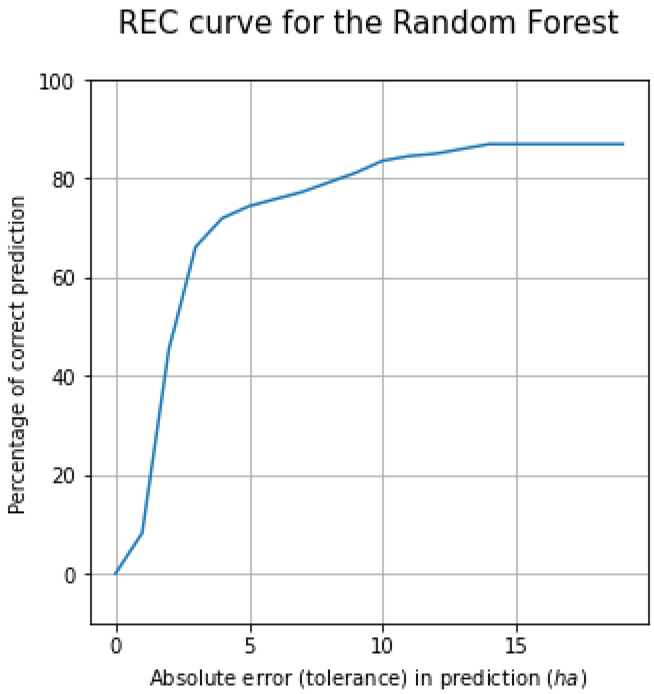


Figure 4.4.7: REC curve for rfr

4.5 NEURAL NETWORK:

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

KEY TAKEAWAYS

- Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data.
- They are used in a variety of applications in financial services, from forecasting and marketing research to fraud detection and risk assessment.
- Use of neural networks for stock market price prediction varies.

Application of Neural Networks

Neural networks are broadly used, with applications for financial operations, enterprise planning, trading, business analytics and product maintenance. Neural networks have also gained widespread adoption in business applications such as forecasting and marketing research solutions, fraud detection and risk assessment.

NEURAL NETWORK

```
[ ] from keras.models import Sequential       #Importing sequential from keras.model
    import keras.optimizers as opti           #Importing opti from keras.optimizers
    from keras.layers import Dense, Activation,Dropout    #Importing Dense,Activation,Dropout from keras.layers
```

Figure 4.5.1: Importing NN packages

```
data=X_train    #assigning data as X_train
target = y_train    #assigning target as y_train
model.fit(data, target, epochs=100, batch_size=10,verbose=0)    #Fitting the model with 100 epochs
```

Figure 4.5.2: dividing data and target

Prediction and RMSE

```
[ ] a=model.predict(X_test)    #predicting the model with X_test
    print("RMSE for NN:",np.sqrt(np.mean((y_test-a.reshape(a.size,))**2)))    #Printing the rmse for nn model

RMSE for NN: 0.6337681878392079
```

Figure 4.5.3: RMSE for NN

The RMSE of rfr is 0.628

The RMSE of NN is 0.633

Random Forest Regressor algorithm in which GridsearchCV is used is the best one to predict.

```
plt.xlabel("Actual area burned")   # To print xlabel as actual area burned
plt.ylabel("Error")    # To print ylabel as error
plt.grid(True)     #plotting as a grid
plt.scatter(10**(y_test),10**(a.reshape(a.size,))-10**(y_test))   ##Plotting a scatterplot
```

```
<matplotlib.collections.PathCollection at 0x7fb45f115080>
```

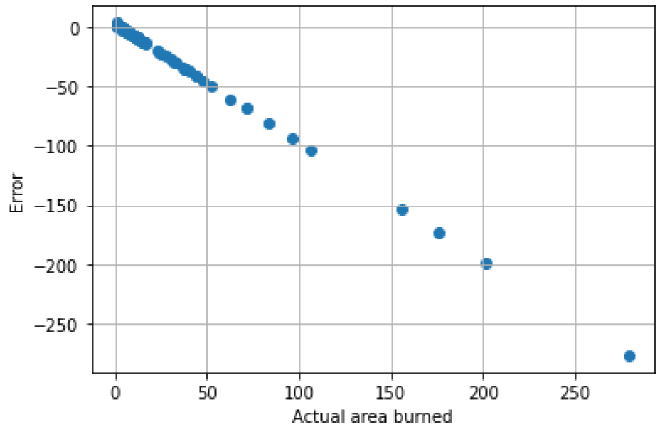


Figure 4.5.4: scatterplot for NN
x-axis: actual area burned
y-axis: error

```
plt.title("Histogram of prediction errors\n",fontsize=16)   ##Title to the plot
plt.xlabel("Prediction error ($ha$)",fontsize=14)   ##xlabel as prediction error
plt.grid(True)   #plotting as a grid
plt.hist(10**(a.reshape(a.size,))-10**(y_test),bins=50)  #plotting a histogram
```

```
(array([  1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   1.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,
          1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   1.,
          0.,   1.,   0.,   0.,   2.,   1.,   0.,   1.,   2.,   3.,   4.,
          4.,   3.,   2.,  18.,  26., 134.]),
 array([-276.85950282, -271.24897555, -265.63844828, -260.02792102,
        -254.41739375, -248.80686648, -243.19633921, -237.58581194,
        -231.97528468, -226.36475741, -220.75423014, -215.14370287,
        -209.5331756 , -203.92264834, -198.31212107, -192.7015938 ,
        -187.09106653, -181.48053926, -175.870012  , -170.25948473,
        -164.64895746, -159.03843019, -153.42790292, -147.81737566,
        -142.20684839, -136.59632112, -130.98579385, -125.37526658,
        -119.76473932, -114.15421205, -108.54368478, -102.93315751,
         -97.32263024,  -91.71210298,  -86.10157571,  -80.49104844,
         -74.88052117,  -69.2699939 ,  -63.65946664,  -58.04893937,
         -52.4384121 ,  -46.82788483,  -41.21735756,  -35.6068303 ,
         -29.99630303,  -24.38577576,  -18.77524849,  -13.16472122,
          -7.55419396,   -1.94366669,    3.66686058]),
 <a list of 50 Patch objects>)
```

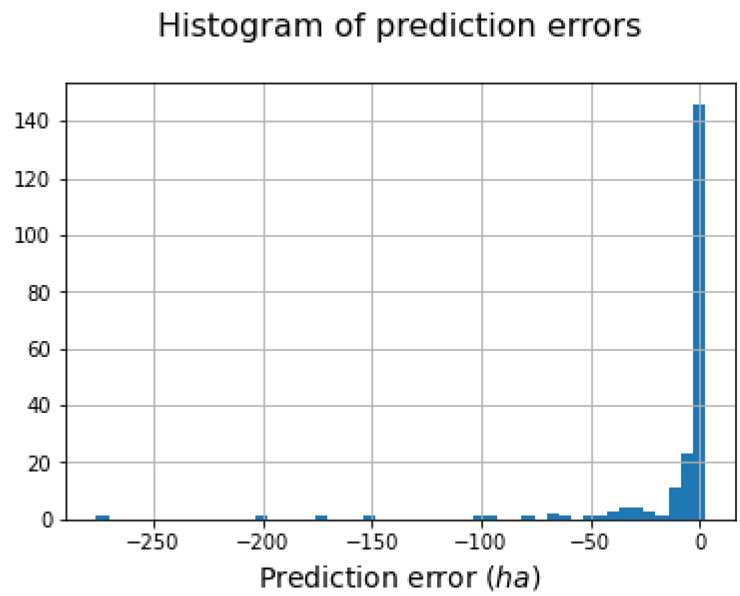


Figure 4.5.5: Histogram of prediction error of NN

```
rec_NN=[]   #defining rec
for i in range(tol_max):
    rec_NN.append(rec(a,y_test,i))
plt.figure(figsize=(5,5))  #assigning the size to the figure
plt.title("REC curve for Neural Network\n",fontsize=16)   ##title to the rec curve with font size 16
plt.xlabel("Absolute error (tolerance) in prediction ($ha$)")   #plotting with x label as absolute error
plt.ylabel("Percentage of correct prediction")   #plotting with y label as percentahe of correct prediction
plt.xticks([i for i in range(0,tol_max+1,5)])
plt.ylim(-10,100)   #assigning a limit for y
plt.yticks([i*20 for i in range(6)])
plt.grid(True)   #plotting a grid
plt.plot(range(tol_max),rec_NN)
```

```
[<matplotlib.lines.Line2D at 0x7f59f1dbeda0>]
```

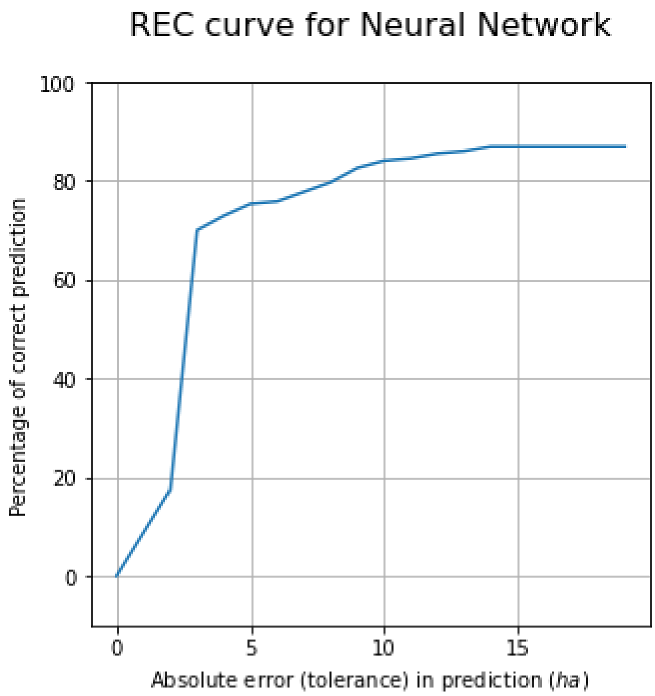


Figure 4.5.6: REC curve for NN

4.6 Relative Performance of Random Forest Regressor and Neural Networks

Relative performance of Random Forest Regressor and Neural Networks(REC Curves)

```
[ ] ### Relative performance of random forest and nn
    plt.figure(figsize=(10,8))    #plotting figure with size 10,8
    plt.title("REC curve for RFR and NN\n",fontsize=20)  #title to the rec curve for rfr and nn
    plt.xlabel("Absolute error (tolerance) in prediction ($ha$)",fontsize=15)   #Plotting x label as absolute error(tolerance)in prediction
    plt.ylabel("Percentage of correct prediction",fontsize=15)   #Plotting y label as percentage of correct prediction
    plt.xticks([i for i in range(0,tol_max+1,1)],fontsize=13)
    plt.ylim(-10,100)   #assigning a y limit
    plt.xlim(-2,tol_max)   #assigning a x limit
    plt.yticks([i*20 for i in range(6)],fontsize=18)
    plt.grid(True)  #plotting as a grid
    plt.plot(range(tol_max),rec_RF,'--',lw=3)
    plt.plot(range(tol_max),rec_NN,'-',lw=3)
    plt.legend(['Random Forest','NN'],fontsize=13)   #Plotting rfr and nn

<matplotlib.legend.Legend at 0x7fb45eef65c0>
```

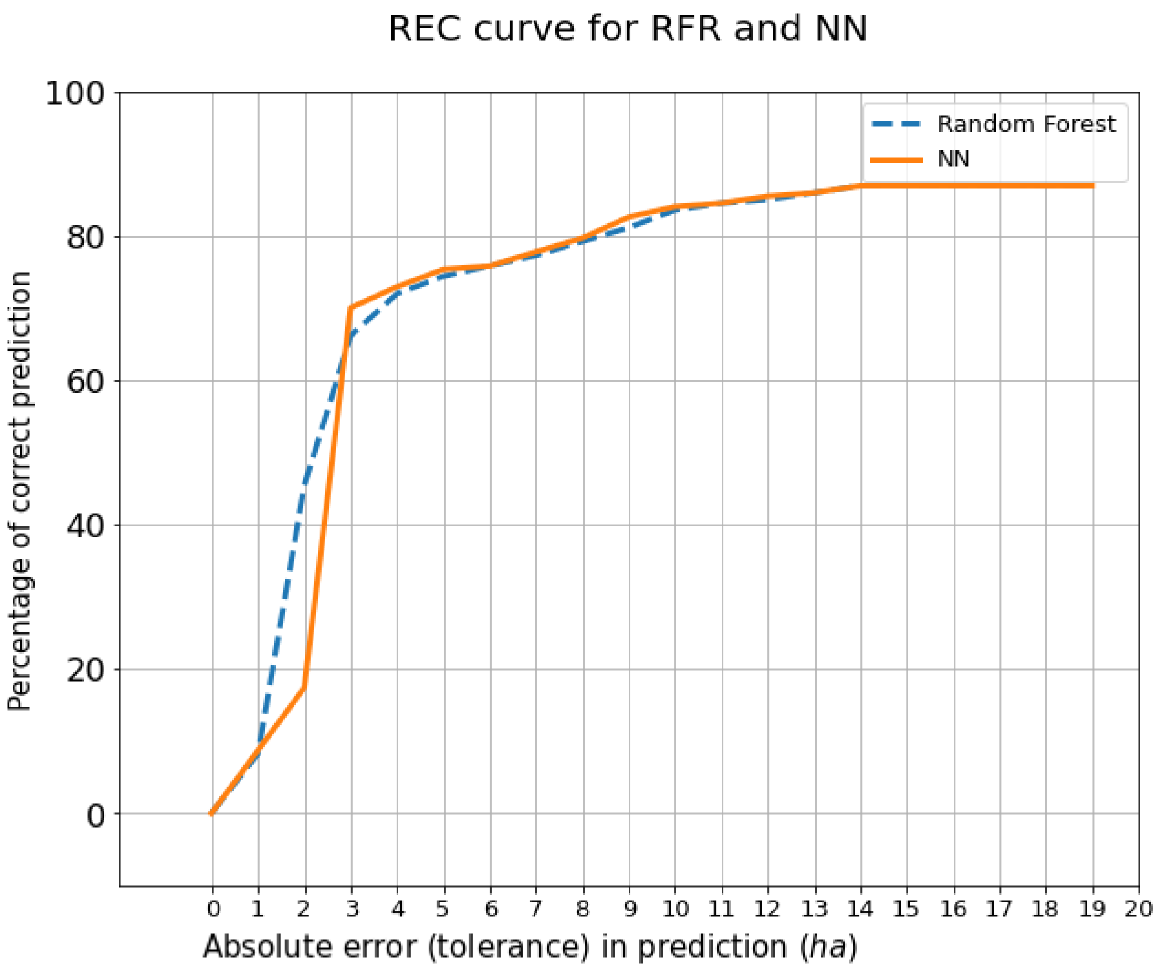


Figure 4.6: Relative performance of Random forest regressor and Neural Network(REC curves)

4.7 LINEAR REGRESSION

Linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Linear regression plays an important role in the field of artificial intelligence such as machine learning. The linear regression algorithm is one of the fundamental supervised machine-learning algorithms due to its relative simplicity and well-known properties.

LINEAR REGRESSION

```
[ ] from sklearn.linear_model import LinearRegression   #Importing linear regression package from sklearn.linear_model
    lm = LinearRegression()   #Creating an object to linear regression
    lm.fit(X, y)   ##Fitting the model

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

Figure 4.7.1: Importing Linear Regression package

- Import linear regression method which is available in linear_model package from scikit learn library
- Once the model is built we need to check for accuracy.
- This can be done using predict method which is used to predict the output.

```
y_pred = lm.predict(X)  ### With help of predict, we are going to find our predicted values
y_pred
```

```
        0.65131886,  0.54228458,  0.56459563,  0.63199172,  0.53803678,
        0.60136047,  0.39505905,  0.4546305 ,  0.61387202,  0.2494343 ,
        0.36337273,  0.36753822,  0.32831805,  0.60296054,  0.52482066,
        0.53943461,  0.53851168,  0.29481433,  0.55786099,  0.3390227 ,
        0.56897206,  0.53994186,  0.51693502,  0.48267641,  0.50366492,
        0.43160331,  0.45625655,  0.37510907,  0.28021793,  0.42854449,
        0.36817175,  0.4200503 ,  0.3544825 ,  0.48024714,  0.38136455,
        0.51179349,  0.36566379,  0.41826757,  0.42524331,  0.37958271,
        0.34535445,  0.51545882,  0.41133437,  0.48828492,  0.57632338,
        0.41131095,  0.57748397,  0.56003905,  0.57748397,  0.57748397,
        0.3504686 ,  0.65213691,  0.40226624,  0.36989985,  0.45444155,
        0.54797053,  0.59685151,  0.45082113,  0.45995944,  0.46448187,
        0.32296289,  0.45008792,  0.37992734,  0.68175072,  0.4862352 ,
        0.46756237,  0.41236318,  0.47737349,  0.60278114,  0.27826641,
        0.55719579,  0.54143811,  0.45880104,  0.45880104,  0.37999693,
        0.39658715,  0.60797478,  0.52638896,  0.39453453,  0.46406592,
        0.56959157,  0.51150749,  0.40434984,  0.68301715,  0.49444548,
        0.55941719,  0.64061753,  0.60417944,  0.58012885,  0.69858909,
        0.72370959,  0.53915602,  0.6323155 ,  0.52037656,  0.44298647,
        0.47844616,  0.5480411 ,  0.52360728,  0.50027989,  0.55516452,
        0.50740113,  0.64077867,  0.47133458,  0.37461691,  0.52305422,
```

Figure 4.7.2: predicting the output

```
y==y_pred   # Compare the actual with the predicted values
```

```
0       False
1       False
2       False
3       False
4       False
        ...
512     False
513     False
514     False
515     False
516     False
Name: Log-area, Length: 517, dtype: bool
```

Figure 4.7.3: Comparing y with y_pred

4.8 R-Squared

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model. Or:

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

```
## r2_value--> to check the model performance
from sklearn.metrics import r2_score
r2_score(y, y_pred)
```

```
0.027927989936573083
```

Figure 4.8: r2_score to check the model performance

The r2_score obtained is 0.02

4.9 MAE and MSE

Mean absolute error (MAE) is a measure of errors between paired observations expressing the same phenomenon.

MAE is not identical to RMSE (root-mean square error), but some researchers report and interpret RMSE as if RMSE reflects the measurement that MAE gives. MAE is conceptually simpler and more interpretable than RMSE. MAE does not require the use of squares or square roots. The use of squared distances hinders the interpretation of RMSE. MAE is simply the average absolute vertical or horizontal distance between each point in a scatter plot and the Y=X line.

In other words, MAE is the average absolute difference between X and Y. MAE is fundamentally easier to understand than the square root of the average of the sum of squared deviations. Furthermore, each error contributes to MAE in proportion to the absolute value of the error, which is not true for RMSE. See the example above for an illustration of these differences.

```
[70] from sklearn.metrics import mean_squared_error as mse   #Importing mean_squared_error as mse from sklearn.metrics
     from sklearn.metrics import mean_absolute_error as mae  #Importing mean_absolute_error as mae from sklearn.metrics
     print("mse=", mse(y,y_pred))  #Printing mean_squared_error as 'mse='
     print("mae=", mae(y,y_pred))  #Printing mean_absolute_error as 'mae='

mse= 0.3578585376627844
mae= 0.4931553756634356
```

Figure 4.9: mae and mse scores

Best Algorithm for the project:

The best model is Random Forest Regressor which has RMSE value 0.628 for which we are using GridSearchCV.

The lower values of RMSE indicated better fit models.

Scikit-learn has the functionality of trying a bunch of combinations and see what works best, built in with GridSearchCV. The CV stands for cross-validation.
GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

CONCLUSION:

Forest fires cause a significant environmental damage while threatening human lives. In the last two decades, a substantial effort was made to build automatic detection tools that could assist Fire Management Systems (FMS). The three major trends are the use of satellite data, infrared/smoke scanners and local sensors (e.g. meteorological).The advantage is that such data can be collected in real-time and with very low costs, when compared with the satellite and scanner approaches. Recent real-world data, from the northeast region of Portugal, was used in the experiments. The database included spatial, temporal, components from the Canadian Fire Weather Index (FWI) and four weather conditions.

This problem was modeled as a regression task, where the aim was the prediction of the burned area. The drawback is the lower predictive accuracy for large fires. To our knowledge, this is the first time the burn area is predicted using only meteorological based data and further exploratory research is required. As argued in , predicting the size of forest fires is a challenging task. To improve it, we believe that additional information is required, such as the type of vegetation and firefighting intervention (e.g. time elapsed and firefighting strategy). Since the FWI system is widely used around the world, further research is need to confirm if direct weather conditions are preferable than accumulated values, as suggested by this study.

Finally, since large fires are rare events, outlier detection techniques will also be addressed.

REFERENCES:

[1] *https://archive.ics.uci.edu/ml/datasets/forest+fires*

[2] *https://en.wikipedia.org/wiki/Machine_learning*

[3] *https://en.wikipedia.org/wiki/Deep_learning*

[4] *https://github.com/maansi0706/Summer-Internship-Project/tree/master*