A Project on

Prediction of Forest Fires

Submitted for fulfillment of award of

Bachelor of Technology

In

Information Technology

By

Harshdeep Rana (1602713043)
Ishaan Bhattacharya (1602713046)
Ishita Tyagi (1602713048)

Under the Guidance of

Ms. Yogita Chhabra

AJAY KUMAR GARG ENGINEERING COLLEGE, GHAZIABAD

YEAR: 2019-2020

AJAY KUMAR GARG ENGINEERING COLLEGE, GHAZIABAD

CERTIFICATE

Certified that ***Harshdeep Rana, Ishaan Bhattacharya and Ishita Tyagi*** *have carried out the Project work entitled* ***Prediction of Forest Fires*** *for the award of* ***Bachelor of Technology*** *from Ajay Kumar Garg Engineering College, Ghaziabad under our supervision.*

To the best of our knowledge, this work has not been submitted earlier to any university for the award of any degree.

Ms. Yogita Chhabra
Assistant Prof. IT Department
Project Guide

Dr. Anu Chaudhary
Head of Department
Information Technology

ACKNOWLEDGEMENT

We would like to express our sincerest gratitude to all the people who have contributed towards the successful completion of our project.

We would like to extend our heartfelt thanks to the Head of Information Technology Department **Dr. Anu Chaudhary,** for nurturing a congenial yet competitive environment in the department, which motivates all the students to pursue higher goals.

We want to express our special gratitude to our guide **Ms. Yogita Chhabra**, **Assistant Professor,** Department of Information Technology, Ajay Kumar Garg Engineering College, Ghaziabad for her constant support, guidance, encouragement and much needed motivation. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us.

Last but not the least; we would like to extend our thanks to all the teaching and non-teaching staff members of our department, and to all our colleagues who helped us in completion of the project.

1. **Harshdeep Rana**
2. **Ishaan Bhattacharya**
3. **Ishita Tyagi**

DECLARATION

We hereby declare that the **"Prediction Of Forest Fires"** submitted to the department of Information technology, Ajay Kumar Garg Engineering College Ghaziabad is a record of an original work done by us under the guidance of **Ms. Yogita Chhabra, Assistant Professor,** Department of Information Technology, Ajay Kumar Garg Engineering College Ghaziabad and this project work is submitted as a part of fulfillment of award of the degree of Bachelor of Technology under AKTU.

Name and Signature of Students:

1. Harshdeep Rana (signature)

   Roll No. 1602713043

2. Ishaan Bhattacharya (signature)

   Roll No. 1602713046

3. Ishita Tyagi (signature)

   Roll No. 1602713048

Date: 17/05/2020

Place: Ghaziabad (U.P.)

TABLE OF CONTENTS

Chapters ***Page No.***

LIST OF FIGURES

LIST OF ABBREVIATIONS

Keywords	**Abbreviations**
FWI	Fire Weather Index
DM	Data Mining
KDD	Knowledge Discovery in Databases
FFMC	Fine Fuel Moisture Code
DMC	Duff Moisture Code
DC	Drought Code
ISI	Initial Spread Index
BUI	Build Up Index
FMP	Forest Management Planning
WFDDI	Wildfire Destruction Danger Index
WS	Wildfire Simulator
SEI	Soil Erosion Index
FA	Financial Analysis
PR	Timber and Non-timber Production
DB	Database
GB	Geo-database

Chapter I: INTRODUCTION

The most common hazard in forests is forests fire. Forests fires are as old as the forests themselves. They pose a threat not only to the forest wealth but also to the entire regime to fauna and flora seriously disturbing the bio-diversity and the ecology and environment of a region. During summer, when there is no rain for months, the forests become littered with dry senescent leaves and twinges, which could burst into flames ignited by the slightest spark. Forest fire causes imbalances in nature and endangers biodiversity by reducing faunal and floral wealth.

Forest fires (also called wildfires) affect forest preservation, create economical and ecological damage and cause human suffering. Such phenomenon is due to multiple causes (e.g. human negligence and lightning) and despite an increasing of state expenses to control this disaster, each year millions of forest hectares (ha) are destroyed all around the world.

Fast detection is a key element for a successful firefighting. Since traditional human surveillance is expensive and affected by subjective factors, there has been an emphasis to develop automatic solutions. These can be grouped into three major categories satellite-based, infrared/smoke scanners and local sensors (e.g. meteorological). Satellites have acquisition costs, localization delays and the resolution is not adequate for all cases. Moreover, scanners have a high equipment and maintenance costs.

Fast detection is a key element for a successful firefighting. Since traditional human surveillance is expensive and affected by subjective factors, there has been an emphasis to develop automatic solutions.

Therefore, we came up with our project as Prediction of Forest Fires which will help a lot in detecting the forest fires with the help of Data Mining algorithms and Artificial Intelligence.

1.1) Preface

Forest fires represent a considerable threat to the Mediterranean forests. It is estimated that about 50,000 fires occur each year in the Mediterranean basin and affect more than 600,000 ha. To face this permanent threat, the Mediterranean countries have mobilized and organized for a long time increasingly efficient forest fire prevention and suppression systems, thus gradually according to their culture, their means and their national preferences, the Mediterranean countries have developed a considerable knowledge, tools and adapted methodologies, typical for each country.

Unfortunately, this information remains scattered and is even sometimes not written down. The exchange of information and experiences between countries constitute an effective mean to contribute to the reinforcement of forest fire prevention and suppression.

We used Data Mining algorithms and Artificial Intelligence for Prediction of Forest fire, where the emphasis is the use of real-time and non-costly meteorological data (i.e. temperature, relative humidity, rain and wind). Since, human experts are limited and may overlook important details. Moreover, classical statistical analysis breaks down when such vast and/or complex data is present.

1.2) Background

In the past, meteorological data has been incorporated into numerical indices, which are used for prevention (e.g. warning the public of a fire danger) and to support fire management decisions (e.g. level of readiness, prioritizing targets or evaluating guidelines for safe fire fighting). In particular, the Canadian Forest Fire Weather Index (FWI) system was designed in the 1970s when computers were scarce, thus it required only simple calculations using look-up tables with readings from four meteorological observations (i.e. temperature, relative humidity, rain and wind) that could be manually collected in weather stations. On the other hand, the interest in Data Mining (DM), also known as Knowledge Discovery in Databases (KDD), arose due to the advances of Information Technology, leading to an exponential growth of business, scientific and engineering databases [1]. All this data holds valuable information, such as trends and patterns, which can be used to improve decision making. Yet, human experts are limited and may overlook important details. Moreover, classical statistical analysis breaks down when such vast and/or complex data is present. Hence, the alternative is to use automated Data Mining tools to analyse the raw data and extract high-level information for the decision-make. In contrast with these previous works, we present a forest fire approach using Data Mining algorithms and Artificial Intelligence. The algorithm depends on previous weather conditions in order to predict the fire hazard level of a day.

1.3) Purpose

Our goal is to predict the burned area of forest fires. Such knowledge is particularly useful for improving fire fighting resource management (e.g. prioritizing targets for air tankers and ground crews).

Since traditional human surveillance is expensive and affected by subjective factors, there has been an emphasis to develop automatic solutions. These can be grouped into three major categories: satellite-based, infrared/smoke scanners and local sensors (e.g. meteorological). Satellites have acquisition costs, localization delays and the resolution is not adequate for all cases. Moreover, scanners have a high equipment and maintenance costs. Weather conditions, such as temperature and air humidity, are known to affect fire occurrence. Since automatic meteorological stations are often available, such data can be collected in real-time, with low costs.

Several experiments were carried out by considering Data Mining algorithms and then they are fed into normalization. Then the output 'area' was first transformed with a ln(x+1) function. Then experiments were conducted using a 10-fold (cross-validation) x 30 runs. The best RMSE was attained by the Neural Networks. An analysis to the Regression Error Curve (REC) shows that the SVR model predicts more examples within a lower admitted error. In effect, the SVR model predicts better small fires, which are the majority.

1.4) Scope of the Project

The objective of this project is to create a Regression Model to predict the size of forest fires, where the emphasis is the use of real-time and non-costly meteorological data. We will use recent real-world data, collected from the northeast region of Portugal, with the aim of predicting the burned area (or size) of forest fires.

This study will consider forest fire data from the Montesano natural park, from the Trás-os-Montes northeast region of Portugal. Inserted within a supra-Mediterranean climate, the average annual temperature is within the range 8 to 12◦C. The data used in the experiments was collected from January 2000 to December 2003 [2].

We present a novel Data Mining forest fire approach, where the emphasis is the use of real-time and non-costly meteorological data. We will use recent real-world data, collected from the northeast region of Portugal, with the aim of predicting the burned area (or size) of forest fires.

1.5) Theoretical Concepts

1.5.1) Regression Models

In this we will consider ten different regression models to predict the size of the forest fires from the attributes in the dataset. The regression models are:

1. Linear Regression: Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model. A linear regression line has an equation of the form $Y = a + bX$, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when $x = 0$).

2. Ridge Regression: Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multi collinearity occurs,

least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable.

3. Lasso Regression: Lasso regression is type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of muticollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination. The acronym "LASSO" stands for Least Absolute Shrinkage and Selection Operator.

4. ElasticNet Regression: ElasticNet merged as a result of critique on lasso, whose variable selection can be too dependent on data and thus unstable. The solution is to combine the penalties of ridge regression and lasso to get the best of both worlds. In statistics and, in particular, in the fitting of linear or logistic regression models, the elastic net is a regularized regression method that linearly combines the L_1 and L_2 penalties of the lasso and ridge methods.

5. Baggging Regression: A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

6. Random Forest Regression: 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. A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement.

7. ExtraTrees Regression: ExtraTree Regression implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

8. KNeighbours Regression: The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points

in the training set. From our example, we know that ID11 has height and age similar to ID1 and ID5, so the weight would also approximately be the same.

9. Decision Tree Regression: Decision Tree regression is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve. A decision tree is arriving at an estimate by asking a series of questions to the data, each question narrowing our possible values until the model get confident enough to make a single prediction. The order of the question as well as their content is being determined by the model. In addition, the questions asked are all in a True/False form.

10. Support Vector Regression: The Support Vector Regression uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reasons, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

1.5.2) Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.
For example, consider a data set containing two features, age (x1), and income (x2), where age ranges from 0–100, while income ranges from 0–20,000 and higher. Income is about 1,000 times larger than age and ranges from 20,000–500,000. So, these two features are in very different ranges. When we do further analysis, like multivariate linear regression, for example, the attributed income will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor.

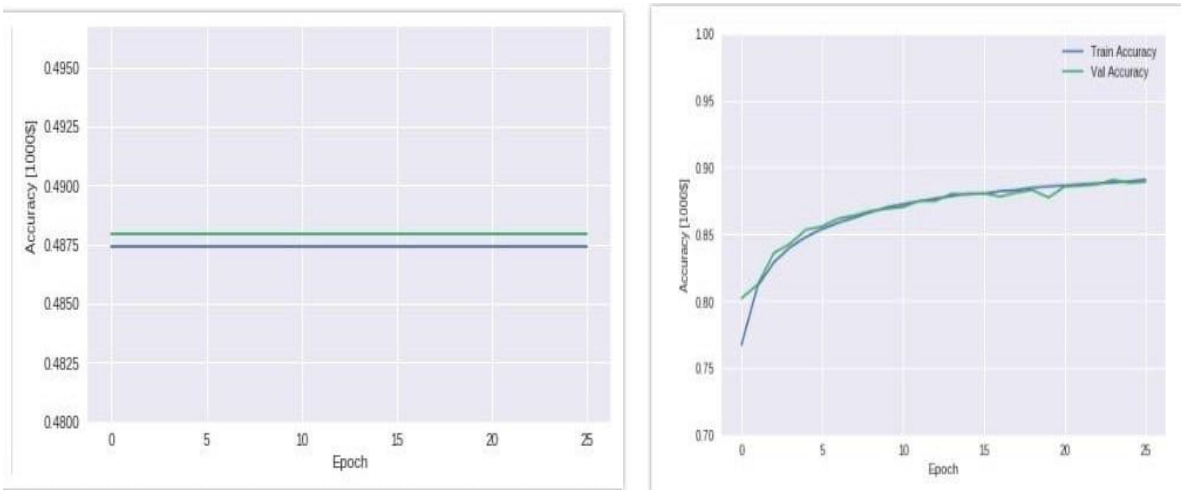

Figure 1.5.2: L- Model Accuracy without normalized data, R- Model Accuracy with normalized data

From the above graphs, we see that model 1 (left side graph) have very low validation accuracy (48%) and a straight line for accuracy is coming in a graph for both test and train data. Straight line for accuracy means that accuracy is not changing with the number of epochs and even at epoch 26 accuracy remains the same (what it was at an epoch 1). The reason for straight accuracy line and low accuracy is that the model is not able to learn in 26 epochs. Because different features do not have similar ranges of values and hence gradients may end up taking a long time and can oscillate back and forth and take a long time before it can finally find its way to the global/local minimum. To overcome the model learning problem, we normalize the data. We make sure that the different features take on similar ranges of values so that gradient descents can converge more quickly. From the above right-hand side graph, we can see that after normalizing the data in model 2 accuracy is increasing with every epoch and at epoch 26, accuracy reached 88.93%.

1.5.3) Log-Transformation

The log transformation, a widely used method to address skewed data, is one of the most popular transformations used in biomedical and psychosocial research. Many features are normally distributed and many machine learning algorithms also tend to assume a normal distribution. However, there are also attributes that have a log-normal distribution. In such a distribution, most of the observations have smaller values and few take on higher values. A feature could have a positive skew where the tail of the distribution is on the right or a negative skew where its tail is on the left. The log transformation works well on some distributions because there are often values in nature that follow this distribution. For example, the amount of rainfall when there is little to some rainfall most of the time and higher amounts of rainfall on fewer occasions. Stock prices are often described using a log-normal distribution. Unfortunately, its popularity has also made it vulnerable to misuse – even by statisticians – leading to incorrect interpretation of experimental results. Such misuse and misinterpretation is not unique to this particular transformation; it is a common

problem in many popular statistical methods. For example, the two-sample t-test is widely used to compare the means of two independent samples with normally distributed (or approximately normal) data, but many researchers take this critical assumption for granted, using t-tests without bothering to check or even acknowledge this underlying assumption. Another example is the Cox regression model used in survival analysis; many studies apply this popular model without even being aware of the proportionality assumption (i.e., the relative hazard of groups of interest is constant over time) required for valid inference. A transformation can be applied to the dependent and independent variables to achieve a more normal distribution. Whether it would be beneficial to apply such a transformation has to be determined based on the distribution.

1.5.4) Binary Classification

Binary or binomial classification is the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule. Contexts requiring a decision as to whether or not an item has some qualitative property, some specified characteristic, or some typical binary classification include:

1. Medical testing to determine if a patient has certain disease or not – the classification property is the presence of the disease.
2. A "pass or fail" test method or quality control in factories, i.e. deciding if a specification has or has not been met – a Go/no Go classification.
3. Information retrieval, namely deciding whether a page or an article should be in the result set of a search or not – the classification property is the relevance of the article, or the usefulness to the user.

Given a collection of objects let us say we have the task to classify the objects into two groups based on some feature(s). For example, let us say given some pens and pencils of different types and makes, we can easily separate them into two classes, namely pens and pencils. This seemingly trivial task, if we take a moment would seem to involve a lot of underlying computation. The question to ask would be what is our basis for classification? And how perhaps might we incorporate the principle to allow for efficient classification by 'intelligent algorithms'.

Binary classification is dichotomization applied to practical purposes, and in many practical binary classification problems, the two 2 groups are not symmetric – rather than overall accuracy, the relative proportion of different types of errors is of interest. For example, in medical testing, a false positive (detecting a disease when it is not present) is considered differently from a false negative (not detecting a disease when it is present).

1.5.5) k-fold Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
4. Take the group as a hold out or test data set
5. Take the remaining groups as a training data set
6. Fit a model on the training set and evaluate it on the test set
7. Retain the evaluation score and discard the model
8. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Chapter II: REQUIREMENT ANALYSIS AND FEASIBILITY STUDY

2.1) Requirement Analysis

In this phase, we try to find out what are the requirements that need to be fulfilled by the proposed project, what all aspects we need to cover so as to make this project worth of usage.

2.1.1) Information Gathering

Information gathering is the collection of data for dealing with the individual's or the organization's current situation. More data means more and better ways of dealing with the current situation. More data broadens the minds of those who will use the data to solve current organizational problems. New ideas come more easily if there are lots of facts to be used as bases.
The collection of observations on forest fires over a long period provides essential information for the development of fire fighting and prevention policies. It is essential to obtain reliable figures and statistics on fires in order to improve the understanding of the phenomenon, to prioritize technical and economic choices for fire prevention and suppression, and to evaluate the effectiveness of these.

In this project, we have used data from Canadian Forest Fire Weather Index (FWI) system. It has applications in the Mediterranean region. The Canadian Forest Fire Weather Index (FWI) System consists of six components that account for the effects of fuel moisture and wind on fire behaviour.

2.1.2) Functional Requirements

The functional requirements are completed with the following set of operational requirements:

1. The system should be able to handle both static and real-time data (i.e. data that evolve over time).
2. The system model base should contain mathematical models and empirical rules that can provide estimations of the impact and the consequences of the incident, and optimize the utilization of the available resources.
3. The model base should contain empirical rules for associating the appropriate response actions to the incident characteristics.
4. The model base should be able to operate under limited data availability, and should take into consideration the fact that the quality and the quantity of data evolve over time.
5. The model base should contain mathematical models and algorithms for implementing routing plans.

6. The data base should be able to retrieve information from external sources or from data bases spatially distributed. For example, experience should be easily extracted from forest fires in the region or other regions and from international data bases as well.
7. The system should take advantage of the innovative telematics, networking and communications technologies.

2.1.2.1) Input and Output Requirements

The most important input requirements have to do with past fire occurrence statistics and with terrain value. Data should also be available on fire causes. Depending on the models included and the degree of sophistication of the system, socio-economic data may also be needed.

In addition to these, basic data like the ones required for pre-suppression planning are also needed here:

1. Topographic information (DTM)
2. Vegetation distribution information (both vegetation types and vegetation as fuel maps)
3. Detailed road network map
4. Map of firebreaks, fuel breaks and other fuel treatment zones
5. Map of especially high fire danger zones (e.g. areas of forest blow-downs, snow damage, excessive slash, etc.)
6. Maps of sensitive areas (wild land-urban interface areas, tourist areas, national parks, high-biodiversity areas, etc.), other facilities and infrastructures, to be used for setting protection priorities
7. Ownership map

A database of potential prevention actions should be developed, including rules of when to employ each type of action and the associated cost of the actions.

2.1.2.2) Hardware and Software Requirements

Hardware requirements

1. Processor: Intel Xeon E2630 v4 – 10 core processor, 2.2 GHz with Turbo boost up to 3.1 GHz. 25 MB Cache.
2. RAM: 8 GB.
3. Operating System: Windows XP/7/10.

Software Requirements

1. Coding Language: Python 3
2. Scikit Learn: Scikit-learn provide a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial uses. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn.

3. Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning
4. Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications.
5. CSV: A comma-separated values file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

2.1.3) Non-functional Requirements

1. Performance Requirements: The system must be interactive and the delays involved must be less. So in every action response of the system, there are no immediate delays. In case of opening windows forms, of popping error messages and saving the settings or sessions there is delay much below 2 seconds.

2. Safety Requirements: Information transmission should be securely received from the server without any changes in information.

3. Security Requirements: The data should be taken from a trusted Government website for the authentication, in order to stay clear of the spam or false data. The main security concern is for users account hence proper login mechanism should be used to avoid hacking. The tablet id registration is way to spam check for increasing the security. Hence, security is provided from unwanted use of recognition software.

4. Reliability: As the system provide the right tools for discussion, problem solving it must be made the system reliable in its operations and for securing the sensitive details.

5. Usability: As the system is easy to handle and navigates in the most expected way with no delays. In that case the system program reacts accordingly and transverses quickly between its states.

2.2) Feasibility Study

2.2.1) Economic Feasibility

We will be using Python language to develop the project with the help of some open source libraries such as pandas, Scikit Learn and other libraries which are free to use.

2.2.2) Technical Feasibility

This project is based on machine learning that uses natural language processing. We will use Python for programming and require some libraries such as pandas, Scikit Learn which are easily available. We need to process large data set so, any mid-specs computer with internet facility is just enough.

2.2.3) Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. Operational feasibility reviews the willingness of the organization to support the proposed system. This is probably the most difficult of the feasibilities to gauge. In order to determine this feasibility, it is important to understand the management commitment to the proposed project. If the request was initiated by management, it is likely that there is management support and the system will be accepted and used. The operational feasibility is the one that will be used effectively after it has been developed.

2.2.4) Scheduling Feasibility

With the required availability of resources project will be completed within the specified time frame. Here, we have studied all the feasibility aspects of the project under consideration to check out if the project is feasible with the decided requirements and availability of information, technologies, and budget.

Chapter III: SYSTEM ANALYSIS AND DESIGN

3.1) System Analysis

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose. Analysis specifies what the system should do. Here, in this project we analysed the dataset where wildfire occurred previously and how to detect these forest fires with the help of meteorological data using data mining algorithms and artificial intelligence. We identified our purposes which are identifying best algorithm which predicts forest fire with most accuracy among all used algorithms.

3.1.1) Existing System

Existing system includes Earth-orbiting satellites and even air-floating devices for observation and detection of forest fires. Satellite images gathered by two main satellites launched for forest fire detection purposes, the advanced very high resolution radiometer, launched in 1998, and the moderate resolution imaging spectroradiometer, launched in 1999, have been used. Unfortunately, these satellites can provide images of the regions of the earth every two days and that is a long time for fire scanning; besides the quality of satellite images can be affected by weather conditions. Any existing satellite-based observations for forest fires suffer from severe limitations resulting in a failure in speedy and effective control for forest areas [3].

3.1.2) Proposed System

In contrast with these previous works, we present a novel Data Mining and Artificial Intelligence forest fire approach, where the emphasis is the use of real-time and non-costly meteorological data. We will use recent real-world data, collected from the northeast region of Portugal, with the aim of predicting the burned area (or size) of forest fires. Several experiments were carried out by considering Data Mining algorithms and then they are fed into normalization. Then the output ‘area’ was first transformed with a ln(x+1) function. Then experiments were conducted using a 10-fold (cross-validation) x 30 runs. The best RMSE was attained by the Neural Networks model. An analysis to the Regression Error Curve (REC) shows that the SVR model predicts more examples within a lower admitted error. In effect, the SVR model predicts better small fires, which are the majority.

3.2) System Design

3.2.1) Use Case Diagram

Simplest representation of a user's interaction that shows the relationship between the user and the different use cases in which user involved can be seen from below mentioned figure.

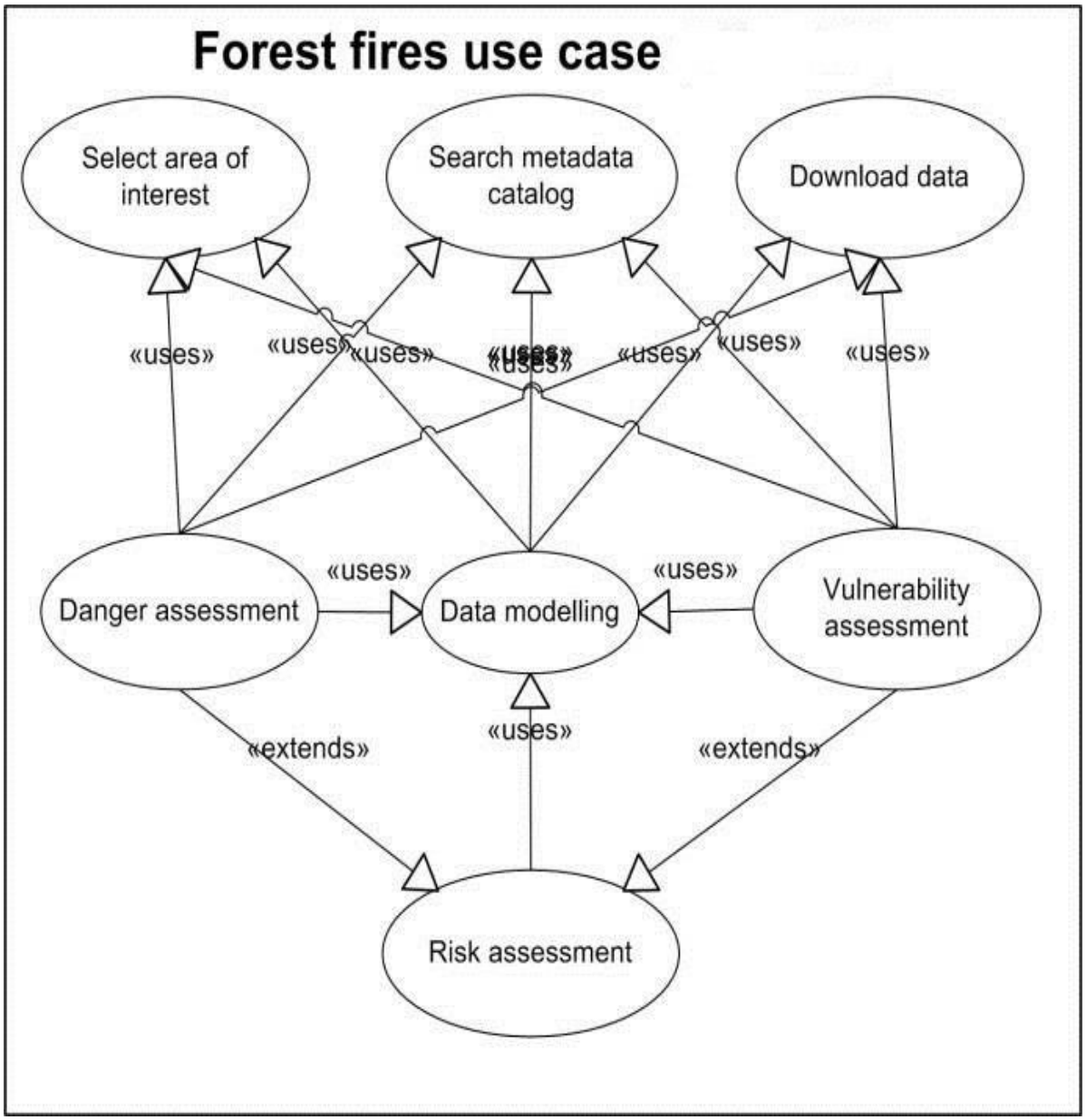

Figure 3.2.1: Use Case Diagram

3.2.2) E-R Diagram

An entity-relationship diagram is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities [4], can be shown from figure.

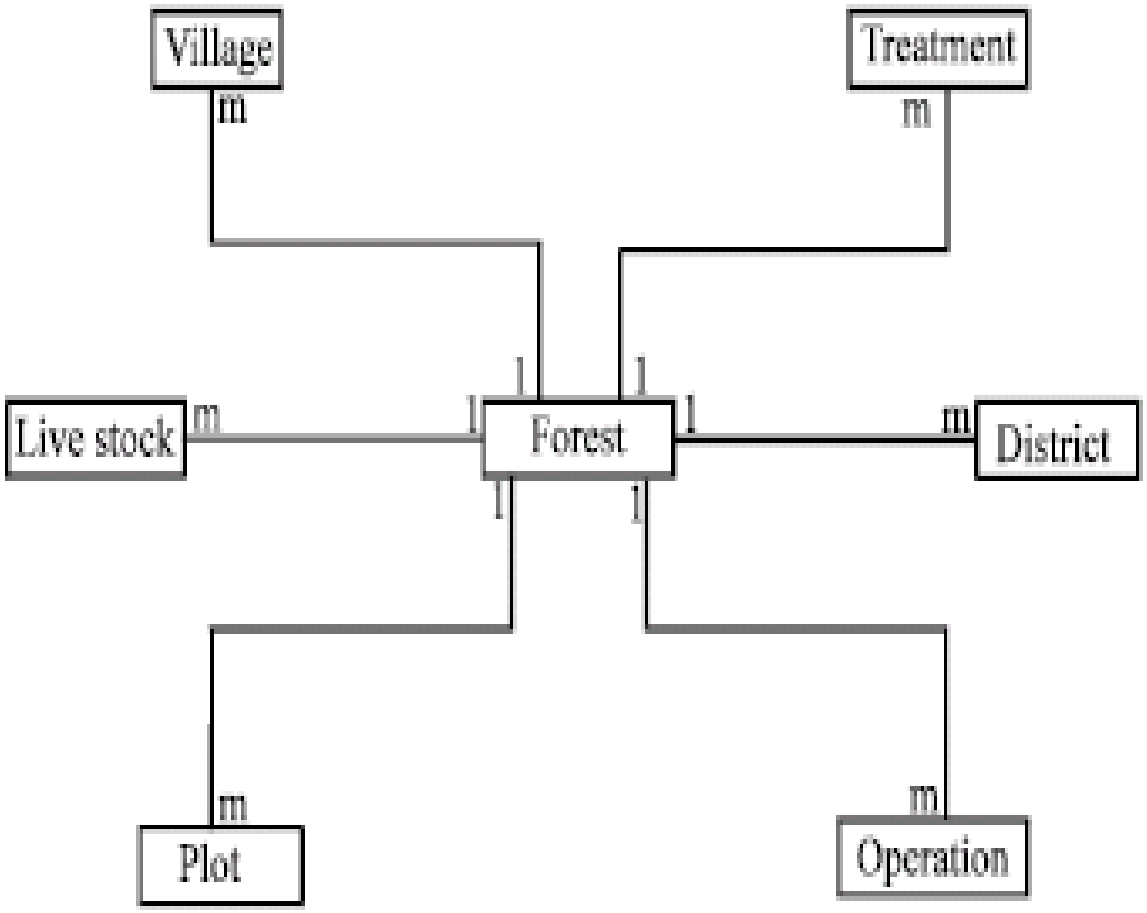

Fig 3.2.2: E-R Diagram

3.2.3) Sequence Diagram

FMP stands for Forest Management Planning Expert System, WFDDI stands for Wildfire Destruction Danger Index Expert System, WS stands for Wildfire Simulator, SEI stands for Soil Erosion Index, PR stands for Timber and non-Timber Production, FA stands for Financial Analysis, GB stands for Geo-database, and DB stands for Database**.**

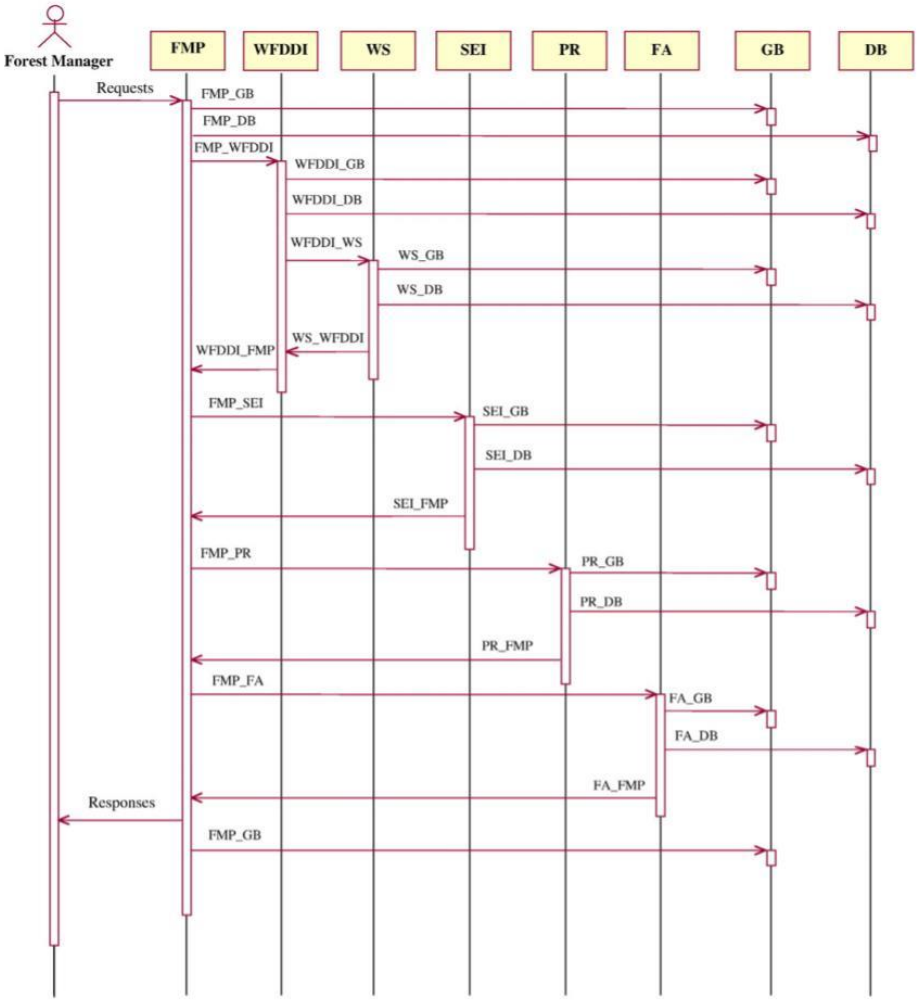

Figure 3.2.3: Sequence Diagram

3.2.4) Gantt Chart

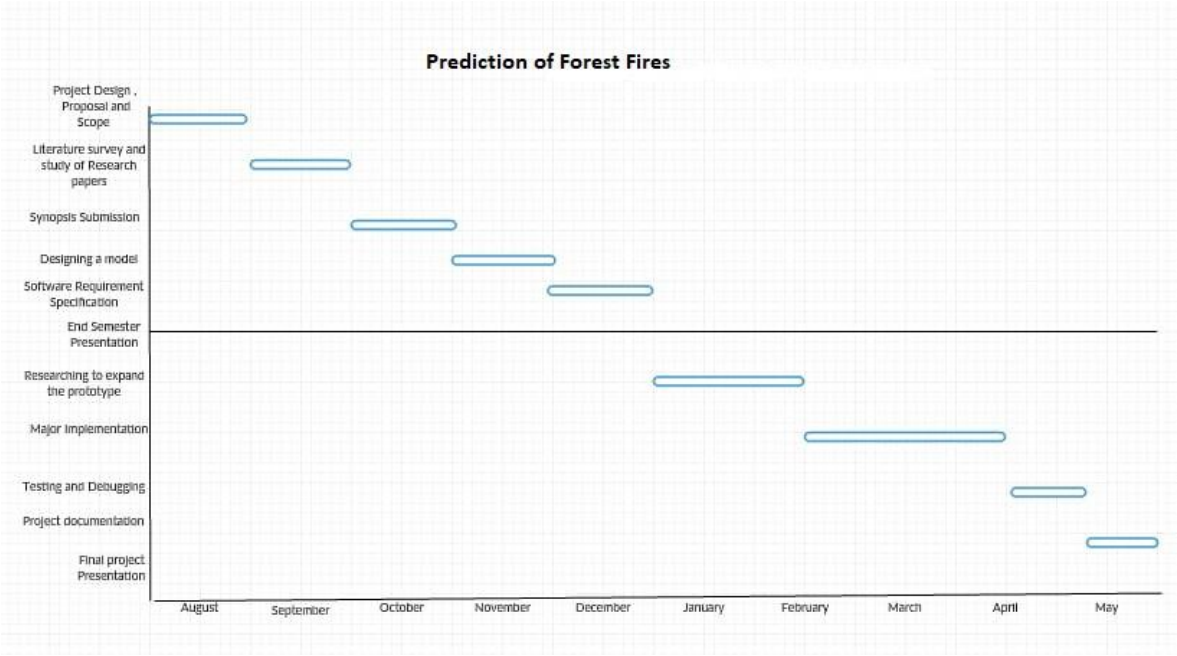

Fig 3.2.4: Gantt Chart

3.3) SRS (SOFTWARE REQUIREMENT SPECIFICATIONS)

The most creative and challenging face of the system development is the SRS. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Design goes through the logical and physical stages of development.

The forest Fire Weather Index (FWI) is the Canadian system for rating fire danger and it includes six components; Fine Fuel Moisture Code (FFMC), Duff Moisture Code (DMC), Drought Code (DC), Initial Spread Index (ISI), Buildup Index (BUI) and FWI. The first three are related to fuel codes: the FFMC denotes the moisture content surface litter and influences ignition and fire spread, while the DMC and DC represent the moisture content of shallow and deep organic layers, which affect fire intensity. The ISI is a score that correlates with fire velocity spread, while BUI represents the amount of available fuel [5].

The FWI index is an indicator of fire intensity and it combines the two previous components. Although different scales are used for each of the FWI elements, high values suggest more severe burning conditions. Also, the fuel moisture codes require a memory (time lag) of past weather conditions: 16 hours for FFMC, 12 days for DMC and 52 days for DC.

Description of work-flows performed by the system:

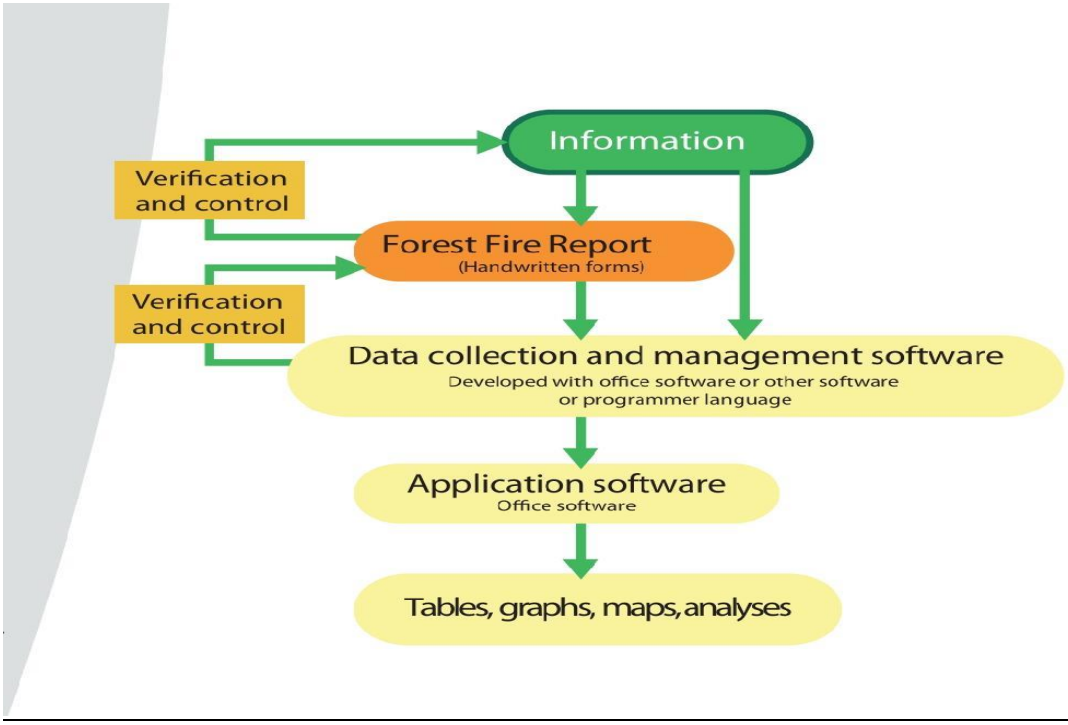

Figure 3.3: Work-flow

3.4) Snap Shots with Details

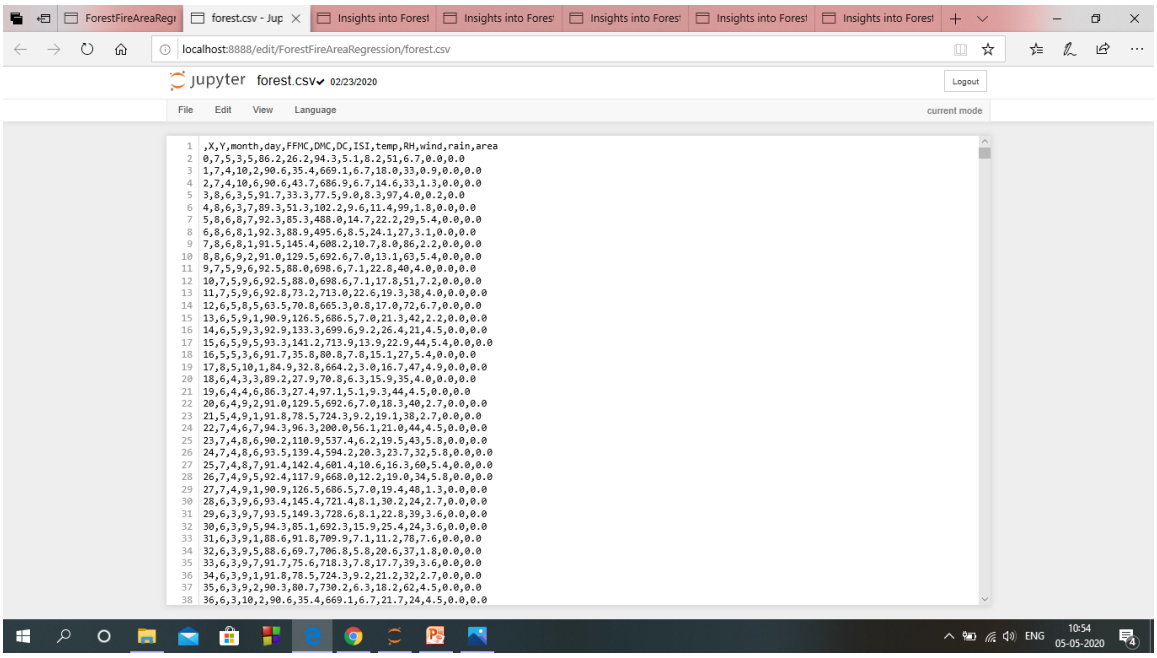

Figure 3.4.1: Dataset

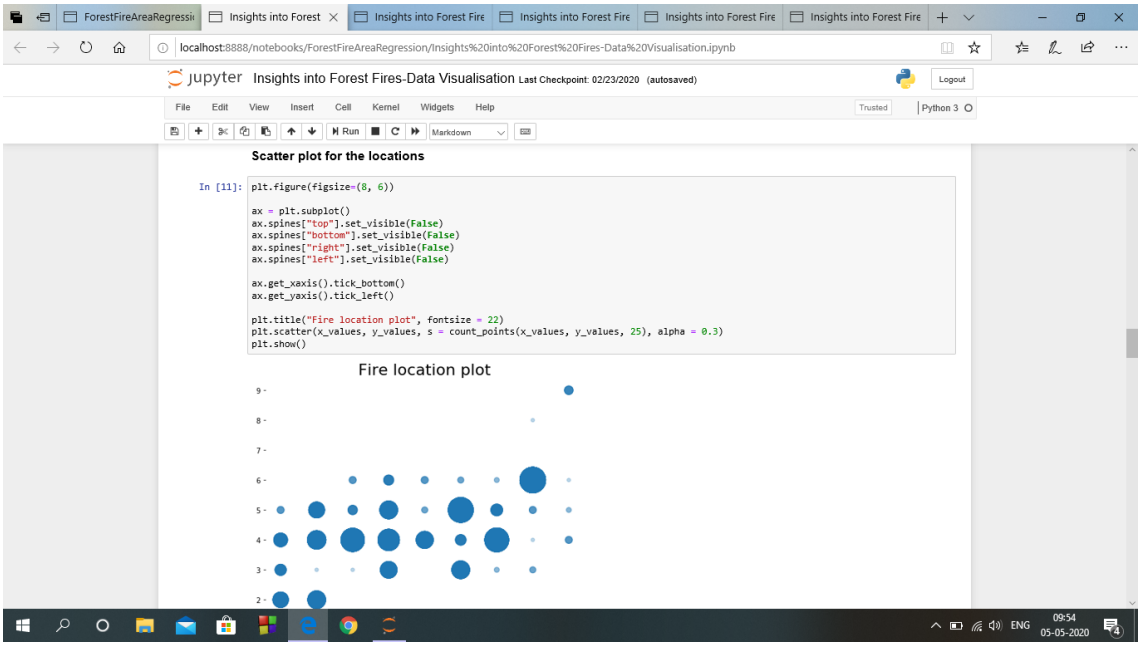

Figure 3.4.2: Fire Location plot of the dataset

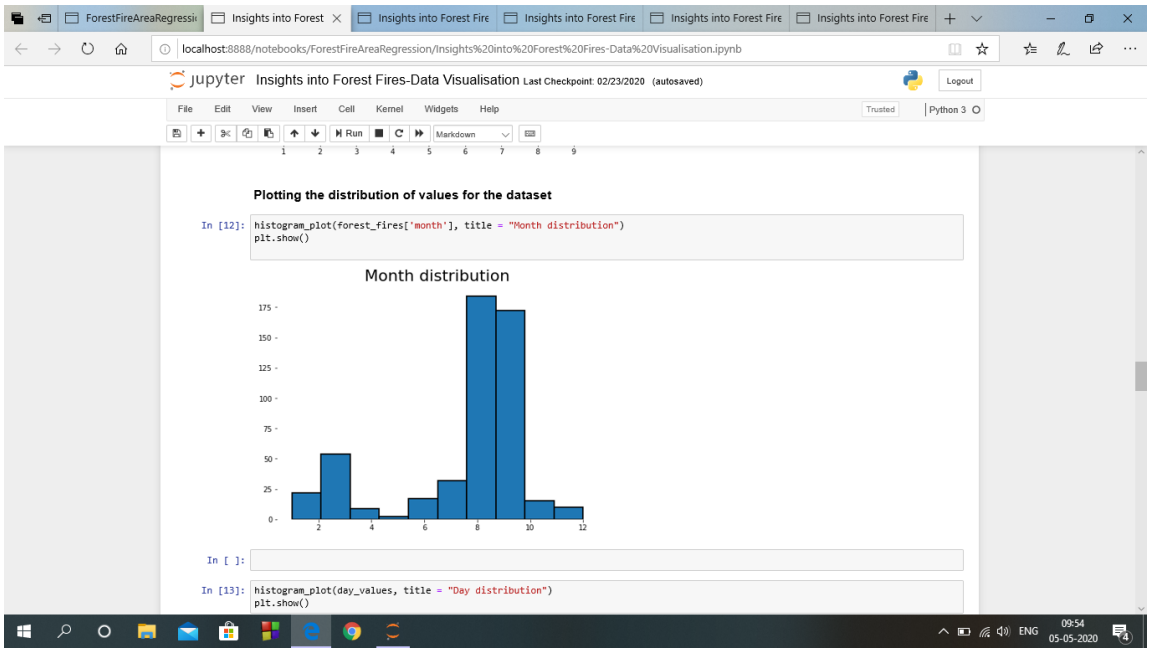

Figure 3.4.3: Histogram of Month Distribution

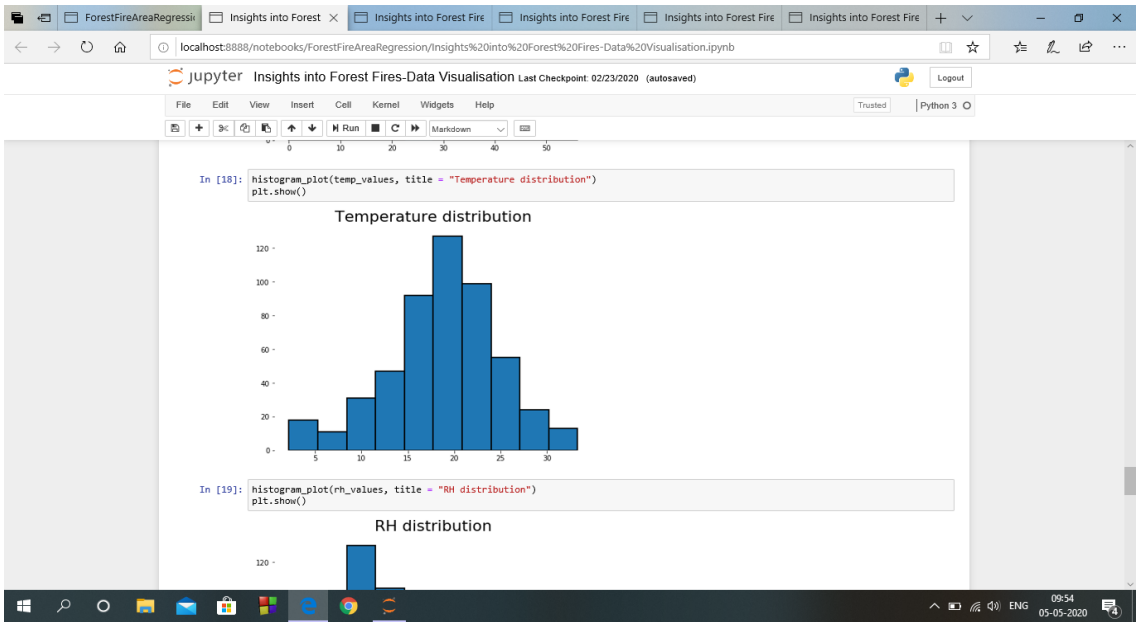

Figure 3.4.4: Histogram of Temperature distribution

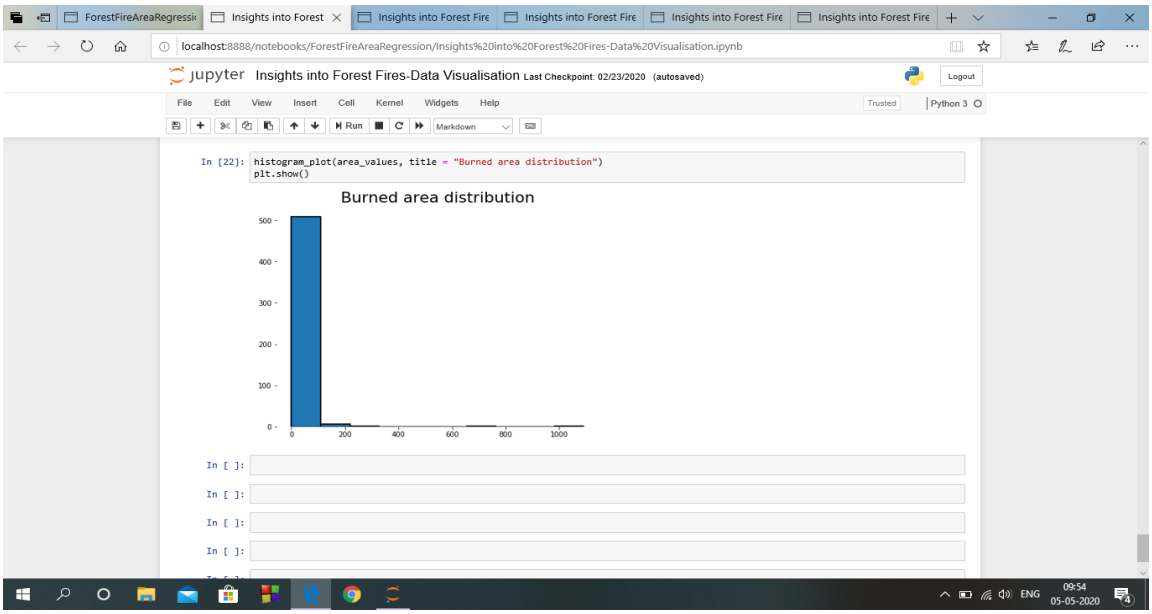

Figure 3.4.5: Histogram of Burned Area

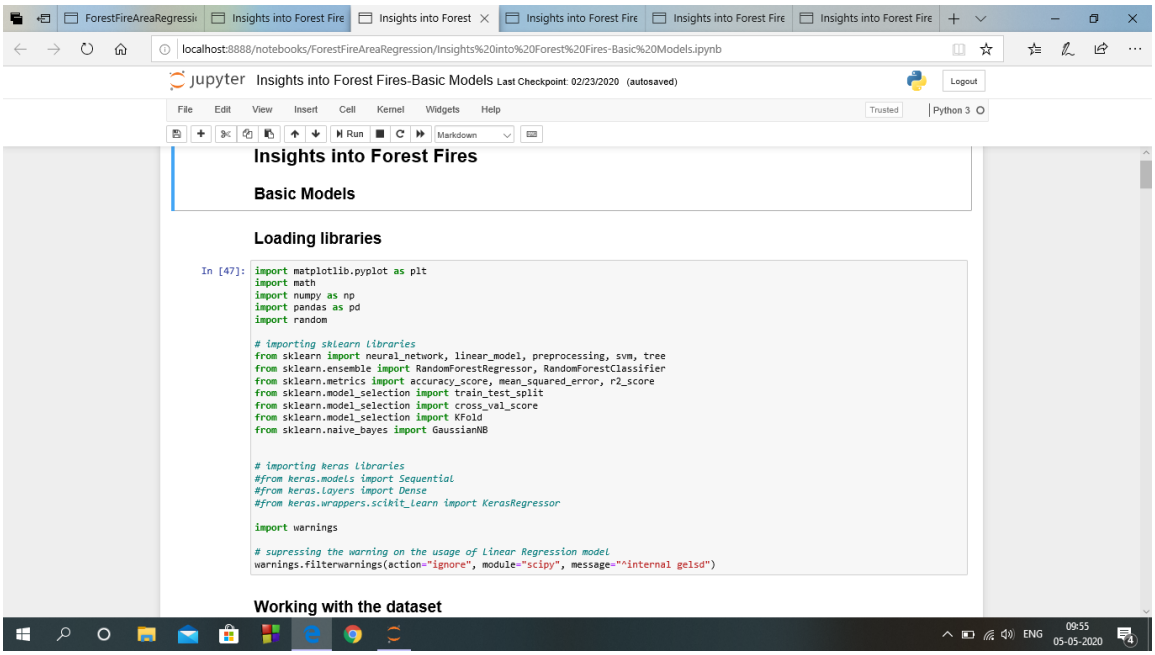

Figure 3.4.6: Importing Libraries

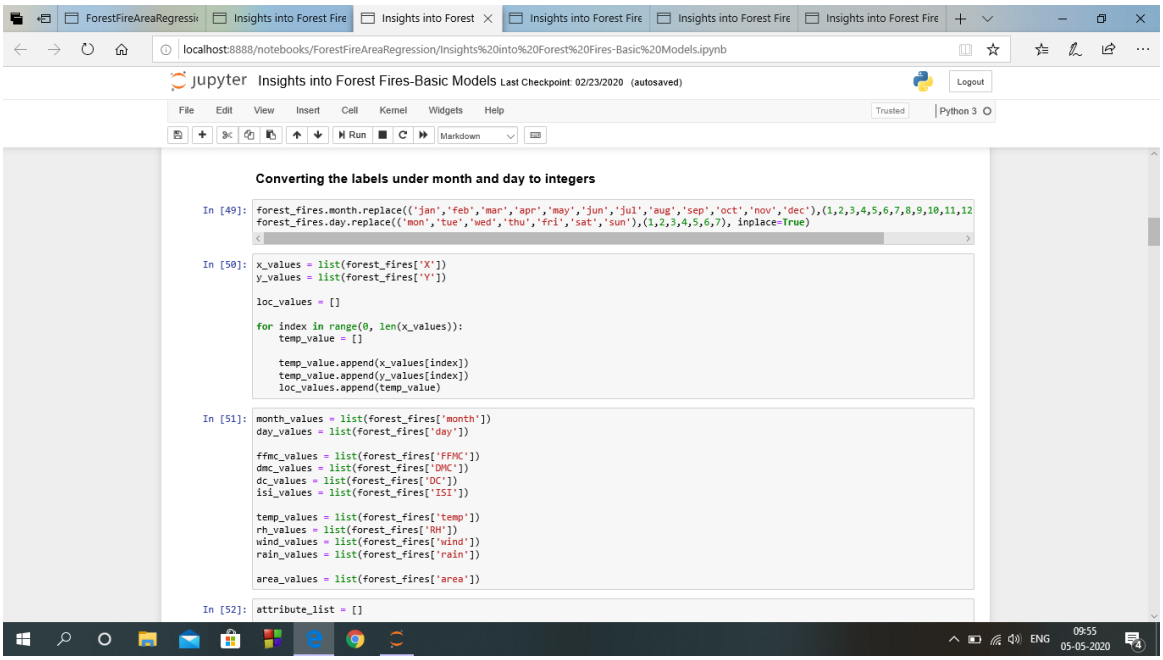

Figure 3.4.7: Conversion of month and days to integers

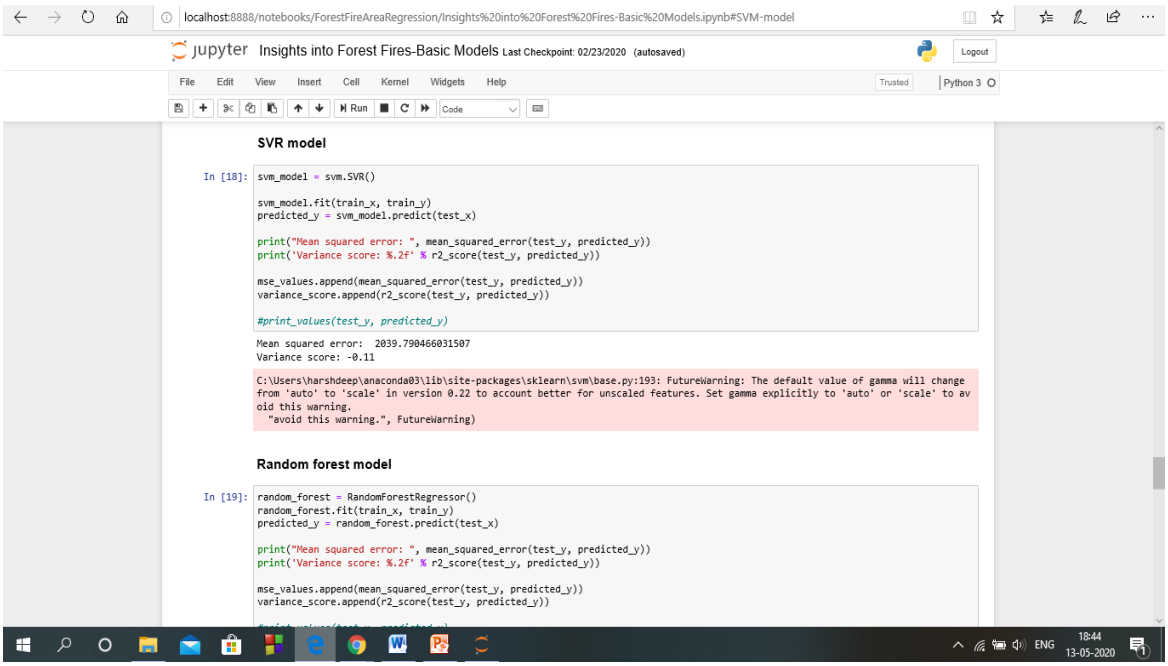

Figure 3.4.8: Using SVR and Random Forest Model

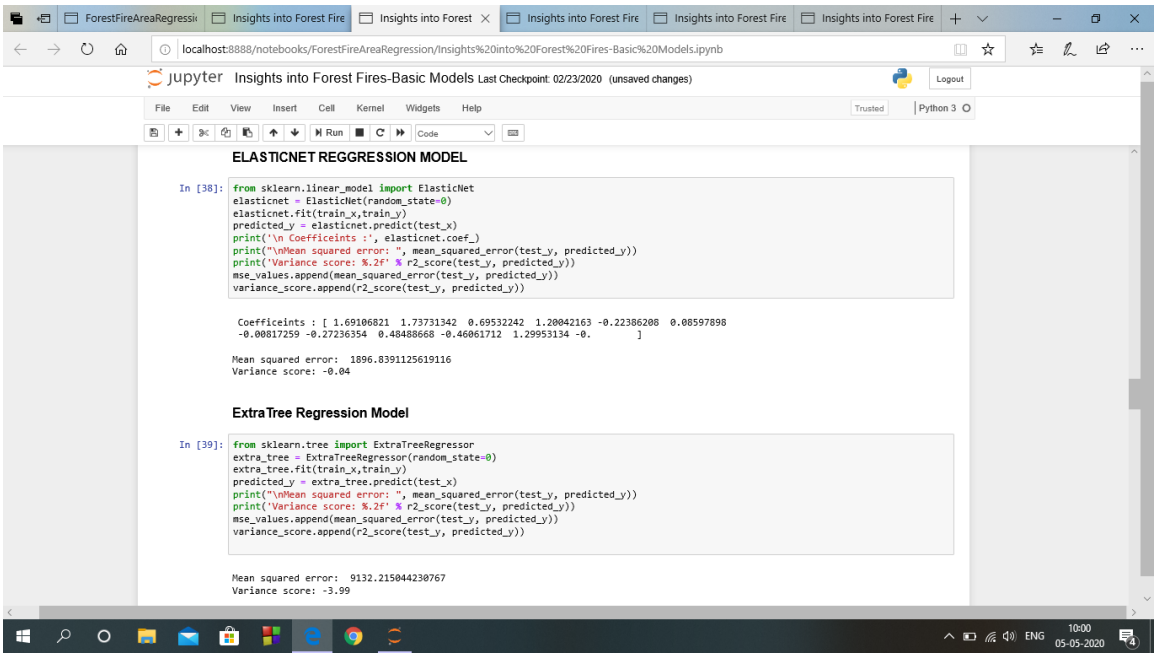

Figure 3.4.9: Using ElasticNet and ExtraTree Regression Models

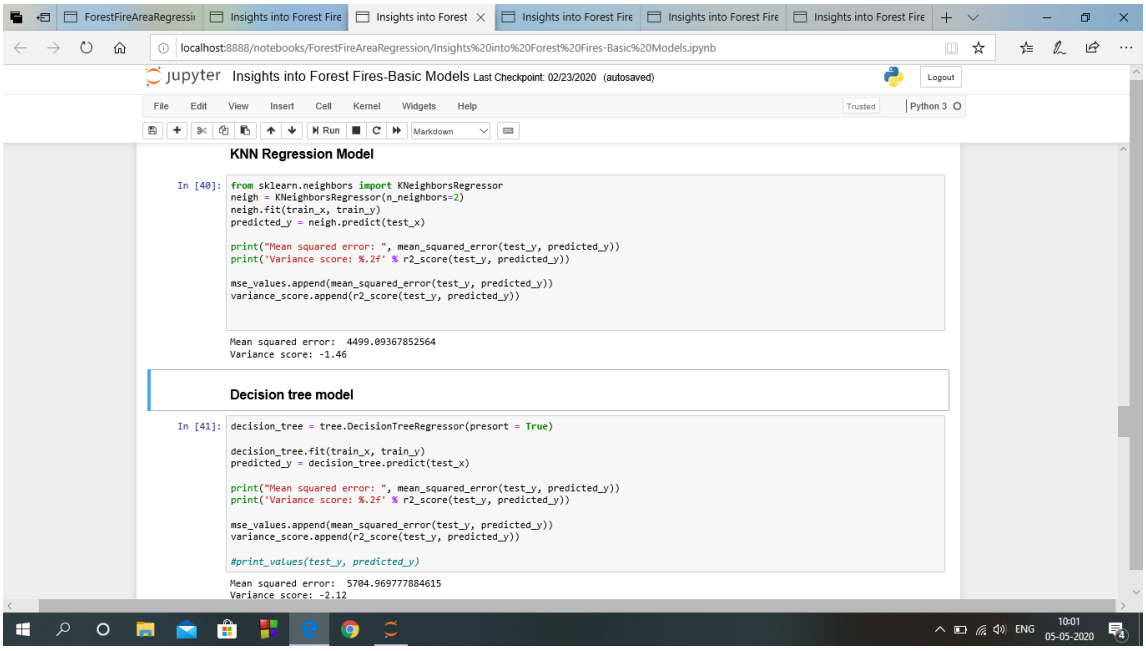

Figure 3.4.10: Using KNN and Decision Tree Model

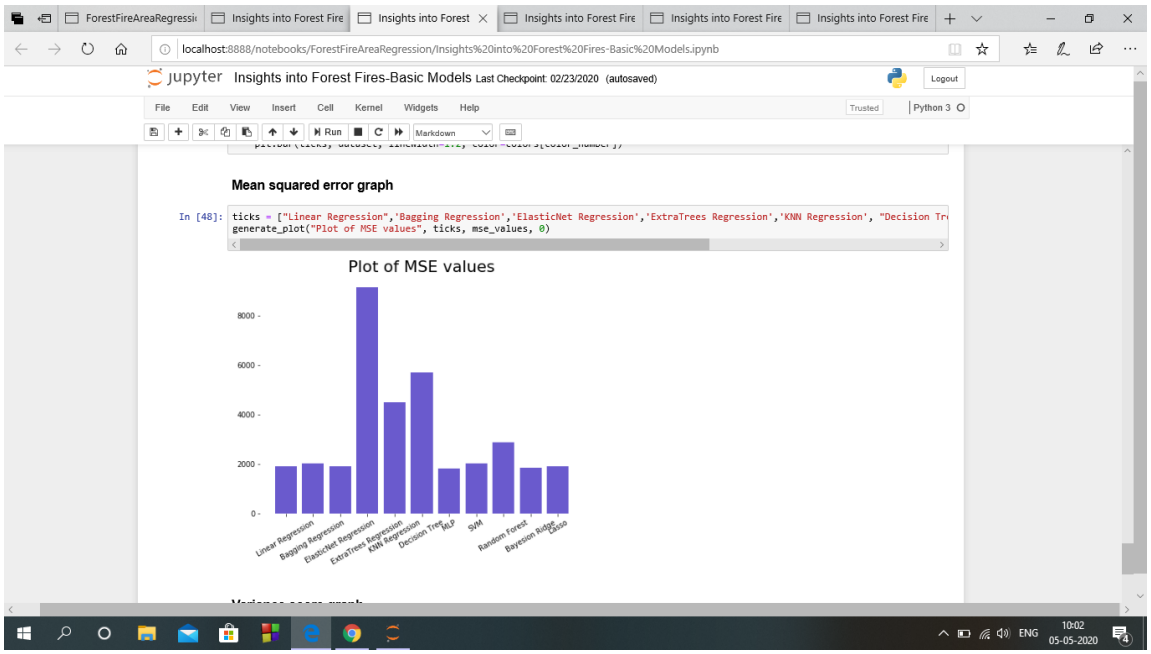

Figure 3.4.11: Plot of MSE values of Basic Models

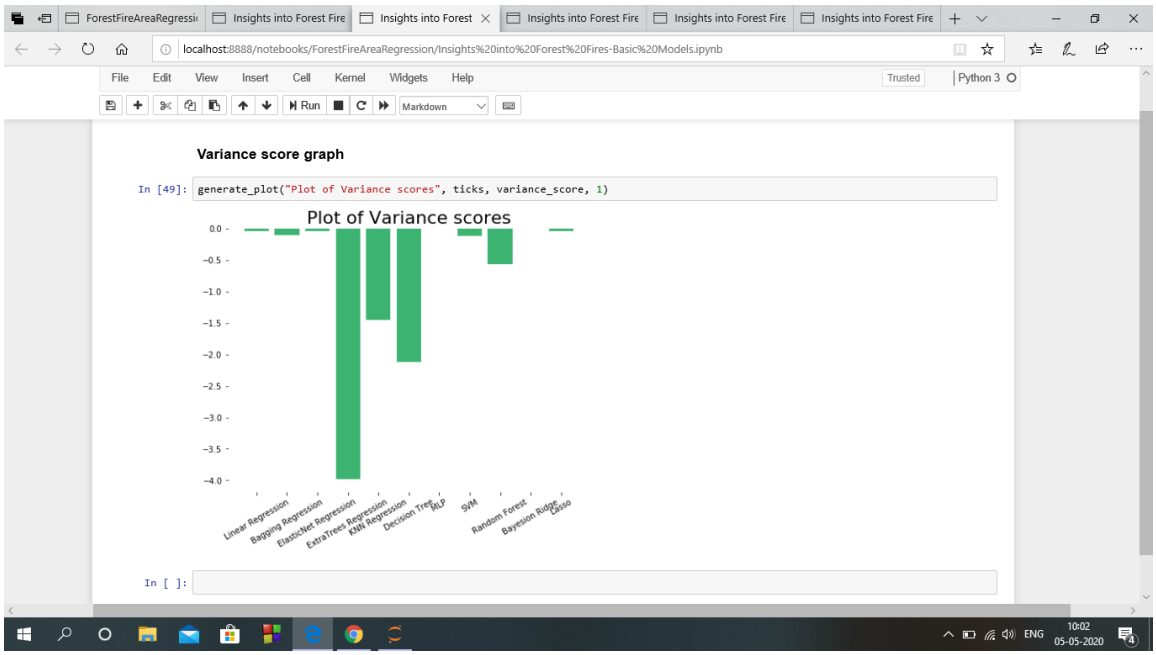

Figure 3.4.12: Plot of Variance Score of Basic Models

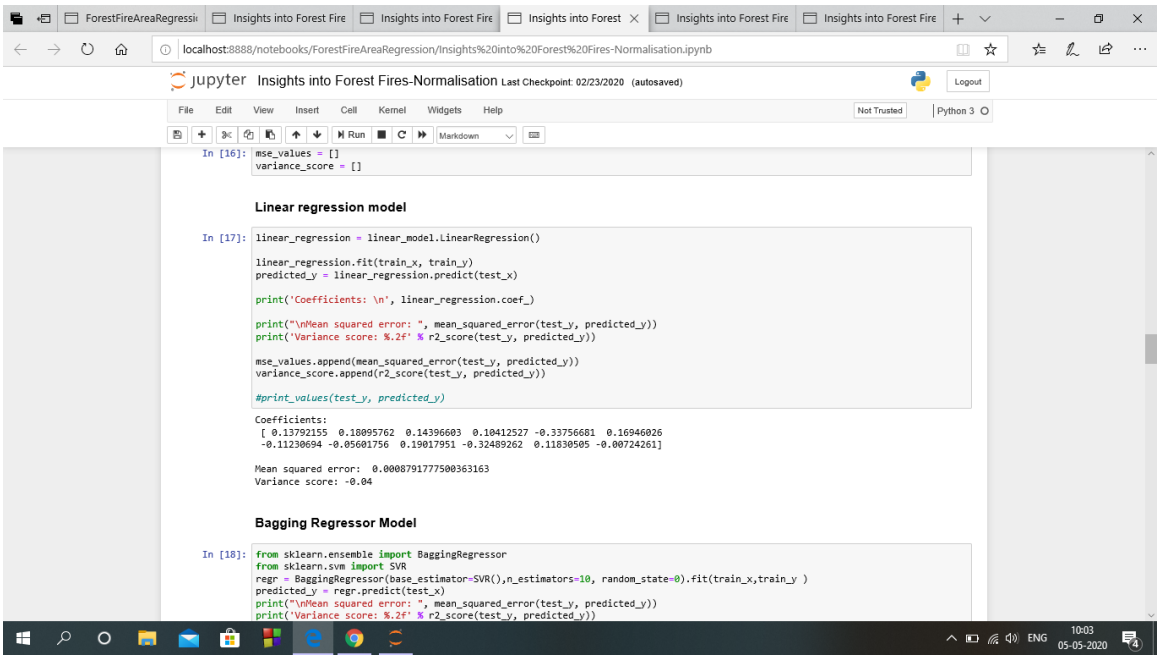

Figure 3.4.13: Linear and Bagging Regression models after Normalization

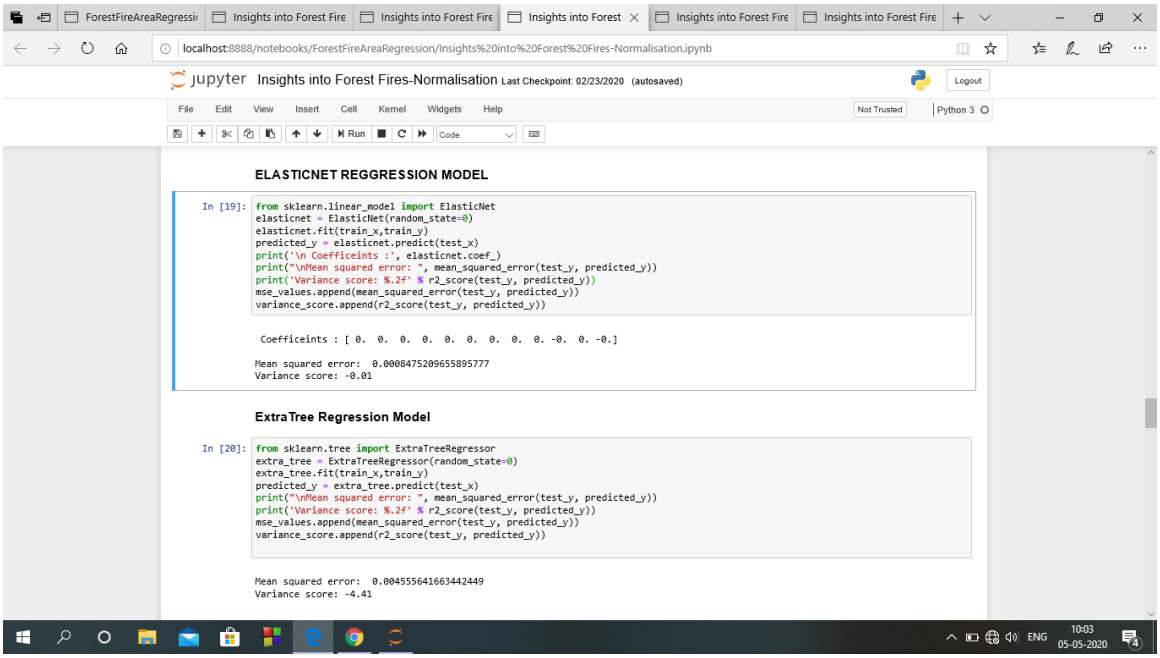

Figure 3.4.14: ElasticNet and ExtraTree Regression models after Normalization

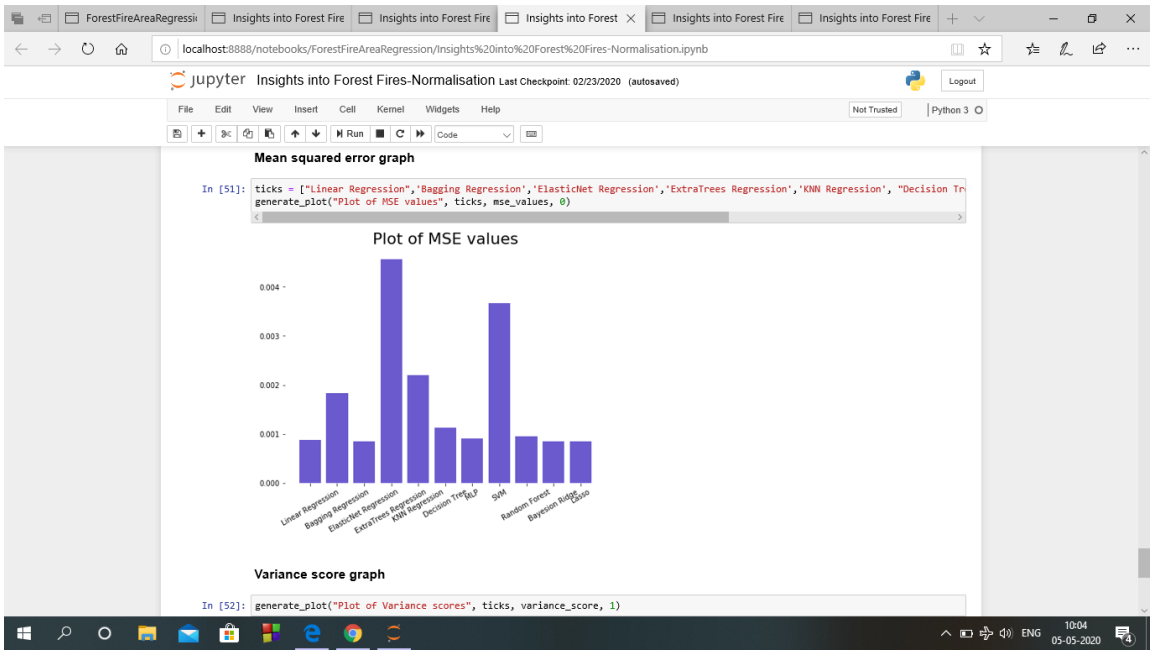

Figure 3.4.15: Plot of MSE values after Normalization

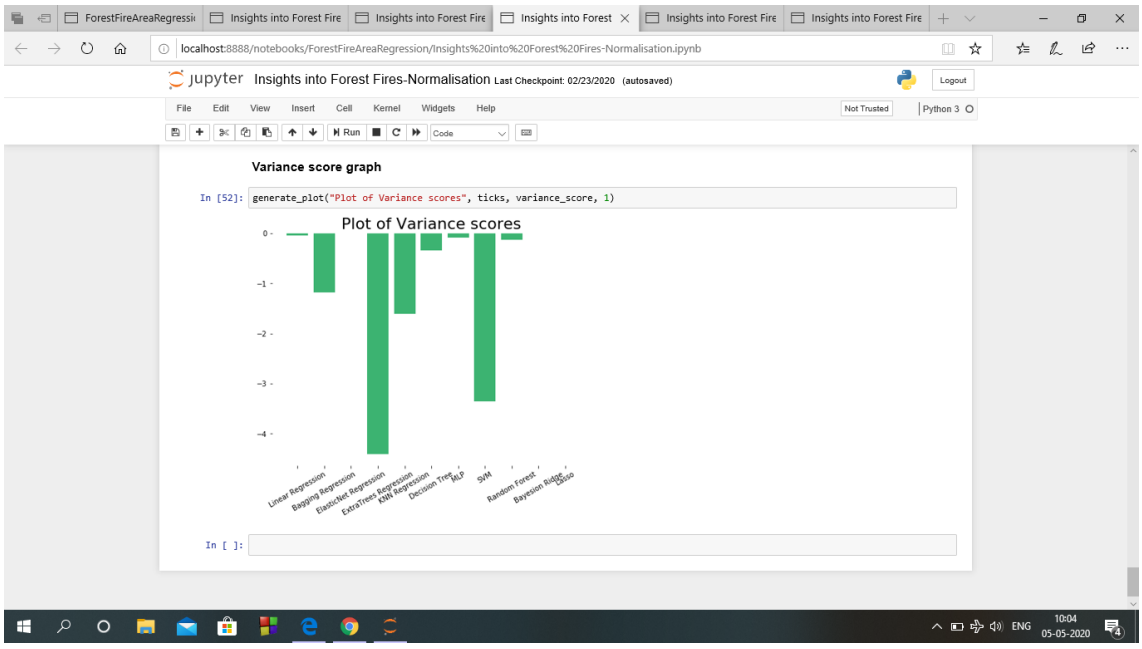

Figure 3.4.16: Plot of Variance Scores after Normalization

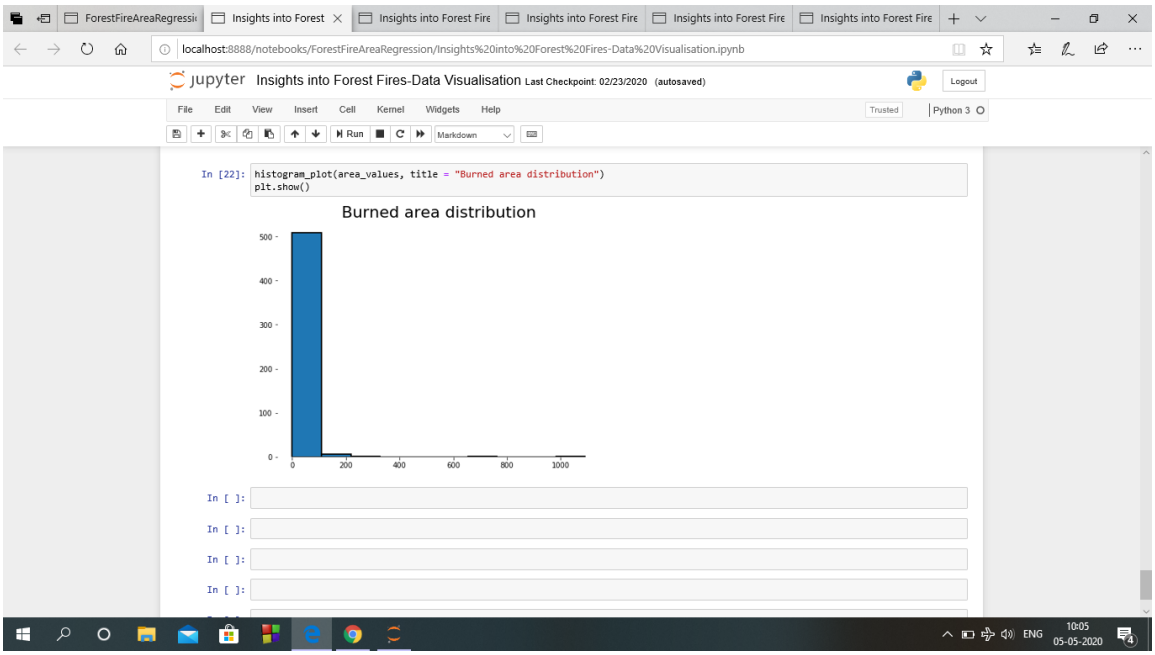

Figure 3.4.17: Plot of Burned Area Distribution

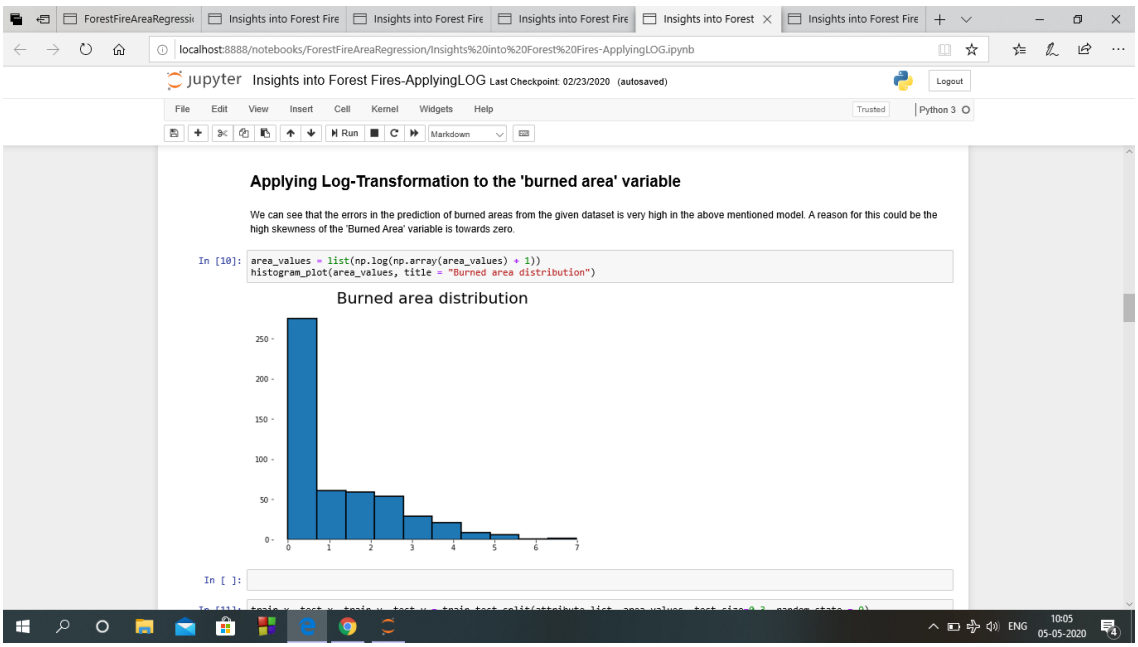

Figure 3.4.18: Plot of Burned Area Distribution after Log-Transformation

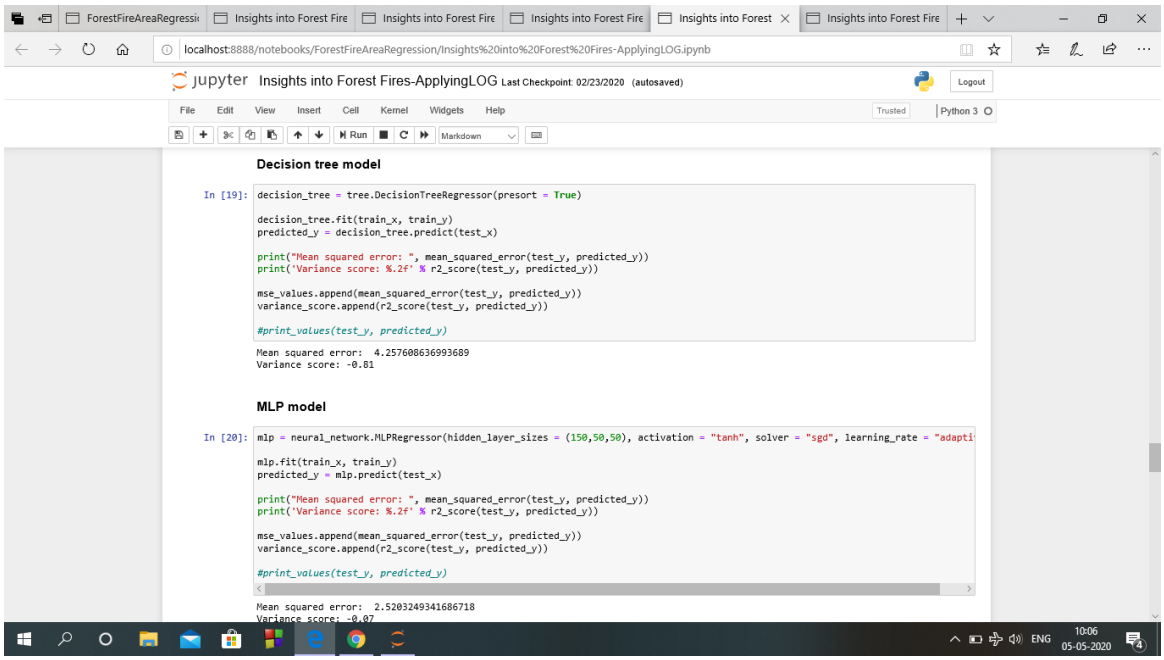

Figure 3.4.19: Decision Tree and MLP models after Log-Transformation

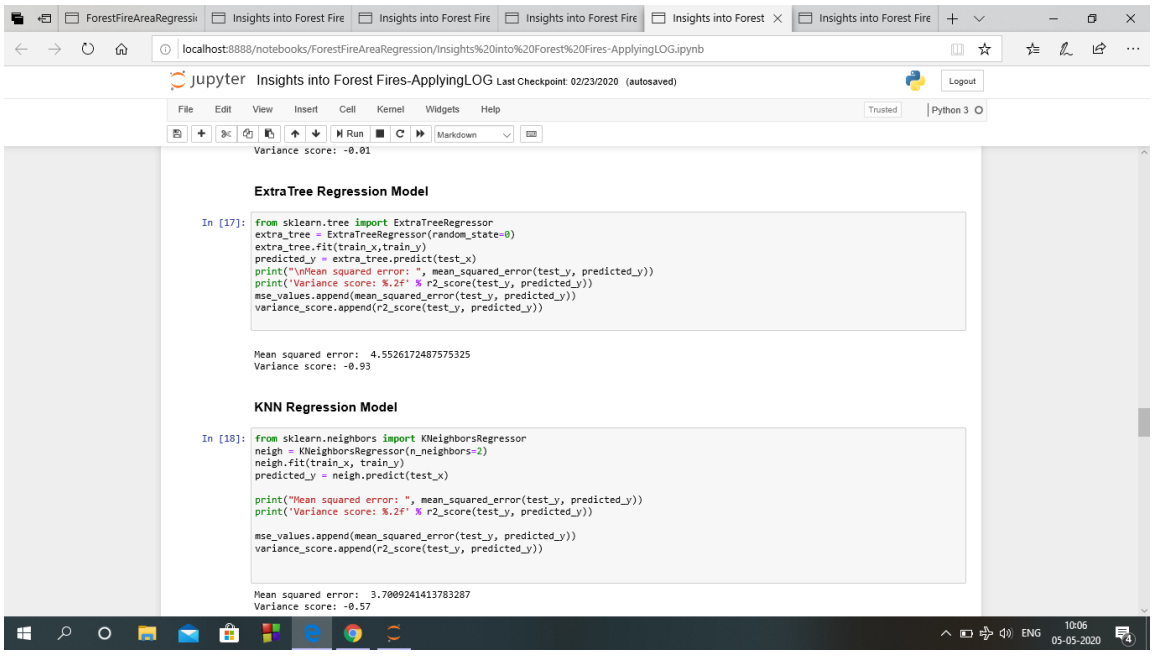

Figure 3.4.20: ElasticNet and ExtraTree Regression models after Log-Transformation

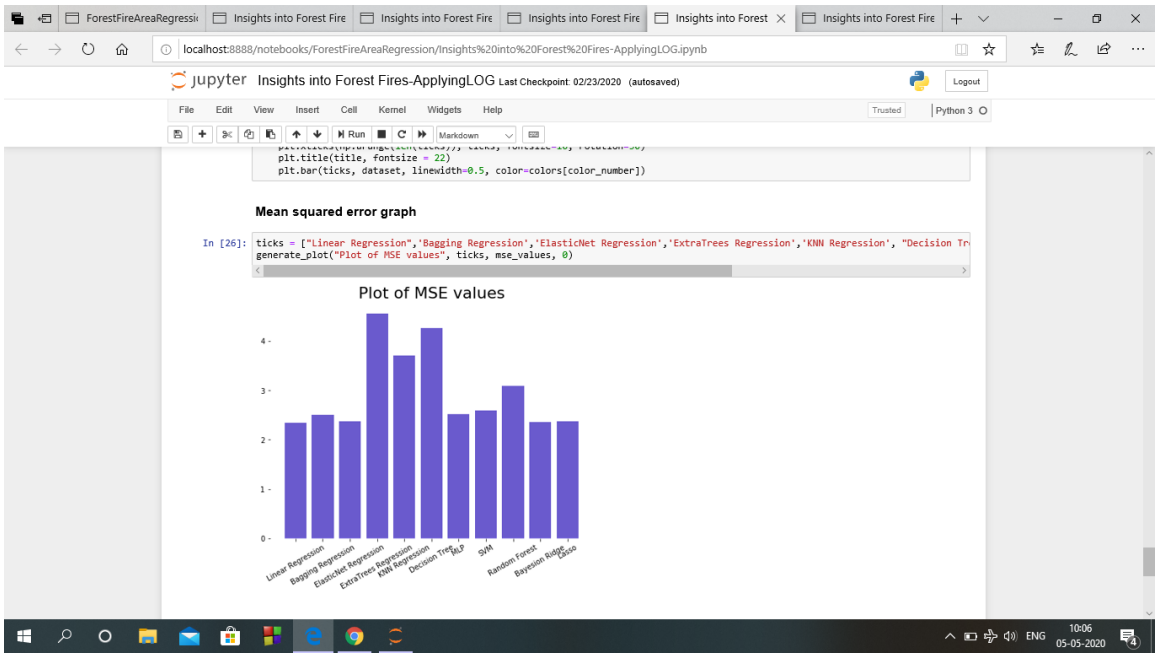

Figure 3.4.21: Plot of MSE values after Log-Transformation

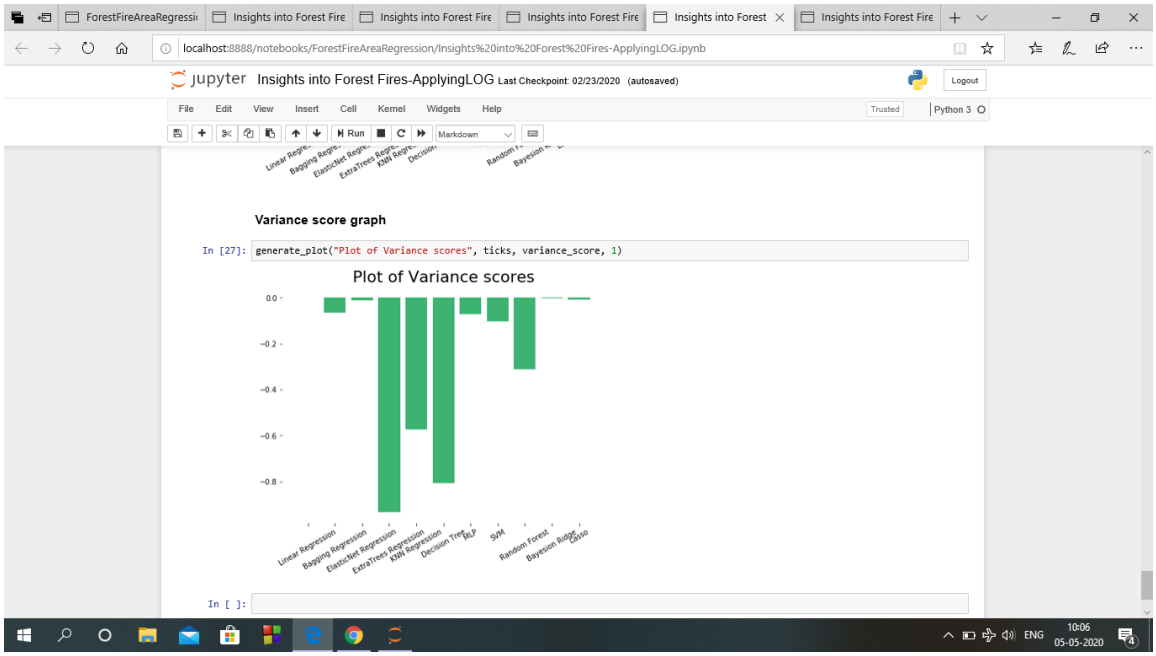

Figure 3.4.22: Plot of Variance Scores after Log-Transformation

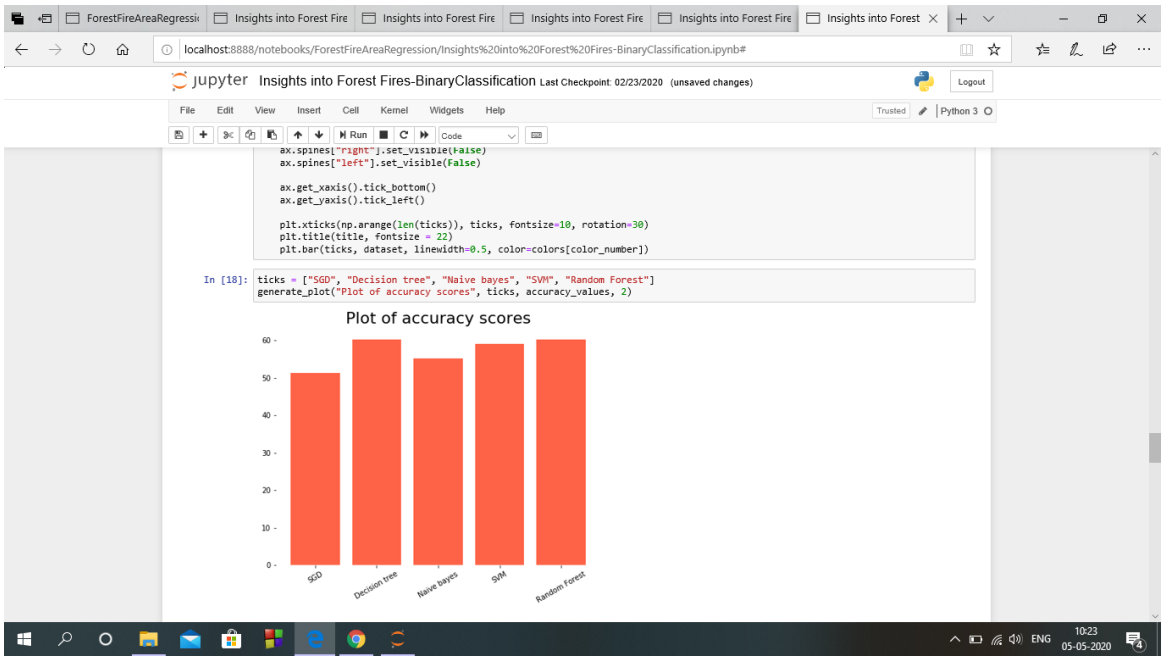

Figure 3.4.23: Plot of Accuracy Scores after Binary Classification

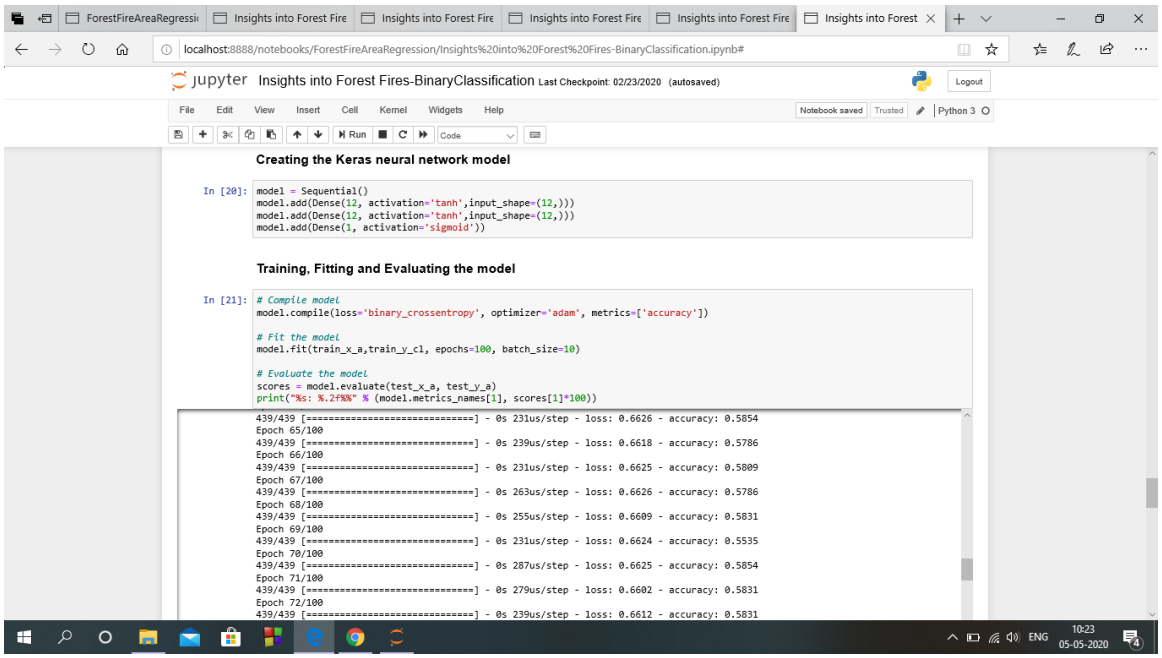

Figure 3.4.24: Creating, Training, Fitting and Evaluating Neural Network model

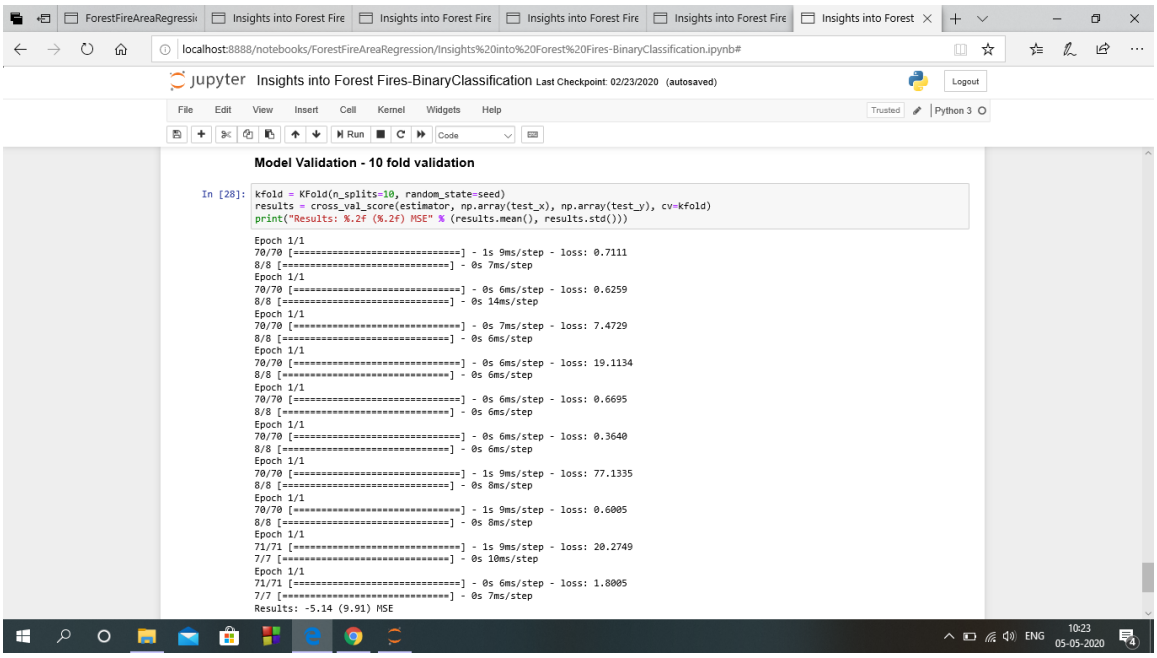

Figure 3.4.25: Performing k-fold Cross validation (here k=10)

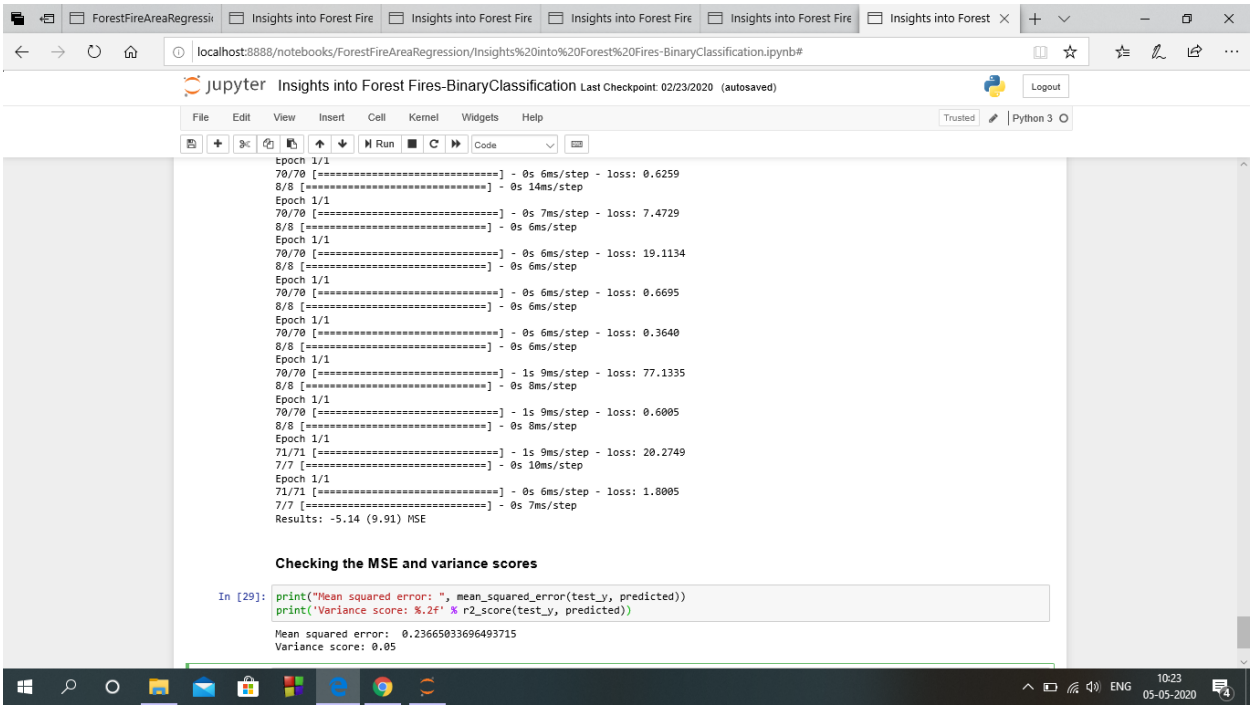

Figure 3.4.26: MSE and Variance Scores after 10-fold Cross Validation

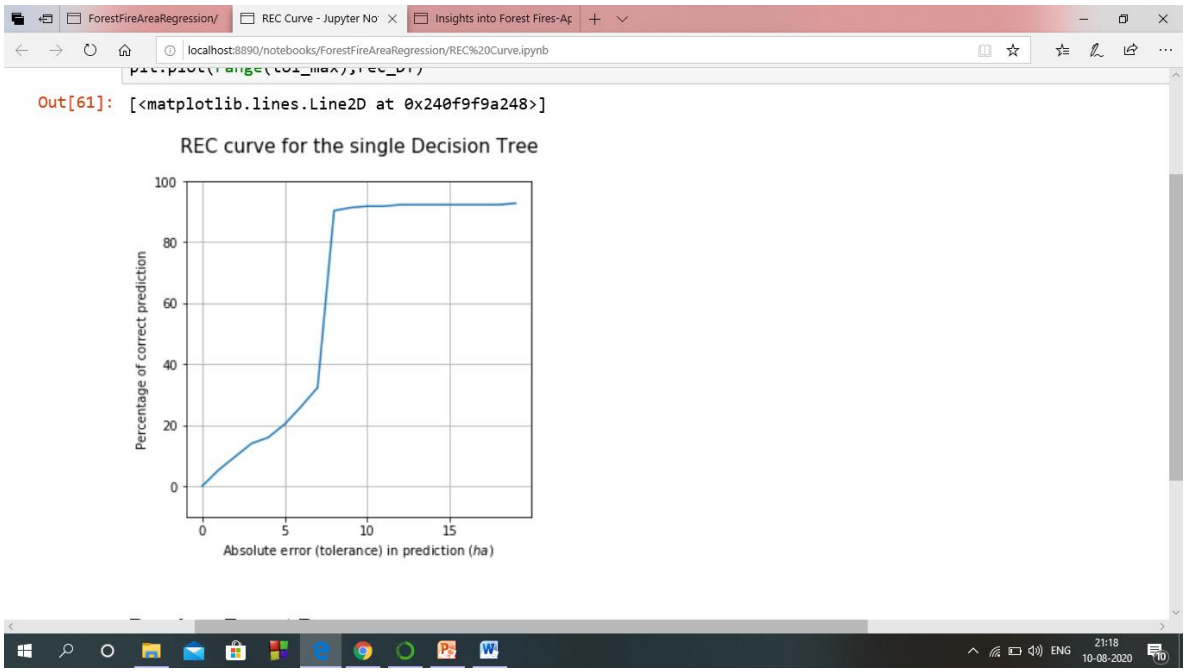

Figure 3.4.27: REC Curve for Decision Tree Model

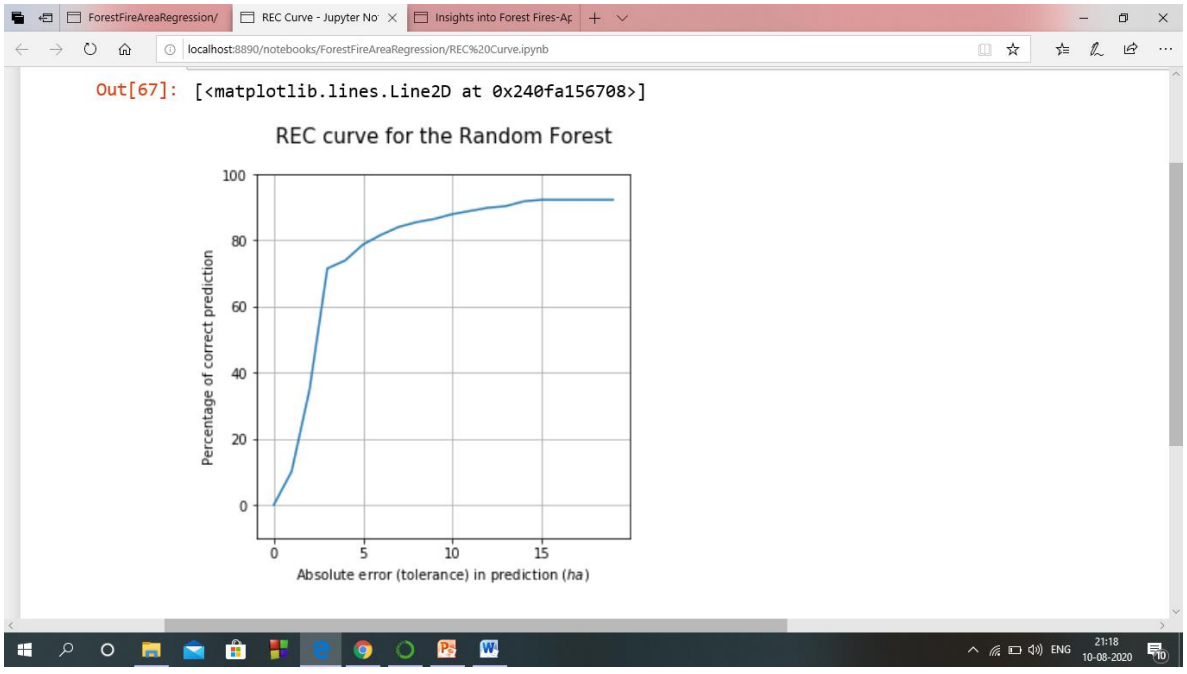

Figure 3.4.28: REC Curve for Random Forest Model

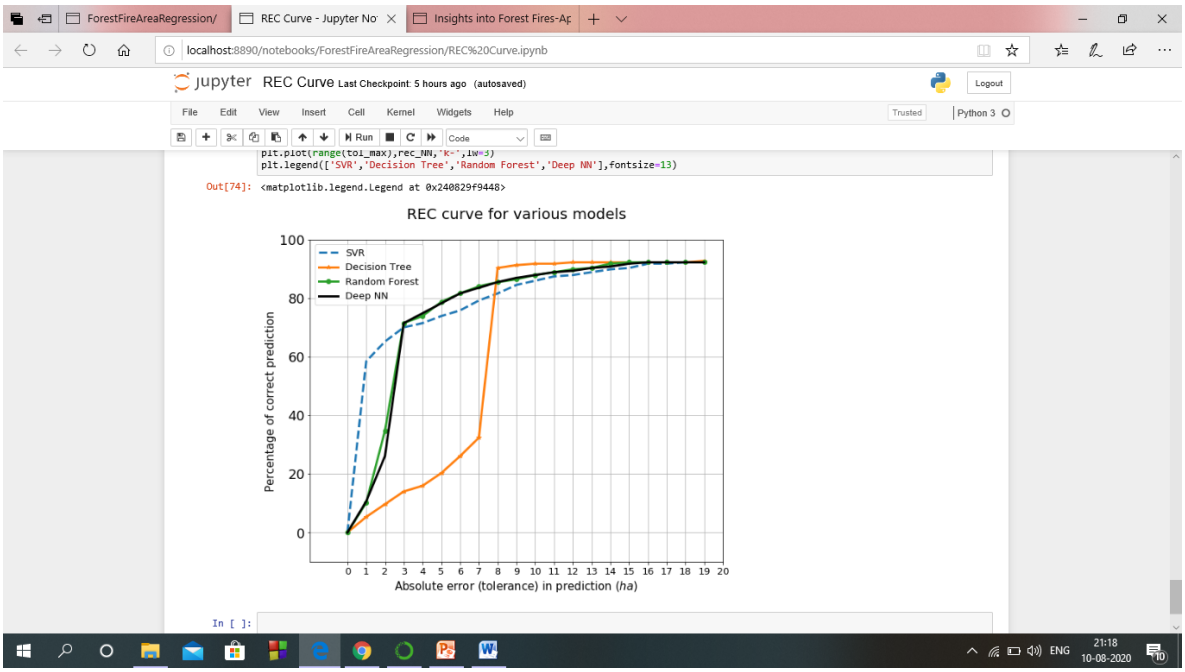

Figure 3.4.29: REC Curves for Various Models

Chapter IV: CODING AND TESTING

4.1) Coding

The goal of the coding phase is to translate the design. The aim of this phase is to implement the design in the best possible manner. Well, the known code can reduce the testing and maintenance effort. Simplicity and clarity should be strived for during the coding. Below are the technologies and libraries that were used for the development of this project.

4.1.1) Technologies Used

Below is a description of the technologies that have been used for the development of this project.

4.1.1.1) Jupyter Notebook

The Jupyter Notebook is an interactive computing environment that enables users to author notebook documents that include: - Live code - Interactive widgets - Plots - Narrative text - Equations - Images - Video.

The Jupyter Notebook combines three components:

1. The notebook web application: An interactive web application for writing and running code interactively and authoring notebook documents.
2. Kernels: Separate processes started by the notebook web application that runs users' code in a given language and returns output back to the notebook web application. The kernel also handles things like computations for interactive widgets, tab completion and introspection.
3. Notebook documents: Self-contained documents that contain a representation of all content visible in the notebook web application, including inputs and outputs of the computations, narrative text, equations, images, and rich media representations of objects. Each notebook document has its own kernel.

The notebook web application enables users to:

1. Edit code in the browser, with automatic syntax highlighting, indentation, and tab completion/introspection.
2. Run code from the browser, with the results of computations attached to the code which generated them.
3. See the results of computations with rich media representations, such as HTML, LaTeX, PNG, SVG, PDF, etc.
4. Create and use interactive JavaScript widgets, which bind interactive user interface controls and visualizations to reactive kernel side computations.

5. Author narrative text using the Markdown markup language.
6. Include mathematical equations using LaTeX syntax in Markdown, which are rendered in-browser by MathJax.

4.1.1.2) Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code that would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems. Using third-party tools, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, allowing the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter. C Python, the reference implementation of Python, is free and open-source software and has a community-based development model, as do nearly all of its alternative implementations. C Python is managed by the non-profit Python Software Foundation.

Features of Python: Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by Metaprogramming and by magic methods). Many other paradigms are supported using extensions, including design by contract and logic programming. Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The design of Python offers some support for functional programming in the Lisp tradition. The language has map(), reduce() and filter() functions; comprehensions for lists, dictionaries, and sets. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. The default kernel of Jupyter Notebook runs Python code. The notebook provides a simple way for users to pick which of these kernels is used for a given notebook[6].

4.1.2) Libraries Used

4.1.2.1) Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. The function returns a Matplotlib container object with all bars.

To install Matplotlib:
python -m pip install -U pip
python -m pip install -U matplotlib

4.1.2.2) Numpy

It stands for numerical python. NumPy is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. It contains among other things:
-A powerful N-dimensional array object
-Sophisticated (broadcasting) functions
-Tools for integrating C/C++ and Fortran code
-Useful linear algebra, Fourier transform, and random number capabilities.

To install Python NumPy, go to your command prompt and type:
pip install numpy.

4.1.2.3) Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. A fast and efficient Data Frame object for data manipulation with integrated indexing, Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format; Intelligent data alignment and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form; Flexible reshaping and pivoting of data sets; Intelligent label-based slicing, fancy indexing, and sub setting of large data sets; Columns can be inserted and deleted from data structures for size mutability, Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets; High performance merging and joining of data sets; Hierarchical axis indexing provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure.

To install Pandas:
pip install pandas

4.1.2.4) Sklearn Libraries

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

To install Sklearn:
pip install sklearn

4.1.2.5) Keras Libraries

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides. Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

To install Keras:
from tensorflow import keras
from tensorflow.keras import layers

4.2) Testing

Testing is a critical aspect of Software Quality Assurance and represents the ultimate review of specification, design, and coding. Testing is a process of executing a program with the intent of finding an error. Testing has many benefits and one of the most important ones is cost-effectiveness. Having testing done can save money in the long run. Software development consists of many stages and if bugs are caught in the earlier stages it costs much less to fix them. That is why it’s important to get testing done as soon as possible.

4.2.1) Unit Testing

Unit Testing focuses first on the modules in the proposed system to locate errors. This enables to detect errors in the coding and logic that are contained within that module alone. Those resulting from the interaction between modules are initially avoided. In unit testing step, each module has to be checked separately.

4.2.2) Integration Testing

Upon completion of unit testing, the units or modules are to be integrated which gives rise to integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

4.2.3) System Testing

Integration testing and system testing is performed once the development of the complete application is done. Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software.

Chapter V: IMPLEMENTATION

5.1) Implementation Activities

Implementation was divided into various modules starting with the installation of Python, its various libraries and Jupyter Notebook that has been used in the project for executing the code. Here, we have used Anaconda as text editor and used Anaconda Navigator to ease the running of our code.

5.1.1) Installing Python

For Mac OS or Linux, Python should already be installed on your computer by default. If not, you can download the latest version by visiting the Python home page, at http://www.python.org where you will also find loads of documentation and other useful information. Windows users can also download Python at this website. We've used Python 3.6. Once you've downloaded the setup, the installation wizard will help you through the installation.

Figure 5.1.1: Installing Python

5.1.2) Installing Jupyter Notebook

Jupyter runs code in many programming languages, Python is a requirement.
We have used Anaconda distribution to install Python and Jupyter. Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.
Use the following installation steps:

Download Anaconda. We recommend downloading Anaconda's latest Python 3 version.

Install the version of Anaconda which you downloaded, following the instructions on the download page.
Jupyter Notebook has been installed
To run the notebook:
jupyter notebook

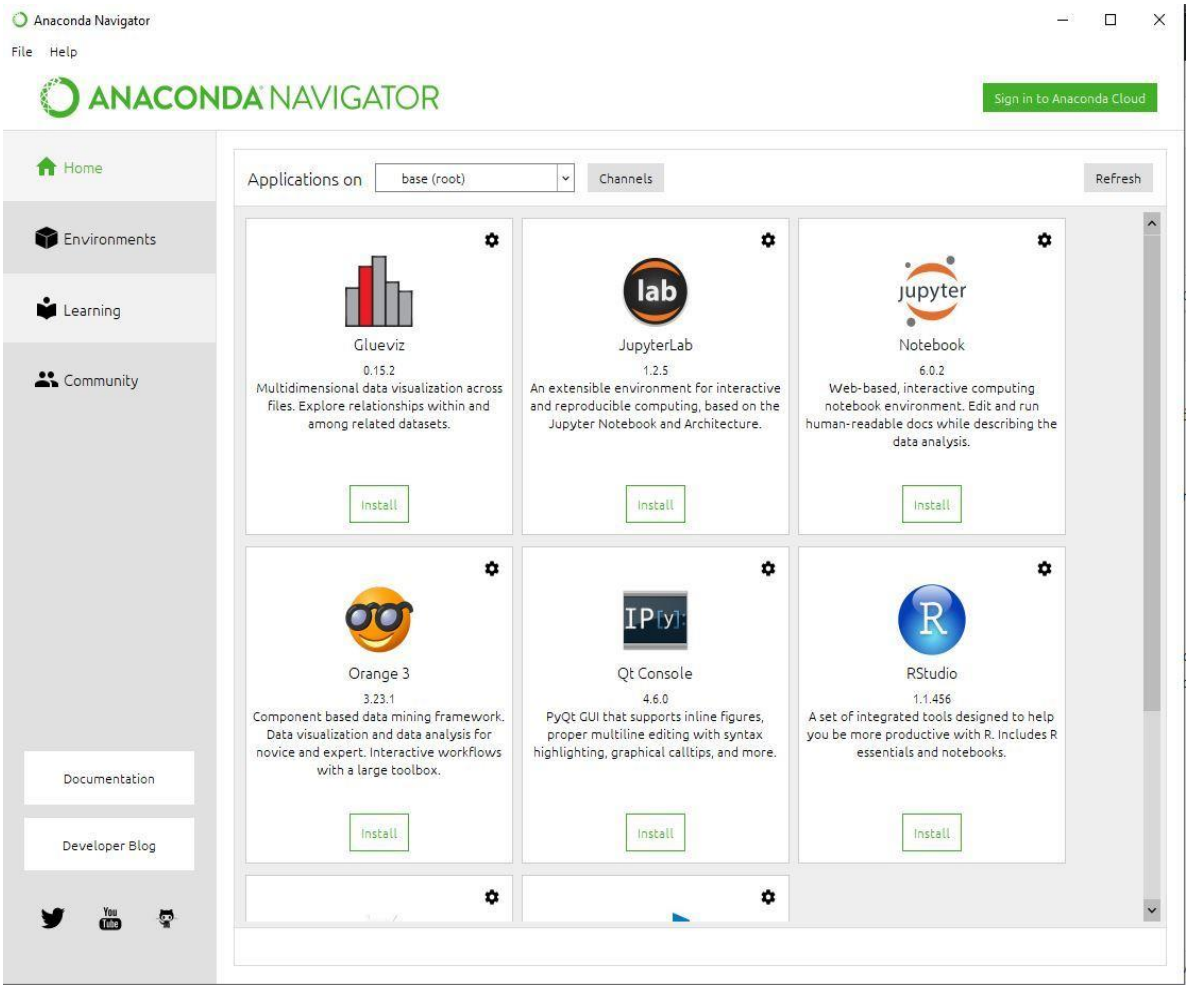

Figure 5.1.2: Installing Jupyter Notebook

5.2) Documentation (User's Manual)

This system is developed using Anaconda as IDE which consists of many inbuilt features to develop the system effectively. To use this system, we need to install Python 3.3 or above. There are some other dependencies which should be installed for the smooth functioning of the project like the tensorflow.

5.2.1) How to execute the System?

To execute the system, one needs to open the Anaconda IDE terminal and then in the Home menu click on Jupyter Notebook to launch it. Then localhost window will appear, and we need to use it to create new project and then execute them in the localhost itself, or we can type in the following command(s) after navigating to the directory where the code of our Prediction of Forest Fires exists.
Python prediction-of-forest-fires

5.2.2) How to enter the data?

If the collection of information is initially taken from a forest fire report, the data entry form on the computer screen should resemble the handwritten form as much as possible to facilitate this data entry. The recording of the data can be carried out at various hierarchical levels.

The recorded data must also be checked, for example, in one of the following ways:

1. Make a double data entry by two operators and to compare the two files
2. Print data and compare them with the original forms
3. In the case of a detected error or missing data, additional information is requested from the operator

Latitude of incident	41,90163999
Longitude of incident	-7,01175806
Current temperature	11,5 °C
Current wind	5,8 km/h
Current relative humidity	92 %
Rainfall in last 3 hours	0 mm/m²
Expected burning area in ha	16 ha

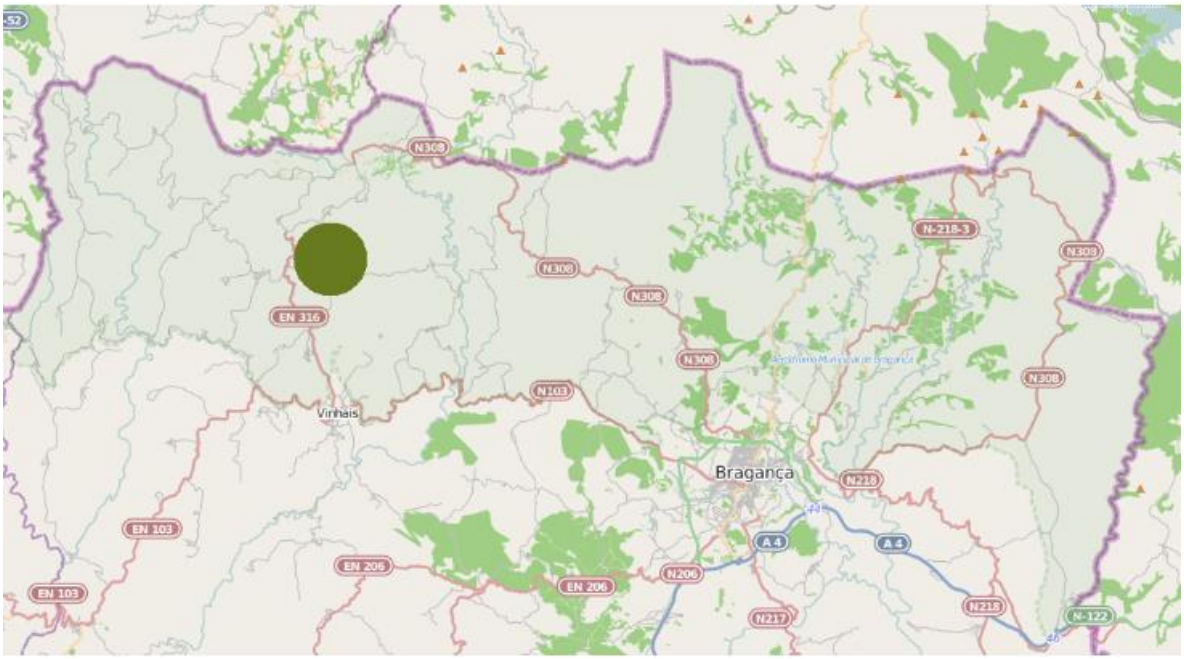

Figure 5.2.2: Data Entry

The database was collected by the Polytechnic Institute, containing several weather observations (e.g. wind speed) that were recorded with a 30 minute period by a meteorological station located in the center.

The two databases were stored in tens of individual spreadsheets, under distinct formats, and a substantial manual effort was performed to integrate them into a single dataset with a total of 1551 entries [7].The map of the Montesinho natural park Table 5 shows a description of the selected data features. The first four rows denote the spatial and temporal attributes. Only two geographic features were included, the X and Y axis values where the fire occurred, since the type of vegetation presented a low quality (i.e. more than 80% of the values were missing).

The following table shows a description of the selected data features:

Attribute	*Description*
X	x-axis coordinate
Y	y-axis coordinate
Month	Month of the year(January to December)
Day	Day of the Week (Monday to Sunday)
FFMC	Fine Fuel Moisture Code
DMC	Duff Moisture Code
DC	Drought Code
ISI	Initial Spread Index
Temperature	Outside Temperature in Celsius scale
RH	Outside Relative Humidity (in %)
Wind	Outside Wind Speed (in km/h)
Rain	Outside Rain (in $mm/m^{2)}$
Area	Total Burned Area (in ha)

Table 5

We selected the month and day of the week temporal variables. Average monthly weather conditions are quite distinct; while the day of the week could also influence forest fires (e.g. work days v/s weekend) since most fires have a human cause. Next comes the four FWI components that are affected directly by the weather conditions. The BUI and FWI were discarded since they are dependent of the previous values. The exception is the rain variable, which denotes the accumulated precipitation within the previous 30 minutes [8].

Regarding the present dataset, there are 447 samples with a zero value. As previously stated, all entries denote fire occurrences and zero value means that an area lower than 1ha/100 = 100m2 was burned. To reduce skewness and improve symmetry, the logarithm function $y = \ln(x + 1)$, which is a common transformation that tends to improve regression results for right-skewed targets, was applied to the area attribute.

5.2.3) How to process the data? (processing details)

The information collected in the database allows studying a multitude of relations between certain fire characteristics. The analyses carried out are intended to clarify:
− When and where fires start;
− Their number and burned area;
− Fire fighting techniques;
− Similarities between fires (dates, places, causes, etc.);
− Exceptional fires.

The analysis of the results can take the following shape-:

1. Data in relation over time

To analyze the evolution with time:
− Number of fires and burned areas
− Example: Numbers of fires, total, maximum, and average burned areas by fire, by year, by month, day or hour
− Fire causes.

2. Data in relation to the place

To analyze the relations between the place of ignition (area, province, community):
− The number of fires Example: Number of fires per region
− Burned surface example: Total, maximum, and average burned surfaces by fire and by region.
− Evolution over time of the fire number and burned surfaces, Example: Number of fires and total, maximum, and average burned surfaces by fire, area, year
− Fire causes, Example: Fire causes by area

3. Data in relation to elapsed time

Relation analysis of burned surfaces and Intervention times, class of intervention time, suppression time, class of extinction time.

Chapter VI: MAINTENANCE FEATURES

Project maintenance is a matter of practicing some very simple values throughout the course of your project. It means being very intentional about tracking your progress toward milestones and goals, instead of assuming everything will happen as planned.

1. Ensure Quality
   Quality is, “the degree to which a set of inherent characteristics fulfil requirement".
   The type or size of project we are managing, the time to define the quality criteria for our current work so that our team members understand what it is, and how to reach and improve upon it.

2. Commit to quality
   A project commitment to quality must come from the top and be reinforced repeatedly. As a project manager or leader, commit to quality, share the commitment with your staff and think about how you will handle the any conflict between your stated objective and an attractive cost-saving, short cut that compromises quality.

3. Stick to the Project Requirements
   Once we have defined the quality criteria and project requirements, balance continual project improvements with gold-plate requirements. Adding features, the customer did not request increases the potential for delays, and higher cost. Project Managers drive improvements and project quality but beware of out-of-scope extras creeping in.

4. Perform Quality Assurance
   Execute the quality management plan using the standards and processes defined in the project blueprint. Perform a quality audit to evaluate how well the team is following the plan and meeting the customer’s expectations.

5. Follow the project processes
   Follow the processes and tasks contained in our project blueprint. If we identify a more efficient way to do something, adding this into the blueprint will continually improve the processes.

6. Checking the maintenance
   Checking the maintenance of the project after a particular interval of time. Maintaining consistent project quality takes commitment, focus and courage, and following these steps we will not only help ourselves maintain high levels of quality in projects but also continuously improve the consistency of quality of the project.

Chapter VII: Advantages and Limitations of Developed System

7.1) Advantages of Developed System

1. Predicting the source and spread of forest fires could have considerable benefits for human health and life, the economy and the environment. This could help identify areas with higher risk for example, with limited resources; the authorities could choose to focus on monitoring specific areas.

2. Since traditional human surveillance is expensive and affected by subjective factors, there has been an emphasis to develop automatic solutions. So, this developed system is helpful.

3. The prediction of burned area of forest fires is particularly useful for improving firefighting resource management. It can be used for prioritizing targets for air tankers and ground crews.

4. The use of low cost meteorological data for the prediction of forest fires will reduce the overall cost of the fire-fighting system. It will be more cost effective than satellite based sensors and infrared or smoke sensors.

7.2) Limitations

1. Fire spread by spotting (flying embers or firebrands) is not accounted: This includes laboratory or theoretically based rate of fire spread models and fire modeling systems that rely upon such models. In situations where this form of fire propagation is influential or a dominant mechanism, forecasts or predictions of fire spread are likely to result in underestimates.

2. Vertical and horizontal fire whirlwinds are not modeled: Amongst the other factors we also pointed out that the influence of fire whirls and similar extreme, fire-induced vortices on the rate of spread or growth of a free burning wild land fire are not considered. While fire whirls have been documented to travel in excess of 2.5 km from the main fire. An understanding of how these events will specifically affect rate of fire spread for example is lacking.

3. Accuracy of model input data: Predictive models must be sensitive to those parameters known to readily affect fire behavior, such as wind speed, dead fuel moisture and slope steepness, amongst others. If these input data are not known accurately enough or the user fails to appreciate the spatial and temporal variability of input data, model output can in turn be significantly in error.

4. The greatest challenge a fire model user faces in a predictive situation is the accurate estimation of representative input values. A deterministic approach for fire behavior prediction assumes best estimates of input conditions to represent the fire environment.

5. Internal accuracy of model relationships: Wildfires, being unpredictable as to their timing and location and often occurring in remote locations, are seldom amenable subjects for conventional instrumentation and measurement as afforded by a prescribed fire or experimental fire although there is the odd exception. Furthermore, some aspects of wildfire behavior, such as observations of maximum spot fire distance and associated influences, are difficult to monitor and thus to precisely document.

Chapter VIII: Conclusions and Suggestions for further work

8.1) Conclusion

1. Our findings highlight the importance of monitoring the changes that take place in the South. We now have a well-developed and on-going forest inventory system to monitor forest conditions and uses. Southern forest assessments have long depended on these data. The U.S. Forest Assessment System used for the Futures Project could not have been built without the inventory data; indeed, the maturation and usefulness of regional and national forecasting will depend on a steady updating of these core data. Our key findings indicate that forest inventories will remain critical tools for monitoring changes, but also that monitoring certain socioeconomic elements—including land use dynamics, investment, and land transactions—is also needed. Forecasts highlight areas where changes might be concentrated in the short-to-medium runs and therefore where intensified monitoring activities might be useful. The findings also highlight the need for new and continuously measured land use data that effectively monitor changes that are driven by urbanization.

2. Our scope does not include prescriptions for policymaking, yet policy is an important factor in the forecasts. The empirical models and future forecasts are based on existing policy across several domains, and an important assumption of the forecasts is that the future develops without change to the policy setting.

3. All forecasts of the future are likely to be wrong to one degree or another.

8.2) Suggestions for further work

1. This work opens room for the development of automatic tools for fire management support. Indeed in the future we intend to test the proposed approach by using an online learning environment as part of a Fire Management system
2. Another interesting possibility would be the use of weather forecasts in order to build proactive responses.
3. Finally since large fires are rare events outlier detection techniques can be applied.

References

[1] Forest Fire Research in California: An Annotated Bibliography, 1923-1961 (Wilson & Nilsson) 1962 (Pacific Southwest Forest and Range Experiment Station Miscellaneous Paper No. 75)

[2] Bibliographies on Chaparral and the Fire Ecology of Other Mediterranean Systems (Keeley) 1988 (California Water Resources Center Report #69)

[3]https://www.jstor.org/stable/43490292?Search=yes&resultItemClick=true&searchText=prediction&searchText=of&searchText=forest&searchText=fires&searchUri=%2Faction%2FdoBasicSearch%3FQuery%3Dprediction%2Bof%2Bforest%2Bfires%2B%26amp%3Bacc%3Don%26amp%3Bwc%3Don%26amp%3Bfc%3Doff%26amp%3Bgroup%3Dnone&ab_segments=0%2Fbasic_search%2Fcontrol&refreqid=search%3A81b8b2634b9c74777898ec6d45f5bd08&seq=1#metadata_info_tab_contents

[4] https://scialert.net/fulltextmobile/?doi=jas.2009.578.582

[5] Fire: A Summary of Literature in the United States from the Mid-1920s to 1966 (Cushwa) 1968 Includes 823 references categorized by broad topic.

[6] Python Documentation

[7] https://www.knime.com/knime-applications/forest-fire-prediction

[8] Effects of Fire on Forests: A Bibliography (US Forest Service Library) 1938 Includes 605 references arranged by subject.

APPENDIX

Source Code

```
import matplotlib.pyplot as plt
import math
import numpy as np
import pandas as pd
import random

# importing sklearn libraries
from sklearn import neural_network, linear_model, preprocessing, svm, tree
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB

# importing keras libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor

import warnings

# supressing the warning on the usage of Linear Regression model
warnings.filterwarnings(action="ignore", module="scipy", message="^internal gelsd")

#Loading the dataset
forest_fires = pd.read_csv('forest_fires.csv')
forest_fires

#Converting the labels under month and day to integers
forest_fires.month.replace(('jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec'),(1,2
,3,4,5,6,7,8,9,10,11,12), inplace=True)
forest_fires.day.replace(('mon','tue','wed','thu','fri','sat','sun'),(1,2,3,4,5,6,7), inplace=True)

#Statistical analysis of dataset
forest_fires.describe()

#Corelation analysis for the dataset
forest_fires.corr()
```

```
#Extracting features from the dataset
x_values = list(forest_fires['X'])
y_values = list(forest_fires['Y'])
loc_values = []
for index in range(0, len(x_values)):
temp_value = []
temp_value.append(x_values[index])
temp_value.append(y_values[index])
loc_values.append(temp_value)
month_values = list(forest_fires['month'])
day_values = list(forest_fires['day'])
ffmc_values = list(forest_fires['FFMC'])
dmc_values = list(forest_fires['DMC'])
dc_values = list(forest_fires['DC'])
isi_values = list(forest_fires['ISI'])
temp_values = list(forest_fires['temp'])
rh_values = list(forest_fires['RH'])
wind_values = list(forest_fires['wind'])
rain_values = list(forest_fires['rain'])
area_values = list(forest_fires['area'])
attribute_list = []
for index in range(0, len(x_values)):
temp_list = []
temp_list.append(x_values[index])
temp_list.append(y_values[index])
temp_list.append(month_values[index])
temp_list.append(day_values[index])
temp_list.append(ffmc_values[index])
temp_list.append(dmc_values[index])
temp_list.append(dc_values[index])
temp_list.append(isi_values[index])
temp_list.append(temp_values[index])
temp_list.append(rh_values[index])
temp_list.append(wind_values[index])
temp_list.append(rain_values[index])
attribute_list.append(temp_list)

#Counting the instances of location points in dataset
def count_points(x_points, y_points, scaling_factor):
count_array = []
for index in range(0, len(x_points)):
temp_value = [x_points[index], y_points[index]]
count = 0
for value in loc_values:
```

```
if(temp_value == value):
count = count + 1
count_array.append(count * scaling_factor )
return count_array

#Histogram plotting function for dataset
def histogram_plot(dataset, title):
plt.figure(figsize=(8, 6))
ax = plt.subplot()
ax.spines["top"].set_visible(False)
ax.spines["bottom"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.get_xaxis().tick_bottom()
ax.get_yaxis().tick_left()
plt.title(title, fontsize = 22)
plt.hist(dataset, edgecolor='black', linewidth=1.2)

#Scatter plot for the locations
plt.figure(figsize=(8, 6))
ax = plt.subplot()
ax.spines["top"].set_visible(False)
ax.spines["bottom"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.get_xaxis().tick_bottom()
ax.get_yaxis().tick_left()
plt.title("Fire location plot", fontsize = 22)
plt.scatter(x_values, y_values, s = count_points(x_values, y_values, 25), alpha = 0.3)
plt.show()
histogram_plot(ffmc_values, title = "FFMC distribution")
plt.show()
histogram_plot(dmc_values, title = "DMC distribution")
plt.show()
histogram_plot(dc_values, title = "DC distribution")
plt.show()
histogram_plot(isi_values, title = "ISI distribution")
plt.show()
histogram_plot(temp_values, title = "Temperature distribution")
plt.show()
histogram_plot(rh_values, title = "RH distribution")
plt.show()
histogram_plot(wind_values, title = "Wind distribution")
plt.show()
histogram_plot(rain_values, title = "Rain distribution")
```

```
plt.show()
histogram_plot(area_values, title = "Burned area distribution")
plt.show()

#Percentage of dataset with 'burned area' > 0
total_count = 0
positive_data_count = 0
for value in area_values:
if(value > 0):
positive_data_count = positive_data_count + 1
total_count = total_count + 1
print("The number of data records with 'burned area' > 0 are " + str(positive_data_count) +
" and the total number of records are " + str(total_count) + ".")
print("The percentage value is " + str(positive_data_count/total_count * 100) + ".")

#Plotting the distribution of values for the dataset
histogram_plot(month_values, title = "Month distribution")
plt.show()
histogram_plot(day_values, title = "Day distribution")
plt.show()


##Gaining insights with learning models

#Spilliting the available data/Setting the initial parameters
train_x, test_x, train_y, test_y = train_test_split(attribute_list, area_values, test_size=0.3,
random_state = 9)
mse_values = []
variance_score = []

#Printing the actual vs predicted values
def print_values(test, predicted):
print("The actual output and the predicted output are:")
for value in range(0, len(predicted_y)):
print('%.4f' % test_y[value], " ", '%.4f' % predicted_y[value])

#Linear regression model
linear_regression = linear_model.LinearRegression()
linear_regression.fit(train_x, train_y)
predicted_y = linear_regression.predict(test_x)
print('Coefficients: \n', linear_regression.coef_)
print("\nMean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
```

```
#print_values(test_y, predicted_y)
#Decision tree model
decision_tree = tree.DecisionTreeRegressor(presort = True)
decision_tree.fit(train_x, train_y)
predicted_y = decision_tree.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#MLP model
mlp = neural_network.MLPRegressor(hidden_layer_sizes = (150,50,50), activation = "tanh", solver = "sgd", learning_rate = "adaptive")
mlp.fit(train_x, train_y)
predicted_y = mlp.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#SVR model
svm_model = svm.SVR()
svm_model.fit(train_x, train_y)
predicted_y = svm_model.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#Random forest model
random_forest = RandomForestRegressor()
random_forest.fit(train_x, train_y)
predicted_y = random_forest.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#Bayesian ridge model
bayesian_ridge = linear_model.BayesianRidge()
bayesian_ridge.fit(train_x, train_y)
```

```
predicted_y = bayesian_ridge.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#Lasso model
lasso_model = linear_model.Lasso()
lasso_model.fit(train_x, train_y)
predicted_y = lasso_model.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#Mean squared error graph
ticks = ["Linear Regression", "Decision Tree", "MLP", "SVM", "Random Forest", "Bayesion Ridge", "Lasso"]
generate_plot("Plot of MSE values", ticks, mse_values, 0)

#Variance score graph
generate_plot("Plot of Variance scores", ticks, variance_score, 1)

#Applying Log-Transformation to the 'burned area' variable
area_values = list(np.log(np.array(area_values) + 1))
histogram_plot(area_values, title = "Burned area distribution")



##Applying learning models on the processed data

#Setting the initial parameters
mse_values = []
variance_score = []

#Decision tree model
decision_tree = tree.DecisionTreeRegressor(presort = True)
decision_tree.fit(train_x, train_y)
predicted_y = decision_tree.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
```

```
#print_values(test_y, predicted_y)

#MLP model
mlp = neural_network.MLPRegressor(hidden_layer_sizes = (150,30,50), activation = "tanh", solver = "sgd", learning_rate = "adaptive")
mlp.fit(train_x, train_y)
predicted_y = mlp.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#Normalisation of all data
n_x_values = preprocessing.normalize([x_values])[0]
n_y_values = preprocessing.normalize([y_values])[0]
n_month_values = preprocessing.normalize([month_values])[0]
n_day_values = preprocessing.normalize([day_values])[0]
n_ffmc_values = preprocessing.normalize([ffmc_values])[0]
n_dmc_values = preprocessing.normalize([dmc_values])[0]
n_dc_values = preprocessing.normalize([dc_values])[0]
n_isi_values = preprocessing.normalize([isi_values])[0]
n_temp_values = preprocessing.normalize([temp_values])[0]
n_rh_values = preprocessing.normalize([rh_values])[0]
n_wind_values = preprocessing.normalize([wind_values])[0]
n_rain_values = preprocessing.normalize([rain_values])[0]
n_area_values = preprocessing.normalize([area_values])[0]
n_attribute_list = []

##Applying learning models on the normalised data

#Setting the initial parameters
mse_values = []
variance_score = []

#Spilliting the available data
train_x, test_x, train_y, test_y = train_test_split(n_attribute_list, n_area_values, test_size=0.3, random_state = 9)

#SVM model
svm_model = svm.SVR()
svm_model.fit(train_x, train_y)
predicted_y = svm_model.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
```

```
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)

#Lasso model
lasso_model = linear_model.Lasso()
lasso_model.fit(train_x, train_y)
predicted_y = lasso_model.predict(test_x)
print("Mean squared error: ", mean_squared_error(test_y, predicted_y))
print('Variance score: %.2f' % r2_score(test_y, predicted_y))
mse_values.append(mean_squared_error(test_y, predicted_y))
variance_score.append(r2_score(test_y, predicted_y))
#print_values(test_y, predicted_y)


##Visualising the results

#Mean squared error graph
ticks = ["Linear Regression", "Decision Tree", "MLP", "SVM", "Random Forest", "Bayesion Ridge", "Lasso"]
generate_plot("Plot of MSE values", ticks, mse_values, 0)

#Variance score graph
generate_plot("Plot of Variance scores", ticks, variance_score, 1)

#Converting the target values to binary classes
binary_area_values = []
count = 0
for value in area_values:
if(value == 0):
binary_area_values.append(0)
else:
binary_area_values.append(1)


##Gaining insights with classification algorithm

#SGD model
sgd = linear_model.SGDClassifier()
sgd.fit(train_x, train_y)
predicted_y = sgd.predict(test_x)
print("The predicted values are:", predicted_y)
print("The accuracy score is " + str(accuracy_score(test_y, predicted_y) * 100) + ".")
accuracy_values.append(accuracy_score(test_y, predicted_y) * 100)
```

```
#Decision tree model
decision_tree = tree.DecisionTreeClassifier()
decision_tree.fit(train_x, train_y)
predicted_y = decision_tree.predict(test_x)
print("The predicted values are:", predicted_y)
print("The accuracy score is " + str(accuracy_score(test_y, predicted_y) * 100) + ".")
accuracy_values.append(accuracy_score(test_y, predicted_y) * 100)

#Naive bayes model
naive_bayes = GaussianNB()
naive_bayes.fit(train_x, train_y)
predicted_y = naive_bayes.predict(test_x)
print("The predicted values are:", predicted_y)
print("The accuracy score is " + str(accuracy_score(test_y, predicted_y) * 100) + ".")
accuracy_values.append(accuracy_score(test_y, predicted_y) * 100)

#SVM model
svm_model = svm.SVC(kernel='linear', gamma=100)
svm_model.fit(train_x, train_y)
predicted_y = svm_model.predict(test_x)
print("The predicted values are:", predicted_y)
print("The accuracy score is " + str(accuracy_score(test_y, predicted_y) * 100) + ".")
accuracy_values.append(accuracy_score(test_y, predicted_y) * 100)

##Visualising the results

#Variance score graph
ticks = ["SGD", "Decision tree", "Naive bayes", "SVM", "Random Forest"]
generate_plot("Plot of accuracy scores", ticks, accuracy_values, 2)

#Artificial Neural Network - Implementation
train_x_a = np.array(train_x)
test_x_a = np.array(test_x)
test_y_a = np.array(test_y)
train_y_temp = np.array(train_y)
train_y_cl = []

for i in range(len(train_y)):
if(train_y[i]>0):
train_y_cl.append(1)
else:
train_y_cl.append(0)
train_y_cl = np.array(train_y_cl)
```

```
print("Length of Training data is : "+str(len(train_x_a))+" and Test data is : " + str(len(test_x_a)))

#Creating the Keras neural network model
model = Sequential()
model.add(Dense(12, activation='tanh',input_shape=(12,)))
model.add(Dense(12, activation='tanh',input_shape=(12,)))
model.add(Dense(1, activation='sigmoid'))


##Training, Fitting and Evaluating the model

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
model.fit(train_x_a,train_y_cl, epochs=100, batch_size=10)

# Evaluate the model
scores = model.evaluate(test_x_a, test_y_a)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

##Artificial Neural Network - Regression Model

#Defining base model
def baseline_model():
# create model
model = Sequential()
model.add(Dense(12, input_dim=12, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
# compile model
model.compile(loss='mean_squared_error', optimizer='adam')
return model

#Creating sequntial model
model = Sequential()
model.add(Dense(12, input_dim=12, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))


#Compiling & training the model

# compile model
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
# train model
history = model.fit(np.array(train_x), np.array(train_y), epochs=150, batch_size=5, verbose=1)

#Make Predictions
predicted = model.predict( np.array(test_x),batch_size=None, verbose=0, steps=1)

#Evaluate model with standardized dataset
estimator = KerasRegressor(build_fn=baseline_model, nb_epoch=100, batch_size=5, verbose=1)
seed = 12
np.random.seed(seed)

#Model Validation - 10 fold validation
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(estimator, np.array(test_x), np.array(test_y), cv=kfold)
print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))

#Checking the MSE and variance scores
print("Mean squared error: ", mean_squared_error(test_y, predicted))
print('Variance score: %.2f' % r2_score(test_y, predicted))


##Regression Error Characteristic (REC) estimation
def rec(m,n,tol):
if type(m)!='numpy.ndarray':
m=np.array(m)
if type(n)!='numpy.ndarray':
n=np.array(n)
l=m.size
percent = 0
for i in range(l):
if np.abs(10*m[i]-10*n[i])<=tol:
percent+=1
return 100*(percent/l)

#Define the max tolerance limit for REC curve x-axis
#For this problem this represents the absolute value of error in the prediction of the outcome i.e. area burned
tol_max=20

#Support Vector Regressor (SVR)
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

```
scaler = StandardScaler()


#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']}
grid_SVR = GridSearchCV(SVR(),param_grid,refit=True,verbose=0,cv=5)
grid_SVR.fit(scaler.fit_transform(X_train),scaler.fit_transform(y_train))
print("Best parameters obtained by Grid Search:",grid_SVR.best_params_)
Best parameters obtained by Grid Search: {'C': 0.01, 'epsilon': 1, 'kernel': 'rbf'}
a=grid_SVR.predict(X_test)
print("RMSE for Support Vector Regression:",np.sqrt(np.mean((y_test-a)**2)))
plt.xlabel("Actual area burned")
plt.ylabel("Error")
plt.grid(True)
plt.scatter(10*(y_test),10(a)-10*(y_test))

#Plotting REC Curve
rec_SVR=[]
for i in range(tol_max):
rec_SVR.append(rec(a,y_test,i))
plt.figure(figsize=(5,5))
plt.title("REC curve for the Support Vector Regressor\n",fontsize=15)
plt.xlabel("Absolute error (tolerance) in prediction ($ha$)")
plt.ylabel("Percentage of correct prediction")
plt.xticks([i*5 for i in range(tol_max+1)])
plt.ylim(-10,100)
plt.yticks([i*20 for i in range(6)])
plt.grid(True)
plt.plot(range(tol_max),rec_SVR)


#Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
tree_model = DecisionTreeRegressor(max_depth=10,criterion='mae')
tree_model.fit(scaler.fit_transform(X_train),scaler.fit_transform(y_train))
DecisionTreeRegressor(criterion='mae', max_depth=10, max_features=None,
max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
presort=False, random_state=None, splitter='best')
a=tree_model.predict(X_test)
print("RMSE for Decision Tree:",np.sqrt(np.mean((y_test-a)**2)))
plt.xlabel("Actual area burned")
plt.ylabel("Error")
plt.grid(True)
```

```
plt.scatter(10*(y_test),10(a)-10*(y_test))
plt.title("Histogram of prediction errors\n",fontsize=18)
plt.xlabel("Prediction error ($ha$)",fontsize=14)
plt.grid(True)
plt.hist(10*(a.reshape(a.size,))-10*(y_test),bins=50)

#Plotting REC Curve
rec_DT=[]
for i in range(tol_max):
rec_DT.append(rec(a,y_test,i))
plt.figure(figsize=(5,5))
plt.title("REC curve for the single Decision Tree\n",fontsize=15)
plt.xlabel("Absolute error (tolerance) in prediction ($ha$)")
plt.ylabel("Percentage of correct prediction")
plt.xticks([i for i in range(0,tol_max+1,5)])
plt.ylim(-10,100)
plt.yticks([i*20 for i in range(6)])
plt.grid(True)
plt.plot(range(tol_max),rec_DT)


#Relative performance of various models (REC curves)
plt.figure(figsize=(10,8))
plt.title("REC curve for various models\n",fontsize=20)
plt.xlabel("Absolute error (tolerance) in prediction ($ha$)",fontsize=15)
plt.ylabel("Percentage of correct prediction",fontsize=15)
plt.xticks([i for i in range(0,tol_max+1,1)],fontsize=13)
plt.ylim(-10,100)
plt.xlim(-2,tol_max)
plt.yticks([i*20 for i in range(6)],fontsize=18)
plt.grid(True)
plt.plot(range(tol_max),rec_SVR,'--',lw=3)
plt.plot(range(tol_max),rec_DT,'*-',lw=3)
plt.plot(range(tol_max),rec_RF,'o-',lw=3)
plt.plot(range(tol_max),rec_NN,'k-',lw=3)
plt.legend(['SVR','Decision Tree','Random Forest','Deep NN'],fontsize=13)
```