CROP YIELD PREDICTION BASED ON WEATHER USING MACHINE LEARNING

A Mini Project-2 report submitted in partial fulfillment of requirements for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

by

B. BHOOMIKA (199X1A0508)

K. KISHORE KUMAR RAJU (199X1A0561)

B. PRIYANKA (199X1A0507)

Under the esteemed guidance of

Sri. P. Praveen Yadav

Assistant Professor

Department of C.S.E.

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

2022 – 2023

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

CERTIFICATE

This is to certify that the Mini Project-2 Work entitled 'Crop yield prediction based on weather using Machine Learning' is a bonafide record of work carried out by

B. BHOOMIKA (199X1A0508)

K. KISHORE KUMAR RAJU (199X1A0561)

B. PRIYANKA (199X1A0507)

Under my guidance and supervision in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Sri. P. Praveen Yadav
Assistant Professor,
Department of C.S.E.,
G. Pulla Reddy Engineering College,
Kurnool.

Dr. N. Kasiviswanath
Professor & Head of the Department,
Department of C.S.E.,
G.Pulla Reddy Engineering College,
Kurnool.

DECLARATION

I hereby declare that the project titled "**CROP YIELD PREDICTION BASED ON WEATHER USING MACHINE LEARNING**" is an authentic work carried out by us as the student of **G. PULLA REDDY ENGINEERING COLLEGE(Autonomous) Kurnool,** during 2022-23 and has not been submitted elsewhere for the award of any degree or diploma in part or in full to any institute.

B.BHOOMIKA
(199X1A0508)

K.KISHORE KUMAR RAJU
(199X1A0561

B.PRIYANKA
(199X1A0507)

ACKNOWLEDGEMENT

We wish to express our deep sense of gratitude to our project guide **Sri. P. Praveen Yadav**, Head & Professor of CSE Department, G. Pulla Reddy Engineering College, for his immaculate guidance, constant encouragement and cooperation which have made possible to bring out this project work.

We are grateful to our project incharge **Sri. P.N.V.S. Pavan Kumar**, Assistant Professor of ECS Department, G.Pulla Reddy Engineering College, for helping us and giving us the required information needed for our project work.

We are thankful to our Head of the Department **Dr. N. Kasiviswanath**, for his whole hearted support and encouragement during the project sessions.

We are grateful to our respected Principal **Dr. B. Sreenivasa Reddy** for providing requisite facilities and helping us in providing such a good environment.

We wish to convey our acknowledgements to all the staff members of the Computer Science and Engineering Department for giving the required information needed for our project work.

Finally, we wish to thank all our friends and well wishers who have helped us directly or indirectly during the course of this project work.

ABSTRACT

India is an Agriculture based economy whose most of the GDP comes from farming. The motivation of this project comes from the increasing suicide rates in farmers which may be due to low harvest in crops. Climate and other environmental changes have become a major threat in the agriculture field.

Machine learning is an essential approach for achieving practical and effective solutions for this problem. Predicting yield of the crop from historical available data like weather, soil, rainfall parameters and historic crop yield. We can achieve this using the machine learning algorithm. In this project, we will make a comparative study of various machine learning algorithms, i.e., ANN, K Nearest Neighbour, Random Forest, SVM and Linear Regression, Random Forest Algorithms and predict a best model for crop yield based on the accuracy.

CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

Agriculture is one of the main sectors to be impacted by different sources like climatic changes, soil attributes, seasonal changes etc., Crop yield prediction is based on various kinds of data collected and extracted by using data mining techniques such as machine learning techniques different sources which are useful for growth of the crop. It is an art of forecasting crop and the quantity of yield in advance i.e., before the harvest actually takes place. Predicting the crop yield can be extremely useful for farmers. If they have an idea of the amount of yield they can expect, they can contract their crop prior to harvest, often securing a more competitive price than if they were to wait until after harvest. The involvement of experts in prediction of crop yield leads to issues like lack of knowledge about natural events, negation of personal perception and fatigue etc. such issues can be to overcome by using the models and decision tools for crop yield prediction. Likewise, industry can benefit from yield predictions by better planning the logistics of their business.

With the growing population, the imminent challenge is to increase food production in order to feed the population. Farmers will have to produce 50 percent of more grains to meet the current growing demand. The following are the atmospheric weather variables which influences the crop production.

- Precipitation
- Temperature
- Atmospheric pressure

Rainfall one of the most important factor influences the vegetation of a place. Temperature is a measure of intensity of heat energy. The range of temperature for maximum growth of most of the agricultural plants is between 15 and 40c. Achieving maximum crop yield at minimum cost is one of the goals of agricultural production. Early detection and management of problems associated with crop yield indicators can help increases yield and subsequent profit. Predictions could be used by crop managers to minimize losses when unfavorable conditions may occur.

Additionally, these predictions could be used to maximize crop prediction when potential exists for favorable growing conditions. The crop yield prediction comprises of mostly all essential parameters that are needed for the better yield of crop. This enhances the classification results of the crop yield. In general, one of the difficulties faced in the prediction process is that most of the essential parameters that are necessary to consider for the accurate prediction are not consider. It reduces the efficiency of the predicted results which in turn leads to lack of proper forecasting of the crop yield. It is also more complex to predict the optimized number of input parameters that are to be considered in the prediction process

1.2 PROBLEM DEFINITION

India is an agricultural country. Yield of each crop depends on its dependent factors. It is very important to predict the yield of a crop to help farmers. Crop Yield Prediction is predicting the yield of a crop in future based on the dependent factors. Crop yield is dependent on factors like rainfall, pressure, temperature and area or the geographical location. Prediction is done using machine learning algorithms using historical data of dependent factors.

Crop Yield Prediction helps farmers to grow a crop which gives more yields based on the algorithmic prediction. Thus it helps to reduce loss for the farmers. Prediction also helps to increase the national economy.

1.3 OBJECTIVE OF THE PROJECT

This project is based on the curbing the problem faced by the farmers as well as providing the accurate level of the harvest they can expect from the crop they have growth depending on the dependent factors like temperature, rainfall, etc. This project is mainly developed to help farmers so that this may help them in analysis of the harvest of the crop. Farmers are facing loses in the crop yield due improper knowledge of the crop and the natural factor that are effecting them. In this project we analysis the factors like temperature, rainfall, pesticides and predict crop yield along with the best machine learning algorithm for crop yield prediction.

1.4 ORGANISATION OF THE REPORT

The first chapter deals with the introduction, motivating for developing this project, objective of the project. The second chapter deals with the literature survey, existing system and proposed system. The third chapter deals with the system specifications. The fourth chapter gives us the Design and implementations which includes introduction, source code and description of languages used for the project, methods of implementation. Finally, the fifth chapter deals with the conclusion.

2. LITERATURE SURVEY

2.1 Predicting Yield of the Crop Using Machine LearningAlgorithm

Authors: P.Priya, U.Muthaiah & M.Balamurugan

Publication: International Journal of Engineering Sciences & Research Technology (IJESRT), April, 2018.

This paper uses R programming with Machine Learning techniques. R is the leading tool for statistics, data analysis, and machine learning. It is more than a statistical package; it is a programming language, so you can create your own objects, functions, and packages. It is platform-independent, so it can be used on any operating system and it is free. R programs explicitly document the steps of our analysis and make it easy to reproduce and/or update analysis, which means it can quickly try many ideas and/or correct issues. All the datasets used in the research were sourced from the openly accessible records of the Indian Government. This was sourced for the years 1997 to 2013 for different seasons like Kharif and Rabi of rice production. From the vast initial dataset, only a limited number of important factors which have the highest impact on agricultural yield were selected for the present research. The dataset contains the following parameters: rainfall, season, and temperature and crop production. This paper also compares the two machine learning algorithms: decision trees and Random Forest.

- **Decision Tree**: The Decision tree classifiers uses greedy approach hence an attribute chosen at the first step cannot be used anymore which can give better classification if used in later steps. Also it over fits the training data which can give poor results for unseen Crop Yield Prediction based on Weather using Machine Learning data. So, to overcome this limitation ensemble model is used. In ensemble models results from different models are combined. The result obtained from an ensemble model is usually better than the result from any one of individual models.
- **Random Forest**: Random Forests is an ensemble classifier which uses many decision tree models to predict the result. A different subset of training data is selected, with replacement to train each tree. A collection of trees is a forest, and the trees are being trained on subsets which are being selected at random, hence random forests. This can be used for

classification and regression problems. Class assignment is made by the number of votes from all the trees and for regression the average of the results is used.

In this paper, the procedure they followed was as given below.

- Split the loaded data sets into two sets such as training data and test data in the split ratio of 67 and 33%.
- Then calculate Mean and Standard Deviation for needed tuples and then summarize the data sets. Compare the summarized data list and the original data sets & calculate the probability.
- Based on the result the largest probability produced is taken for prediction. The accuracy can be predicted by comparing the resultant class value with the test data set. The accuracy can range from 0% to 100%.

The paper concludes that the Results show that we can attain an accurate crop yield prediction using the Random Forest algorithm. The Random Forest algorithm achieves a largest number of crop yield models with the lowest models. It is suitable for massive crop yield prediction in agricultural planning. The dataset used for modelling here includes the climatic factors as well i.e., rainfall and temperature. The author did a comparative study of decision trees and random forest algorithms. But other algorithms were not considered and the dataset includes very few attributes that would not give accurate predictions.

2.2 Heuristic Prediction of Crop Yield using Machine Learning Technique

Authors: S. Pavani, Augusta Sophy Beulet P.

Publications: International Journal of Engineering and Advanced Technology (IJEAT) Volume9, December 2019

The paper says, vast research has been done and several attempts are made for application of Machine learning in agricultural fields. Major challenge in agriculture is to increase the production in the farm and deliver it to the end customers with best possible price and good quality. It is found that at least 50 percent of the farm produce never reaches the end consumer due to wastage and high-end prices. Machine learning based solutions developed to solve the difficulties faced by the farmers are being discussed in this work. The real time environmental

parameters of Telangana District like soil moisture, temperature, rainfall, humidity are collected and crop yield is being predicted using KNN Algorithm. There is a profound need to raise the farmer‘s income and ensure sustainable growth in Telangana to reduce poverty. Prediction of the yield of the crop in advance for a particular region depending on the climatic conditions and other factors which contribute to more yield is important, which will make farmers select the crop or to get better yield of their particular crops in the particular region. The use of data mining in rural areas has brought advantages in the field of research. This application is evaluations help farmers predict crop yields. Support Vector Machine (SVM), Regression analysis, K Nearest Neighbor (KNN), clustering and various types of techniques are used for prediction. KNN is used in this work. KNN is a language learning algorithm that is non- parametric. The aim is to use a model where information focuses are clustered in a few groups in order to predict the classification of another instance. KNN Algorithm depends on the closeness of the feature: After our training set, how firmly out of test highlights determine how we manage a given point of data. KNN can be utilized for classification- the yield or the output is a class membership (predicts a class-a discrete esteem or value). An element is distinguished by a larger proportion of its neighbours vote, with the entity being divided among its closest neighbours to the most regular class. It can also be used for regression-output, which is the item's reward (predicts unceasing qualities). This calculation is the average (or middle) of its nearest neighbours estimates of k. A commonly used metric for distance calculation in KNN algorithms is Euclidean distance

This technique can be used to predict crop yield with more accuracy for years of data together. The main principle reason of the work is to predict the crop yield in prior according to the details of the factors required for good crop yield.

In this paper, the data sets of different districts of the Telangana state are collected from Telangana State Development Planning Society. The important factors that determine the crop yield are temperature, humidity, soil moisture and rainfall. These samples were taken for the month of May 2019. Among the data set available maximum of the data was used for training and the remaining data was used for testing. The machine learning technique KNN algorithm was used for prediction of crop yield.

So with respect to the application of KNN algorithm towards prediction of crop yield, the nearest neighbours of a particular point (crop yield) like temperature, humidity, rainfall and soil moisture were considered. The implementation steps were:

1. Load the data set.
2. Initialize the k value.
3. For getting the anticipated class, repeat fromone to all the numbers ofthe training data set.
4. Compute the distance between test data and each line of training information. Here the Crop Yield Prediction based on Weather using Machine Learning Euclidean distance is utilized, since it is the most prominent technique.
5. Classify the computed distances in ascending order based on distance values.
6. Get top k rows from the classified array.
7. Get the most continuous class ofthese lines. 8. Return the anticipated class

The paper concludes that a model of machine learning to predict plant yield is proposed and gives reasonable crop yield suggestions for particular districts in Telangana. The research has been done on soil moisture, temperature, humidity and rainfall datasets of all the districts of Telangana State. By applying machine learning algorithms the model has been tested. K-NN suggests suitable accuracy in crop yield prediction. The well- constructed data set and the machine learning algorithm supports the proposed model. In future, providing other factors that greatly influence the crop yield is our concern, also more data of all these parameters of different seasons in the state will be added to make this model more accurate and efficient.

This paper includes a comparative study of the KNN, SVM and Linear Regression giving KNN as the most appropriate one with maximum accuracy. The climatic as well as soil properties are analyzed to predict the yield. This paper does not include recommendation of fertilizers or crops based on the soil, climate and location.

2.3 EXISTING SYSTEM

Other than blogging websites which provide information about the agriculture and agricultural accessories, there is no particular model for predicting the yield of the crop depending on the history in that specific geographical region.

2.4 PROPOSED SYSTEM

We have collected temperature, rainfall, crop yield and other datasets from various sources. In this project we have used machine learning algorithms like Decision tree, Random Forest algorithm, Linear Regression, Gradient Boosting to predict crop yield based on factors like temperature, rainfall, and pesticides. Our proposed model for predicting crop yield produces accurate results which will be more helpful for famers in choosing the crops that give best yield.

3. SYSTEM SPECIFICATIONS

To be used efficiently, all computer_software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements.

NON-FUNCTIONAL REQUIREMENTS

Nonfunctional requirements are the functions offered by the system. It includes time constraints and constraints on the development process and standards. The non-functional requirements are as follows:

- **Speed:** The system should process the given input into output within appropriate time.
- **Ease of use:** The software should be user friendly. Then the customers can use easily, so it doesn't require much training time.
- **Reliability:** The rate of failures should be less then only the system is more reliable
- **Portability**: It should be easy to implement in any system.

3.1 SOFTWARE SPECIFICATIONS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

The system should be able to interface with the existing system

- The system should be accurate
- The system should be better than the existing system

3.2 HARDWARE SPECIFICATIONS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list, especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture.

The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored.

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

RAM	:	8GB
Processor	:	1.9 gigahertz (GHz) x86- or x64-bit dual core processor with SSE2 instruction set
Hard Disk	:	10GB
Monitor	:	1024 X 768 resolution
Keyboard	:	101 keys

SOFTWARE REQUIREMENTS

Operating System	: Windows XP
Technology used	: Python
Platform	: Google Colab

4. DESIGN AND IMPLEMENTATION

4.1 INTRODUCTION

GOOGLE COLABORATORY

What is Colaboratory?

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

Google is quite aggressive in AI research. Over many years, Google developed an AI framework called TensorFlow and a development tool called Colaboratory. Today TensorFlow is open-sourced and since 2017, Google Colab or simply Colab.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for the public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold on a per-use basis.

Is it really free to use?

Yes. Colab is free to use.
As a programmer, you can perform the following using Google Colab.

- Write and execute code in python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

NUMPY

NumPy is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

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

SCIKIT LEARN

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python.

It was originally called **scikits.learn** and was initially developed by David Cournapeau as a Google summer of code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation), took this project at another level and made the first public release (v0.1 beta) on 1st Feb. 2010.

PANDAS

Pandas is a widely-used data analysis and manipulation library for Python. It provides numerous functions and methods that expedite the data analysis and preprocessing steps. Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

FEATURE-ENGINE

Feature-engine is a Python library with multiple transformers to engineer and select features to use in machine learning models. Feature-engine preserves Scikit-learn functionality with methods fit() and transform() to learn parameters from and then transform the data.

MATPLOTLIB

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with

the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc. We use plots like bar, pie, line, scatter etc for our project.

SEABORN

Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas' data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data. One has to be familiar with Numpy and Matplotlib and Pandas to learn about Seaborn. We are using Seaborn library to produce the 10day Rolling Mean sheet in the Sentiment VS Time phase of our Project.

4.2 METHODOLOGY USED

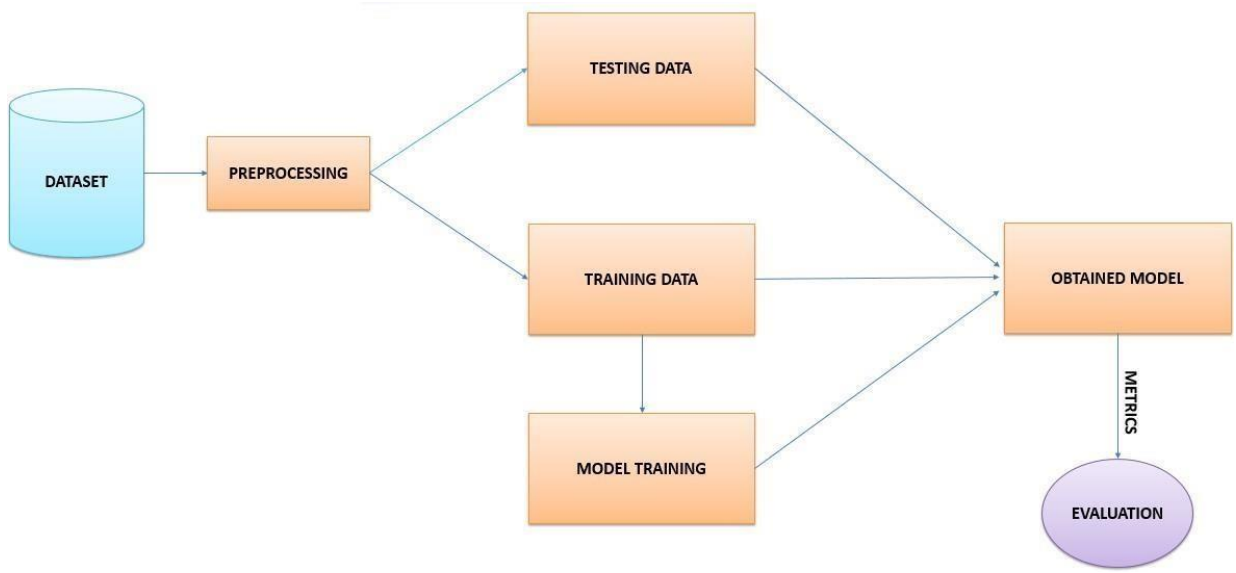

Fig 4.1: Process Flow of the Project

STEPS INVOLVED

1. Data collection
2. Data Exploration
3. Data Preprocessing
4. Feature Scaling
5. Modeling
6. Finding the best algorithm with high performance measures

DATA COLLECTION

Four different sets of data in this project namely pesticides, rainfall, temperature and yield data. These data sets are merged and formed a one dataset under merging module to perform modeling on it.

Dataset Details:

- Contains attributes like year, country, item, Rainfall(mm), Temperature, pesticides and yield.
- Number of Records in dataset are 25229

DATA EXPLORATION

Data exploration is an approach similar to initial data analysis, whereby a data analyst use visual exploration to understand what is in a dataset and the characteristics of the data, rather than through traditional data management systems. These characteristics can include size or amount data, completeness of the data, correction of the data, possible relationships among data elements or files/tables in the data. The Characteristics of data sets namely rainfall data, temperature data, yield data and pesticides data are represented in the figures 4.2, 4.3, 4.4, 4.5 respectively.

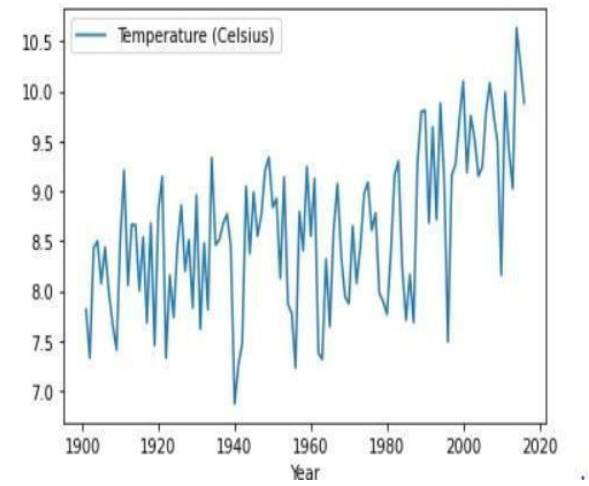

Fig:4.2 Temperature data

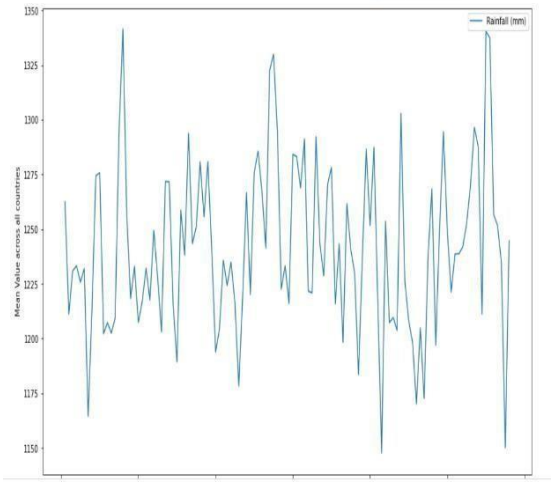

Fig:4.3 Rainfall data

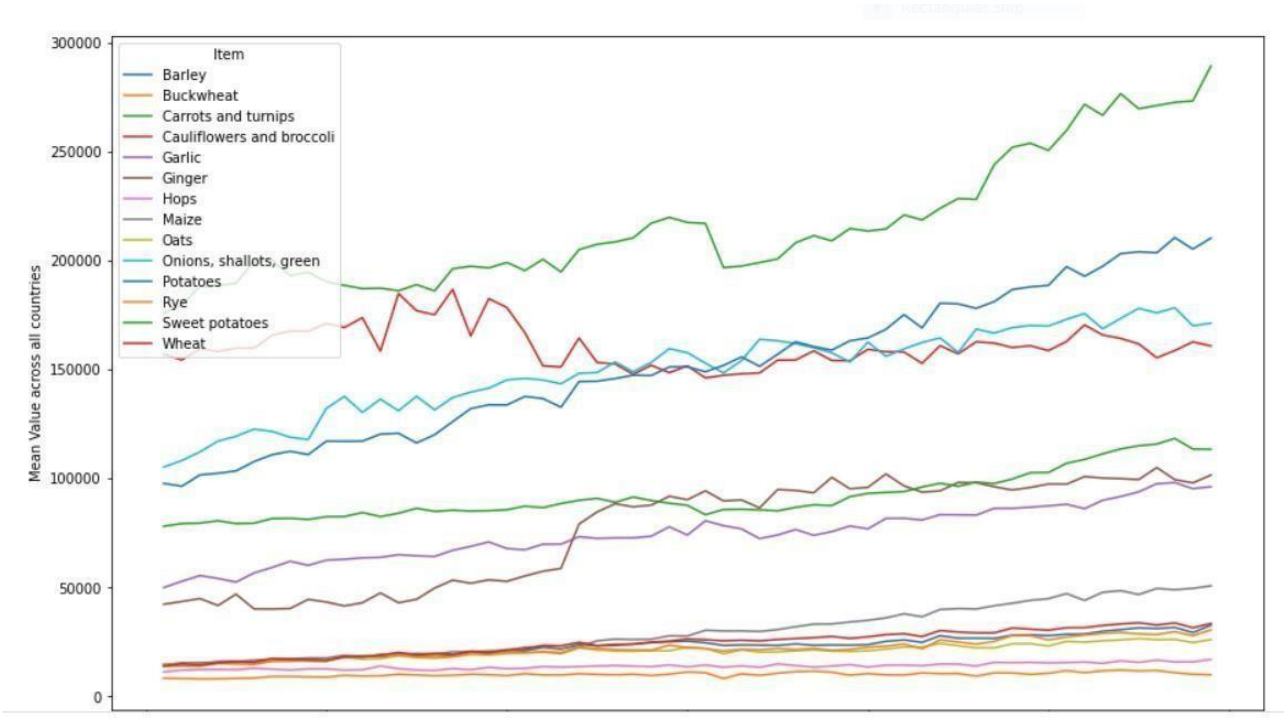

Fig:4.4 Yield data

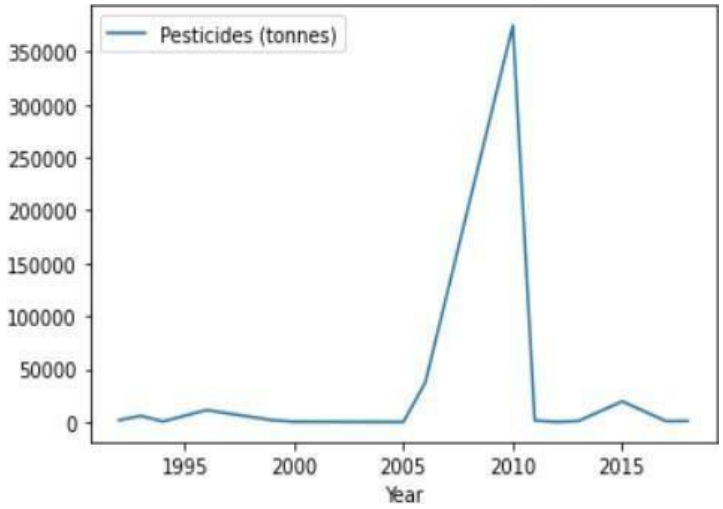

Fig:4.5 Pesticides data

DATA PREPROCESSING

Prior to the usage of the dataset for training and building the machine learning models, the data ispreprocessed from its raw format to a desirable clean format. Some of the preprocessing techniquesused in this project are as follows:

• Feature Scaling:

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model. Scaling is performed so that one significant feature doesn't impact the model just because of their large magnitude. In this project we have used MinMax scaler to scale yield attribute in the dataset. This MinMax estimator scales and transforms each feature individually such that it is in the given range for eg, between0 and 1.

• Checking for Missing Values in the Data:

Missing values can bias the results of the models, hence the data should be checked for missing values. If the data has missing values, then it should be taken care of.

• **Modifying Features:**

The features that are deemed to be unnecessary for training the models are dropped and features that are deemed to be useful are added to the existing dataset.

DATA MODELING

• **Test-Train split**

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions. It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced. The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset. Train Dataset is used to fit the machine learning model. Test Dataset is used to evaluate the fit machine learning model. The objective is to estimate the performance of the machine learning model on new data: data not used to train the model. This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values. The train-test procedure is appropriate when there is a sufficiently large dataset available.

REGRESSION ALGORITHMS

• Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

• Decision Tree

Decision Tree is a supervised learning technique that can be used for both classification and Regression problem. It is a tree where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset

• Gradient boosting

Gradient boosting is a machine learning technique of regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

• **Random Forest**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:

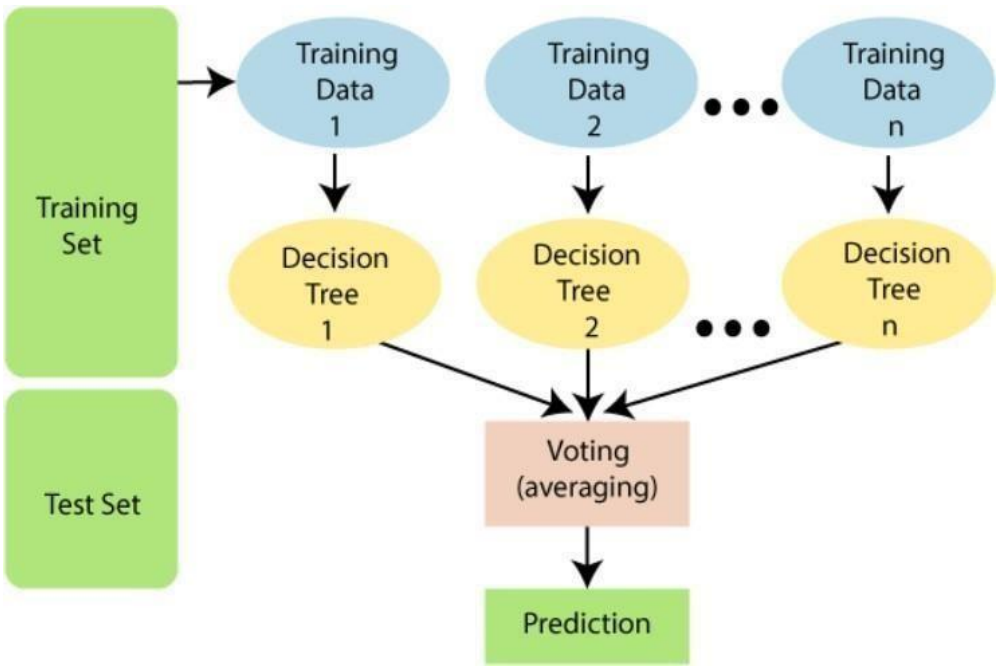

Fig: 4.6 Working of Random Forest Algorithm

4.3 SOURCE CODE

```
from google.colab import drive

drive.mount('/content/drive/')

import numpy as np

import pandas as pd

import  sklearn

import seaborn as sns

import matplotlib.pyplot as plt

pesticides_url ='/content/drive/MyDrive/Colab Notebooks/res/pesticides.csv'

temperature_url ='/content/drive/MyDrive/Colab Notebooks/res/temp.csv'

rainfall_url ='/content/drive/MyDrive/Colab Notebooks/res/rainfall.csv'

yield_url ='/content/drive/MyDrive/Colab Notebooks/res/yield.csv'

pesticides_data = pd.read_csv(pesticides_url,sep=',')

rainfall_data = pd.read_csv(rainfall_url,sep=', ')

temperature_data = pd.read_csv(temperature_url,sep=', ')

yield_data = pd.read_csv(yield_url, sep=',')

"""**Data Preparation**"""

rainfall_data

rainfall_data.info()
```

```
rainfall_data.describe()

len(rainfall_data['Country'].unique())

len(rainfall_data['Year'].unique())

rainfall_data.head(12)

rainfall_df = rainfall_data.groupby(['Year', 'Country', 'ISO3'], as_index=False, axis=0).sum()

rainfall_df

"""**Temperature Data**"""

temperature_data

temperature_data.info()

temperature_data.describe()

len(temperature_data['Country'].unique())

len(temperature_data['Year'].unique())

temperature_data.head(12)

temperature_df =temperature_data.groupby(['Year', 'Country', 'ISO3'], as_index=False, axis=0).mean()

temperature_df

"""**Yield data**"""

yield_data

yield_data.info()
```

```
yield_data.describe()

len(yield_data['Area'].unique())

len(yield_data['Year'].unique())

yield_df = yield_data.drop(['Domain', 'Element'], axis=1)

yield_df

"""**Pesticides data**"""

pesticides_data

pesticides_data.info()

pesticides_data.describe()

len(pesticides_data['Area'].unique())

len(pesticides_data['Year'].unique())

pesticides_df = pesticides_data.drop(['Domain', 'Element'], axis=1)

pesticides_df

"""**Merging**"""

rainfall_df.rename({'Rainfall - (MM)': 'Rainfall (mm)'}, axis=1, inplace=True)

temperature_df.rename({'Temperature - (Celsius)': 'Temperature (Celsius)'}, axis=1,

inplace=True)

yield_df.rename({'Area': 'Country', 'Value': 'Yield (hg/ha)'}, axis=1, inplace=True)

yield_df.drop('Unit', axis=1, inplace=True)
```

```
pesticides_df.rename({'Area': 'Country', 'Value': 'Pesticides (tonnes)'}, axis=1, inplace=True)

pesticides_df.drop(['Unit', 'Item'], axis=1, inplace=True)

rain_temp_df = pd.merge(rainfall_df, temperature_df, on=['Country', 'Year', 'ISO3'])

rain_temp_yield_df = pd.merge(rain_temp_df, yield_df, on=['Country', 'Year'])

rain_temp_yield_pest_df = pd.merge(rain_temp_yield_df, pesticides_df, on=['Country', 'Year'])

rain_temp_yield_pest_df.drop('ISO3', axis=1, inplace=True)

data = rain_temp_yield_pest_df[['Year', 'Country', 'Item', 'Rainfall (mm)', 'Temperature (Celsius)',

'Pesticides (tonnes)', 'Yield (hg/ha)']]

data

data.info()

data.describe()

len(data.Country.unique())

len(data.Year.unique())

"""**Data Exploration**

**Rainfall data**"""

rainfall_df

rainfall_df.loc[rainfall_df['Country'] == 'Germany'].groupby('Year').mean().plot()

plt.show()

temp_data = rainfall_df.groupby('Year').mean(
```

```
fig, ax = plt.subplots(figsize=(15,9))

fig.suptitle('Mean rainfall in mm across all countries between 1901 and 2016')

temp_data.plot(ax=ax)

ax.set_ylabel('Mean Value across all countries')

ax.set_xlabel('Year')

plt.show()

"""**Temperature data**"""

temperature_df.loc[temperature_df['Country'] == 'Germany'].groupby('Year').mean().plot()

plt.show()

# prepare data

temp_data = temperature_df.groupby('Year').mean()

# plot  data

fig, ax = plt.subplots(figsize=(15,9))

fig.suptitle('Mean temperature in celsius across all countries between 1901 and 2016')

temp_data.plot(ax=ax)

ax.set_ylabel('Mean Value across all countries')

ax.set_xlabel('Year')

plt.show()
```

```
"""**Yield data**"""

# prepare data

yield_mean = yield_data.groupby(['Year', 'Item']).mean()

# plot data

fig, ax = plt.subplots(figsize=(15,9))

fig.suptitle('Mean harvested value across all countries between 1961 and 2019')

yield_mean['Value'].unstack().plot(ax=ax)

ax.set_ylabel('Mean Value across all countries')

ax.set_xlabel('Year')

plt.show()

fig, axs = plt.subplots(1,1, figsize=(15,9))

fig.suptitle('Frequency Count', size=30)

temp_df = yield_data['Item'].value_counts().to_frame().reset_index()

g = sns.barplot(x='index', y='Item', data=temp_df, orient='v', ax=axs)

for index, row in temp_df.iterrows():

    g.text(row.name,row.Item + 100, row.Item, color='black', ha="center")

axs.set_xlabel('Item')

axs.set_ylabel('Count')

fig.show()
```

```
pesticides_df.loc[temperature_df['Country'] == 'Germany'].groupby('Year').mean().plot()

plt.show()

# prepare data

temp_data = pesticides_df.groupby('Year').mean()

# plot data

fig, ax = plt.subplots(figsize=(15,9))

fig.suptitle('Mean pesticide usage in tonnes across all countries between 1990 and 2018')

temp_data.plot(ax=ax)

ax.set_ylabel('Mean Value across all countries')

ax.set_xlabel('Year')

plt.show()

"""**Merged data**"""

# prepare data

temp_data = data.groupby(['Year', 'Item']).mean()

# plot data

fig, ax = plt.subplots(figsize=(15,9))

fig.suptitle('Mean harvested value across all countries between 1961 and 2019')

temp_data['Yield (hg/ha)'].unstack().plot(ax=ax)
```

```
ax.set_ylabel('Mean Value across all countries')

ax.set_xlabel('Year')

plt.show()

fig, axs = plt.subplots(1,1, figsize=(15,9))

fig.suptitle('Frequency Count', size=30)

temp_df = data['Item'].value_counts().to_frame().reset_index()

g = sns.barplot(x='index', y='Item', data=temp_df, orient='v', ax=axs)

for index, row in temp_df.iterrows():

    g.text(row.name,row.Item + 50, row.Item, color='black', ha="center")

axs.set_xlabel('Item')

axs.set_ylabel('Count')

fig.show()

corr = data.select_dtypes(include=[np.number]).corr()

mask = np.zeros_like(corr, dtype=np.bool)

mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap

cmap = sns.palette="vlag"

# Draw the heatmap with the mask and correct aspect ratio
```

```
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5});
plt.figure(figsize=(20,3))
plt.subplot(1,2,1)
sns.boxplot(x=data['Yield (hg/ha)'],color='#005030')
plt.title(f'Box Plot of Preis')
plt.subplot(1,2,2)
sns.histplot(x=data['Yield (hg/ha)'], color='#500050', kde=True)
plt.title(f'Distribution Plot of Preis')
plt.show()
"""**Data Preprocessing**
**One Hot Encoding**"""
data.shape
from sklearn.preprocessing import OneHotEncoder
df_onehot = pd.get_dummies(data, columns=['Country', 'Item'], prefix=['Country', 'Item'])
data = df_onehot.loc[:, df_onehot.columns != 'Yield (hg/ha)']
data['Yield (hg/ha)'] = df_onehot['Yield (hg/ha)']
data.shape
"""**Backward Elimination**"""
```

```
data.shape

import statsmodels.api as sm

y = data['Yield (hg/ha)']

X = data.drop('Yield (hg/ha)', axis=1)

cols = list(X.columns)

pmax = 1

while (len(cols)>0):

    p = []

    X_1 = X[cols]

    X_1 = sm.add_constant(X_1)

    model = sm.OLS(y,X_1).fit()

    p = pd.Series(model.pvalues.values[1:], index = cols)

    pmax = max(p)

    feature_with_p_max = p.idxmax()

    if(pmax>0.05):

        cols.remove(feature_with_p_max)

    else:

        break
```

```
data = data[cols]

data.insert(len(data.columns), 'Yield (hg/ha)', y)

data.shape

"""**Removing of Outliers**"""

data.shape

from scipy import stats

y = data['Yield (hg/ha)']

X = data.drop('Yield (hg/ha)', axis=1)

z_scores = stats.zscore(X)

abs_z_scores = np.abs(z_scores)

filtered_entries = (abs_z_scores < 11).all(axis=1)

X = X[filtered_entries]

X.insert(len(X.columns), 'Yield (hg/ha)', y)

data = X

data.shape

"""**Feature Scaling**"""

data.shape

from sklearn.preprocessing import MinMaxScaler

y = data['Yield (hg/ha)']
```

```
X = data.drop('Yield (hg/ha)', axis=1)

scaler = MinMaxScaler()

data_without_yield = pd.DataFrame(scaler.fit_transform(X), index=y.index)

data_without_yield.columns = X.columns

data_without_yield.insert(len(data_without_yield.columns), 'Yield (hg/ha)', y)

data = data_without_yield

data.shape

"""**Modeling**"""

y = data['Yield (hg/ha)']

X = data.drop('Yield (hg/ha)', axis=1)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

def mean_absolute_percentage_error(y_true, y_pred):

y_true, y_pred = np.array(y_true), np.array(y_pred)

return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def plot_regression_results(ax, y_test, y_pred, title, estimated_time, scores):

# linear least-squares

slope, intercept, rvalue, pvalue, stderr = linregress(y_test, y_pred)
```

```
ax.plot([y_test.min(), y_test.max()],   [intercept+y_test.min()*slope,
intercept+y_test.max()*slope], '--r')

ax.scatter(y_test, y_pred, alpha=0.7)

extra = plt.Rectangle((0, 0), 0, 0, fc="w", fill=False)

ax.legend([extra], [scores], loc='upper left')

ax.set_xlabel('Actual values in tonnes')

ax.set_ylabel('Predictes values in tonnes')

ax.set_title('{}\nTrained in {:.2f} Milliseconds'.format(name, estimated_time*1000))

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, max_error

from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor

from sklearn.model_selection import cross_validate, cross_val_predict

from sklearn.linear_model import LinearRegression, SGDRegressor

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeRegresson

import time

from scipy.stats
```

```
lin = LinearRegression()

dtr = DecisionTreeRegressor()

sgd = SGDRegressor(loss='squared_loss')

gbr = GradientBoostingRegressor()

knn = KNeighborsRegressor(n_neighbors=5)

rfr = RandomForestRegressor()

estimators = [('Linear Regression', lin),

              ('Decision Tree Regression', dtr),

              ('Stochastic Gradient Descent Regression', sgd),

              ('Gradient Boosting Regression', gbr),

              ('K-nearest Neighbour 5', knn),

              ('Random Forest Regression', rfr)]

fig, axs = plt.subplots(nrows=2, ncols=3, sharex=True, sharey=True, figsize=(20, 13))

axs = np.ravel(axs)

for ax, (name, est) in zip(axs, estimators):

  start_time = time.time()

  est.fit(X_train, y_train)
```

```
  y_pred = est.predict(X_test)

  estimated_time = time.time() - start_time

  plot_regression_results(ax, y_test, y_pred, name, estimated_time,

                          (r'$R^2$ = {:.2f}' + '\n' +

                          r'MAE = {:.0f}' + '\n' +

                          r'MSE = {:.0f}' + '\n' +

                          r'RMSE = {:.0f}' + '\n' +

                          r'MAX = {:.0f}' + '\n' +

                          r'MAPE = {:.2f}%')

                          .format(r2_score(y_test, y_pred),

                          mean_absolute_error(y_test, y_pred),

                          mean_squared_error(y_test, y_pred),

                          mean_squared_error(y_test, y_pred, squared=False),

                          max_error(y_test, y_pred),

                          mean_absolute_percentage_error(y_test, y_pred)))
plt.suptitle('Regressionsverfahren')

plt.tight_layout()

plt.subplots_adjust(top=0.9)

plt.show()
```

```
r2_values = []

max_error_values = []

neg_mean_absolute_error_values = []

neg_mean_squared_error_values = []

neg_root_mean_squared_error_values = []

for name, est in estimators:

 score = cross_validate(est, X_train, y_train, cv=5, scoring=['r2', 'max_error',

 'neg_mean_absolute_error',

 'neg_mean_squared_error','neg_root_mean_squared_error'],n_jobs=-1)

max_error_values.append(-score['test_max_error'])

neg_mean_absolute_error_values.append(-score['test_neg_mean_absolute_error'])

neg_mean_squared_error_values.append(-score['test_neg_mean_squared_error'])

neg_root_mean_squared_error_values.append(-score['test_neg_root_mean_squared_error'])

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(30, 5))

names = ['LR', 'DTR', 'SGD', 'GBR', 'KNN', 'RFR']

axs[0].boxplot(r2_values, labels=names)

axs[0].set_title('R2')

axs[1].boxplot(max_error_values, labels=names)

axs[1].set_title('MAX')
```

```
axs[2].boxplot(neg_mean_absolute_error_values, labels=names)

axs[2].set_title('MAE')

axs[3].boxplot(neg_mean_squared_error_values, labels=names)

axs[3].set_title('MSE')

axs[4].boxplot(neg_root_mean_squared_error_values, labels=names)

axs[4].set_title('RMSE')

plt.suptitle('Cross-validation')

plt.show()

regression = 5

print(u'R²: {:.3f} \u00B1 {:.3f}'.format(np.mean(r2_values[regression]), np.std(r2_values[regression])
))

print(u'MAX: {:,.0f} \u00B1 {:,.0f}'.format(np.mean(max_error_values[regression]), np.std(max_erro
r_values[regression])))

print(u'MAE: {:,.0f} \u00B1 {:,.0f}'.format(np.mean(neg_mean_absolute_error_values[regression]), n
p.std(neg_mean_absolute_error_values[regression])))

print(u'MSE: {:,.0f} \u00B1 {:,.0f}'.format(np.mean(neg_mean_squared_error_values[regression]), n
p.std(neg_mean_squared_error_values[regression])))

print(u'RMSE: {:,.0f} \u00B1 {:,.0f}'.format(np.mean(neg_root_mean_squared_error_values[regressi
on]), np.std(neg_root_mean_squared_error_values[regression])))
```

4.3.1 OUTPUT SCREENS

=

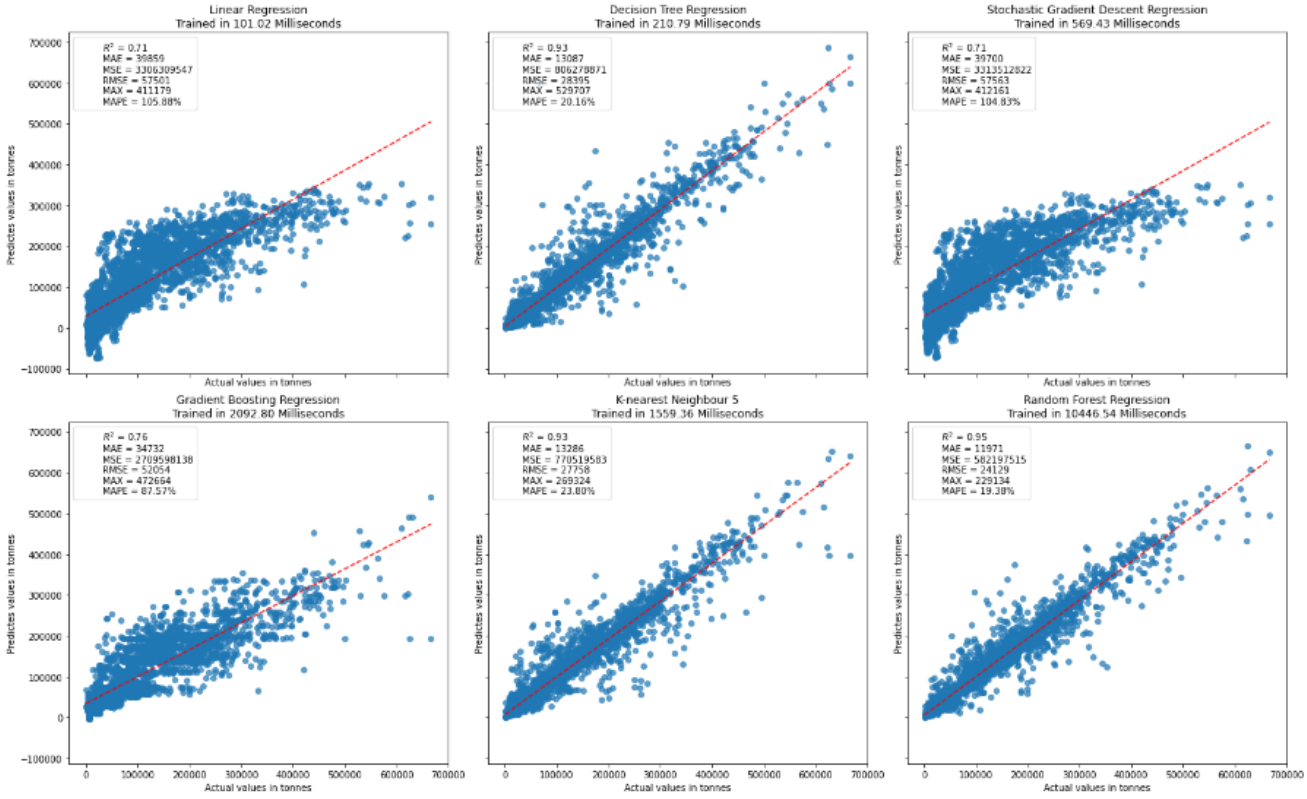

Fig:4.3.1.1 Graph plots representing various Regression models along with their performance measures

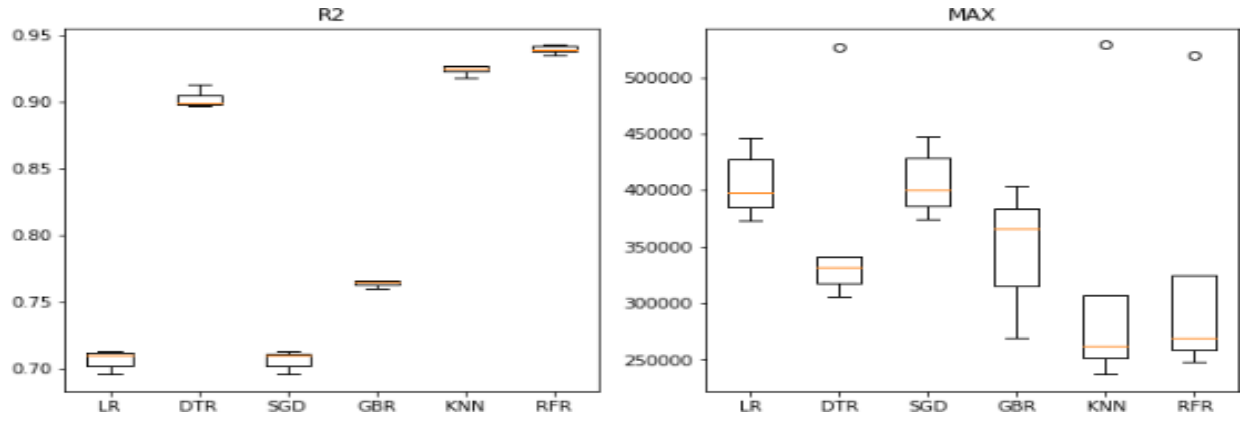

Fig:4.3.1.2 Box plots representing R2 score and MAX error

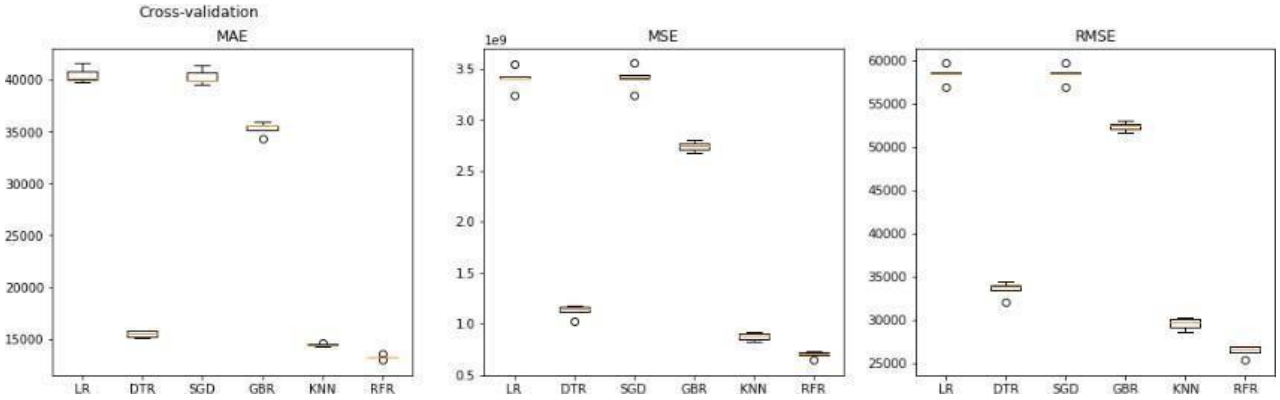

Fig:4.3.1.3 Box plots representing MAE, MSE and RMSE errors

Models/ Metrics	Linear Regression	Decision Tree Regression	SGD Regression	Gradient boosting	KNN	Random Forest
R-SQUARE	0.71	0.93	0.71	0.76	0.93	0.95
MAE	39859	13072	39833	34734	13286	11982
MSE	33063	79899	33136	27101	77051	57550
RMSE	57501	28266	57564	52059	27758	23990
MAPE	105.88	20.20	105.64	87.57	23.80	19.59

Fig:4.3.1.4 Comparison of Performance measures of various algorithms used in modeling

4.4 TESTING AND VALIDATION

TESTING

The purpose of testing can be quality assurance, verification and validation, or reliability estimation. Testing can be used as a generic metric as well. Correctness testing and reliability testing are two major areas of testing. Software testing is a trade-off between budget, time and quality. The main course of testing is to check for the existence of defects or errors in a program or project or product, based up on some predefined instructions or conditions.

Verification : Checks consistency with the inputs

The process of determining whether or not the products of a given phase of the software development cycle meets the implementation steps and can be traced to the incoming objectives established during the previous phase. The techniques for verification are testing, inspection.
Validation : Checks Consistency with the User Requirements.

The process of evaluating software at the end of the software development process to ensure compliance with software requirements. The techniques for validation are testing, inspection and reviewing.

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding software bugs.

Testing can never completely establish the correctness of computer software. Instead, it furnishes a *criticism* or *comparison* that compares the state and behavior of the product against oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Over its existence, computer software has continued to grow in complexity and size. Every software product has a target audience. For example, the audience for video game software is

completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it presumably must assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. **Software testing** is the process of attempting to make this assessment.

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirements gaps, e.g., unrecognized requirements, that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Software faults occur through the following process. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

A frequent cause of software failure is compatibility with another application, a new operating system, or, increasingly, web browser version. In the case of lack of backward compatibility, this can occur (for example...) because the programmers have only considered coding their programs for, or testing the software upon, "the *latest* version of" this-or-that operating

system. The unintended consequence of this fact is that: their latest work might not be fully compatible with earlier mixtures of software/hardware, or it might not be fully compatible with *another* important operating system. In any case, these differences, whatever they might be, may have resulted in (unintended...) software failures, as witnessed by some significant population of computer users.

UNIT TESTING

In computer programming, **unit testing** is a software design and development method where the programmer gains confidence that individual units of source code are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

Unit testing can be done by something as simple as stepping through code in a debugger; modern applications include the use of a test framework such as XUnit.

Ideally, each test case is independent from the others; Double objects like stubs, mock or fake objects as well as test harnesses can be used to assist testing a module in isolation. Unit testing is typically done by software developers to ensure that the code other developers have written meets software requirements and behaves as the developer intended.

The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behavior. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.

Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

On the other hand, ordinary narrative documentation is more susceptible to drifting from the implementation of the program and will thus become outdated (e.g. design changes, feature creep, relaxed practices in keeping documents up-to-date).

When software is developed using a test-driven approach, the Unit-Test may take the place of formal design. Each unit test can be seen as a design element specifying classes, methods, and observable behaviour. The following Java example will help illustrate this point.

Here is a test class that specifies a number of elements of the implementation. First, that there must be an interface called Adder, and an implementing class with a zero-argument constructor called AdderImpl. It goes on to assert that the Adder interface should have a method called add, with two integer parameters, which returns another integer. It also specifies the behaviour of this method for a small range of values.

LIMITATIONS

Testing cannot be expected to catch everyerror in the program - it is impossible to evaluate all execution paths for all but the most trivial programs. The same is true for unit testing. Additionally, by definition unit testing only tests the functionality of the units themselves. Therefore, it will not catch integration errors, or broader system level errors (such as functions performed across multiple units, or non-functional test areas such as performance). Unit testing is more effective if it is used in conjunction with other software testing activities. Like all forms of software testing, unit tests can only show the presence of errors; it cannot show the absence of errors.

Software testing is a combinatorial problem. For example, every boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

To obtain the intended benefits from unit testing, a rigorous sense of discipline is needed throughout the software development process. It is essential to keep careful records not only of the tests that have been performed, but also of all changes that have been made to the source code of this or anyother unit in the software. Use of a version control system is essential. If a later version of the unit fails a particular test that it had previously passed, the version-control software can provide a list of the source code changes (if any) that have been applied to the unit since that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately.If such a process is not implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite increasing false positives and reducing the effectiveness of the test suite.

ACCEPTANCE TESTING

Acceptance testing is formal testing conducted to determine whether a system satisfies its acceptance criteria and thus whether the customer should accept the system.

The main types of software testing are:

- Component.
- Interface.
- System.
- Acceptance.
- Release.

Acceptance Testing checks the system against the "Requirements". It is similar to systems testing in that the whole system is checked but the important difference is the change in focus:

Systems Testing checks that the system that was specified has been delivered.

Acceptance Testing checks that the system delivers what was requested. The customer, and not the developer should always do acceptance testing. The customer knows what is required

from the system to achieve value in the business and is the only person qualified to make that judgment. The User Acceptance Test Plan will vary from system to system but, in general, the testing should be planned in order to provide a realistic and adequate exposure of the system to all reasonably expected events. The testing can be based upon the User Requirements Specification to which the system should conform.

In engineering and its various sub disciplines, **acceptance testing** is black-box testing performed on a system (e.g. software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery. It is also known as functional testing, black-box testing, release acceptance, QA testing, application testing, confidence testing, final testing, validation testing, or factory acceptance testing.

In software development, **acceptance testing** by the system provider is often distinguished from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership. In such environments, acceptance testing performed by the customer is known as user acceptance testing (UAT). This is also known as end-user testing, site (acceptance) testing, or field (acceptance) testing.

A smoke test is used as an acceptance test prior to introducing a build to the main testing process.

Acceptance testing generally involves running a suite of tests on the completed system. Each individual test, known as a case, exercises a particular operating condition of the user's environment or feature of the system, and will result in a pass or fail boolean outcome. There is generally no degree of success or failure. The test environment is usually designed to be identical, or as close as possible, to the anticipated user's environment, including extremes of such. These test cases must each be accompanied by test case input data or a formal description of the operational activities (or both) to be performed—intended to thoroughly exercise the specific case—and a formal description of the expected results.

Acceptance Tests/Criterion (in Agile Software Development) are usually created by business customers and expressed in a business domain language. These are high level tests to test the completeness of a user story or stories 'played' during any sprint/iteration. These tests are

created ideally through collaboration between business customers, business analysts, testers and developers, however the business customers (product owners) are the primary owners of these tests. As the user stories pass their acceptance criteria, the business owners can be sure of the fact that the developers are progressing in the right direction about how the application was envisaged to work and so it's essential that these tests include both business logic tests as well as UI validation elements (if need be).

Acceptance test cards are ideally created during sprint planning or iteration planning meeting, before development begins so that the developers have a clear idea of what to develop. Sometimes (due to bad planning!) acceptance tests may span multiple stories (that are not implemented in the same sprint) and there are different ways to test them out during actual sprints. One popular technique is to mock external interfaces or data to mimick other stories which might not be played out during an iteration (as those stories may have been relatively lower business priority). A user story is not considered complete until the acceptance tests have passed.

The acceptance test suite is run against the supplied input data or using an acceptance test script to direct the testers. Then the results obtained are compared with the expected results. If there is a correct match for every case, the test suite is said to pass. If not, the system may either be rejected or accepted on conditions previously agreed between the sponsor and the manufacturer.

The objective is to provide confidence that the delivered system meets the business requirements of both sponsors and users. The acceptance phase may also act as the final quality gateway, where any quality defects not previously detected may be uncovered.

A principal purpose of acceptance testing is that, once completed successfully, and provided certain additional (contractually agreed) acceptance criteria are met, the sponsors will then sign off on the system as satisfying the contract (previously agreed between sponsor and manufacturer), and deliver final payment.

User Acceptance Testing (UAT) is a process to obtain confirmation by a Subject Matter Expert (SME), preferably the owner or client of the object under test, through trial or review, that the modification or addition meets mutually agreed-upon requirements. In software development,

UAT is one of the final stages of a project and often occurs before a client or customer accepts the new system.

Users of the system perform these tests which developers derive from the client's contract or the user requirements specification.

Test designers draw up formal tests and device a range of severity levels. It is preferable that the designer of the user acceptance tests not be the creator of the formal integration and system test cases for the same system, however there are some situations where this may not be avoided. The UAT acts as a final verification of the required business function and proper functioning of the system, emulating real-world usage conditions on behalf of the paying client or a specific large customer. If the software works as intended and without issues during normal use, one can reasonably infer the same level of stability in production. These tests, which are usually performed by clients or end-users, are not usually focused on identifying simple problems such as spelling errors and cosmetic problems, nor show stopper bugs, such as software crashes; testers and developers previously identify and fix these issues during earlier unit testing, integration testing, and system testing phases.

The results of these tests give confidence to the clients as to how the system will perform in production. There may also be legal or contractual requirement for acceptance of the system.

TYPES OF ACCEPTANCE TESTING

User acceptance testing

This may include factory acceptance testing, i.e. the testing done by factory users before the factory is moved to its own site, after which site acceptance testing may be performed by the users at the site.

Operational acceptance testing

Also known as operational readiness testing, this refers to the checking done to a system to ensure that processes and procedures are in place to allow the system to be used and maintained. This may include checks done to back-up facilities, procedures for disaster recovery, training for end users, maintenance procedures, and security procedures.

Contract and regulation acceptance testing

In contract acceptance testing, a system is tested against acceptance criteria as documented in a contract, before the system is accepted. In regulation acceptance testing, a system is tested to ensure it meets governmental, legal and safety standards.

Alpha and beta testing

Alpha testing takes place at developers' sites, and involves testing of the operational system by internal staff, before it is released to external customers. Beta testing takes place at customers' sites, and involves testing by a group of customers who use the system at their own locations and provide feedback, before the system is released to other customers. The latter is often called “field testing”.

INTEGRATION TESTING

Testing in which software components, hardware components, or both together are combined and tested to evaluate interactions between them. Integration testing takes as its input modules that have been checked out by unit testing, 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.

'Integration testing' (sometimes called Integration and Testing, abbreviated **I&T**) is the phase of software testing in which individual software modules are combined and tested as a group. It follows unit testing and precedes system testing.

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.

The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using Black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input

interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing.

The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.Some different types of integration testing are big bang, top-down, and bottom-up.

There are two types in Integration Testing.

1. Bottom Up

2. Top Down.

Bottom-UP testing: An approach to integration testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottomor low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage. The Top Down is the procedure where the top integrated modules are tested and the branch of the module is testing step by step till end of the related module. The main advantage of the Bottom up is easy to find bugs. In Top down it is easy to find the missing branch link.

Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component, in this sense, refers to an integrated aggregate of more than one unit. In a realistic scenario, many units are combined into components, which are in turn aggregated into even larger parts of the program. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process

are tested together. Beyond that, if the program is composed of more than one process, they should be tested in pairs rather than all at once.

Integration testing identifies problems that occur when units are combined. By using a test plan that requires you to test each unit and ensure the viability of each before combining units, you know that any errors discovered when combining units are likely related to the interface between units. This method reduces the number of possibilities to a far simpler level of analysis.

You can do integration testing in a variety of ways but the following are three common strategies:

- The top-down approach to integration testing requires the highest-level modules be test and integrated first. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers. However, the need for stubs complicates test management and low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality.
- The bottom-up approach requires the lowest-level units be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized. The downside, however, is that the need for drivers complicates test management and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.
- The third approach, sometimes referred to as the umbrella approach, requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. It also helps minimize the need for stubs and drivers. The potential weaknesses of this approach are significant, however, in that it can be less systematic than the other two approaches, leading to the need for more regression testing.

REGRESSION TESTING

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process. Test department coders develop code test scenarios and exercises that will test new units of code after they have been written. Before a new version of a software product is released, the old test cases are run against the new version to make sure that all the old capabilities still work. The reason they might not work because changing or adding new code to a program can easily introduce errors into code that is not intended to be changed. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.

Regression testing is any type of software testing which seeks to uncover software regressions. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes. *Common methods* of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged.

Effective regression tests generate sufficient code execution coverage to exercise all meaningful code branches. Therefore, software testing is a combinatorial problem. However, in practice many combinations are unreachable so the problem size is greatly reduced. Every boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

Traditionally, in the corporate world, regression testing has been performed by a software quality assurance team after the development team has completed work. However, defects found at this stage are most costly to fix. This problem is being addressed by the rise of developer testing. Although developers have always written test cases as part of the development cycle, these test cases have generally been either functional tests or unit tests that verify only intended outcomes. Developer testing compels a developer to focus on unit testing and to include both positive and negative test cases.

Any time you modify an implementation within a program, you should also do regression testing. You can do so by rerunning existing tests against the modified code to determine whether the changes break anything that worked prior to the change and by writing new tests where necessary. Adequate coverage without wasting time should be a primary consideration when conducting regression tests. Try to spend as little time as possible doing regression testing without reducing the probability that you will detect new failures in old, already tested code.

Some strategies and factors to consider during this process include the following:

- Test fixed bugs promptly. The programmer might have handled the symptoms but not have gotten to the underlying cause.
- Watch for side effects of fixes. The bug itself might be fixed but the fix might create other bugs.
- Write a regression test for each bug fixed.
- If two or more tests are similar, determine which is less effective and get rid of it.
- Identify tests that the program consistently passes and archive them.
- Focus on functional issues, not those related to design.
- Make changes (small and large) to data and find any resulting corruption.
- Trace the effects of the changes on program memory.

The selective retesting of a software system that has been modified to ensure that any bugs have been fixed and that no other previously working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the software. Also referred to as *verification testing*, regression testing is initiated after a programmer has attempted to fix a recognized problem or has added source code to a program that may have inadvertently introduced errors. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.

Regression testing means rerunning test cases from existing test suites to build confidence that software changes have no unintended side-effects. The “ideal” process would be to create an

extensive test suite and run it after each and every change. Unfortunately, for many projects this is just impossible because test suites are too large, because changes come in too fast, because humans are in the testing loop, because scarce, highly in-demand simulation laboratories are needed, or because testing must be done on many different hardware.

Researchers have tried to make regression testing more effective and efficient by developing regression test selection (RTS) techniques, but many problems remain, such as:

- **Unpredictable performance.** RTS techniques sometimes save time and money, but they sometimes select most or all of the original test cases. Thus, developers using RTS techniques can find themselves worse off for having done so.
- **Incompatible process assumptions.** Testing time is often limited (e.g., must be done overnight). RTS techniques do not consider such constraints and, therefore, can and do select more test cases than can be run.
- **Inappropriate evaluation models.** RTS techniques try to maximize average regression testing performance rather than optimize aggregate performance over many testing sessions. However, companies that test frequently might accept less effective, but cheaper individual testing sessions if the system would, nonetheless, be well-tested over some short period of time.

These and other issues have not been adequately considered in current research, yet they strongly affect the applicability of proposed regression testing processes. Moreover, we believe that solutions to these problems can be exploited, singly and in combination, to dramatically improve the costs and benefits of the regression testing process.

SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any

applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

5. CONCLUSION

5.1 CONCLUSION

This system is proposed to deal with the increasing rate of farmer suicides and to help them to grow financially stronger. The proposed model helps the farmers to predict the yield of a given crop and also helps them to decide which crop to grow**.** This model for Crop Yield Prediction is able to predict the accurate yield for crops. Based on our analysis, Random Forest model produces more accurate results than the other models. Our system accuracy is more than the existing system. Since we are displaying the results in the form of graph with actual and predicted it is easy to compare the previous data. This model will help farmers to grow the crop which will give more yield so that it will be more profitable**.**

REFERENCES

1. P.Priya, U.Muthaiah & M.Balamurugan, Predicting yield of the crop using machine learning algorithm, International journal of Engineering Sciences and Research Technology (IJESRT), April – 2018.

2. S. Pavani, Augusta Sophy Beulet P, Heuristic Prediction of Crop Yield using Machine Learning Technique, International Journal of Engineering and Advanced Technology (IJEAT), December-2019.

3. https://www.geeksforgeeks.org/machine-learning/

4. https://www.w3schools.com/python/