CSCE 5300 Project

The Predictive Analysis of Crop Recommendation using HADOOP and BIG DATA

Submitted By - Group 8

1.Aditya Naga Venkatakrishna Chaitanya Kandregula

2.Anirudha Kulakarni Karanam

3.Mahaboob Ali Chisti Shaik

4.Surya Vamsi Chintapalli

1. Introduction:

Agriculture is an important source of economic development. It is the backbone of the world's economy and provides food, fibre and other essential resources to sustain life. Agriculture is still a well-known occupation in many countries like India, USA, China, Brazil and etc.

The crop yield mainly depends on weather conditions i.e. Temperature and rainfall. Today youngsters are very much interested in farming system because the current status in Indian agriculture is very satisfactory and using latest technologies to make this agriculture more prestigious as the world's economy is increasing year by year and the need of food is becoming more valuable.

In this project we gather and analyse various types of data related to agriculture such as temperature, rainfall data in selected regions. The aim of this datasets is to provide farmers with valuable insights that can help them improve the yield and quality of their crops.

By collecting temperature and rainfall datasets, for example farmers can determine the best time to cultivate agriculture so that the farmers can get a good profit income.

2. Goals & Objectives:

2.1. Motivation:

The main motivation of this project is to predict the crop yield and whether the crops are grown in a good geological condition. In the world, the population has increased the productivity of the food also need to be good and sufficient for future generations since the world is growing faster in terms of technology and industries. Due to this a lot of gases and chemicals are released into the global environment. This is impacting the climatic conditions and there is a sudden change in the weather seasons. This will impact the production of food.

In today's world to identify the real problem we are predicting which conditions are good to cultivate. At the same time, we can also identify times when the production is less. By these abnormal changes, we can take preventive measures to safeguard everyone's hunger.

2.2. Significance:

Due to weather changes, farmers do not know when to sow or harvest it is becoming difficult these days. Prediction of temperature and monthly rainfall data helps to better understand what best times for sowing and harvesting and find out if a crop can grow in a suitable environment or not. This prediction could also help scientists to find the root cause or solution for low yield in crops.

2.3. Objectives

The main goal is to collect data of weather, rainfall & soil (in terms of soil nutrients like nitrogen, potassium, phosphorus etc. & their PH values) properties of different regions and analyze data. Once the data is collected, we will process the data and run through spark's regression models to get future predictions of required data for the crop yield and that predicted values are used to see if all the conditions were suitable for its growth. This will help to predict the crop yield for upcoming days.

For example, we are considering the wheat crop, for the growth of wheat the ideal conditions are as follows:

- Nutrients in the soil viz. Nitrogen, phosphorus, and potassium should be in doses of 50, 25, and 12 kg per acre.
- Temperature must be between 10-15 degrees centigrade at the time of sowing
- Temperature must be between 21-26 degrees centigrade at the time of harvesting
- Wheat productivity is good if the rainfall is between 75 and 100 cm.

Based on these ideal conditions and the predicted data of rainfall & weather and changes in soil properties we can determine which time in the future is better to grow wheat.

2.4. Features:

1.**Temperature:** As mentioned in the objectives, the temperature could affect the growth of the crop, so this could be measured and used as a feature.

2.**Rainfall:** The amount of rainfall could be measured and used as a feature, as this is important for determining the suitability of the environment for the crop.

3.**Location:** The location where the crop is being grown could also be considered as a feature, as different regions may have different ideal conditions for crop growth.

4.**Recommendation Dataset :** At last, we are obtaining the values of temperature and rainfall We are comparing them with the Crop Recommendation Dataset.

5.**Year** : We have taken this to collect annual average Temperature and Rainfall values.

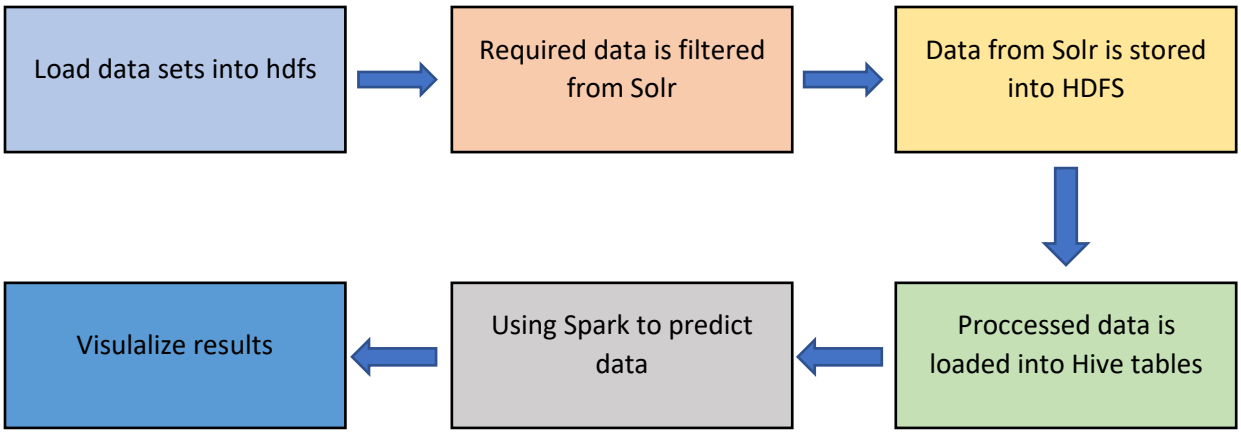
6.**T average:** By using Temperature average feature we are obtaining month wise average Data.

7.SUBDIVISION: In this feature we are describing the area or a particular Region**.**

8.Annual : By using this we can collect all month's data by averaging the values.

9.Seasonal Average : By using this we are able to extract the average Rainfall for each quarter.

Technical Workflow



3. Related Work

Rishi Gupta et al. [21] studied several regions and developed a crop recommendation model using temperature, rainfall, soil, seed, crop yield, humidity, and wind speed data. The data is cleaned using the pandas package before being given into MapReduce for processing. The data is processed in key value pairs in the map function, with the key being region, year, season, and crop, and the value being production per area. The reduction function computes the production per area. MapReduce will produce a super dataset that aggregates all datasets. Lastly, the authors built the recommendation model using K-means clustering.

Monisha et al. [21] published a report comparing various machine learning techniques used in crop yield prediction research. The data is divided into two parts: 80 percent is used to test the model and 20 percent is used to train the model. Once the data has been prepared, it is used to train several models such as linear regression, KNN, Deep learning, XG Boost, K-means clustering based model, random forest classifier, logistic regression, and neural network in order to assess their performance in forecasting the data. According to their findings, the random

forest classifier predicts that data with 99% accuracy, followed by the KNN model with 95% accuracy and XG Boost with 94% accuracy.

This paper was written by Shruthi U, Nagaveni V and Sunil G L in the year 2022. The title of this project is review on the prediction of crop yield using Machine Learning Techniques. So, the aim of this project is to predict the crop yield using the soil properties and rainfall data. This paper has taken samples of different crops from different districts in India. The parameters vary for each crop, but the most used parameters are Rainfall, Soil, Temperature, Area, Production and Yield. In this paper they used supervised and unsupervised machine learning algorithms. Classification and Regression are the techniques used in supervised algorithm and Clustering and association rule are the techniques used in unsupervised algorithm. The result was given in the form of accuracy for each crop. Better accuracy was obtained for the crops which used clustering algorithm.

This paper is written by Swati Vashisht, Praveen K, Munesh C T in the year 2022. The title of this project is Improvised Extreme Learning Machine for Crop Yield Prediction. This article discusses the application of machine learning in crop yield prediction. They have used an extreme learning machine that utilizes Kalman Filter Algorithm for data pre-processing, Linear Discriminant Analysis for feature extraction, and an improved version of Extreme Linear Machine for crop yield prediction. They have only taken the data used to predict the yield of rice crop based on the parameter's geography, season and area of cultivation. Here the accuracy, recall, precision and estimated f1 score are used to evaluate the performance. According to their finding Extreme Learning Machine (ELM) has obtained a higher accuracy i.e., 98.83% than the other CNN & Particle Swarm Optimization with 98.2%, LeNet 98.64% and Deep CNN 97.2% models.

Crop Recommendation System by Atharva Ingle

The crop prediction dataset is used in this notebook to recommend crops. The data includes rainfall, pH, and the presence of nitrogen, phosphorus, and potassium in the soil. In this project, pandas sed to generate the data frame and analyze the data, while the sklearn package is utilized to train and forecast models. Matplotlib is used to visualize data. The author created different models by splitting the data in 80/20 for training and testing, and utilized the Random Forest model, which had the highest accuracy for final prediction.

India Crop Production Analysis by Vasu Adireddy

This dataset is focused on the crop “Arecanut” from the Andaman and Nicobar Island in different monsoons. The data includes Production, Yield, Area, Season, State, District, Crop, Crop year. In this project pandas are used for data pre-processing and NumPy is used for linear algebra while matplotlib is used to visualize the data. This code helps to find in which monsoon the crop grows better.

Reference Project:Crop Yield Prediction. (n.d.). Kaggle.com. Retrieved April 2, 2023, from

Since global economy depends heavily on the agricultural sector. Understanding global agricultural output is essential to addressing issues of food security and minimizing the effects of climate change as the human population continues to grow.

Predicting crop yields is a significant agricultural issue. For making judgments regarding agricultural risk management and generating forecasts for the future, it is vital to understand that

agricultural productivity is primarily influenced by weather conditions (rain, temperature, etc.), pesticides, and reliable information about past crop yield. The fundamental components that keep us alive are identical. We consume large amounts of rice, corn, wheat, and other basic crops. The analysis and forecasting of top 10 most consumed yields all over the world is established by applying machine learning techniques. It consists of 10 most consumed crops. It is a regression problem.

1.**Gathering and Cleaning Data.**

2. **Drop unwanted columns**.

3. **Climate Data: Rainfall**

The climatic factors include rainfall and temperature. They are abiotic components, including pesticides and soil, of the environmental factors that influence plant growth and development.

Rainfall has a dramatic effect on agriculture. For this project rainfall per year information was gathered from World Data Bank.

By using Pandas we can convert average_rain_fall_mm_per_year from object to float.

Next, By dropping any empty rows from dataset and merge yield data frame with rain data frame by year and area columns.

Merge Yield Data frame with rain Data frame by year and area columns

Pesticides Data:

Pesticides used for each item and country was also collected from FAO database.

Data Pre-processing

It is a technique that is used to convert raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for analysis.

Encoding Categorical Variables:

The data frame contains two categorical columns, which are variables that have label values rather than numeric values. The range of available values is frequently constrained to a predetermined set, like in this case, the values for the items and nations. Several machine learning algorithms are unable to directly act on label data. They demand that all input and output variables be numbers.

Thus, categorical data must be transformed into numerical data. One hot encoding method involves transforming categorical information into a format that may be given to ML algorithms to help them perform better at prediction. These two columns will be converted to a one-hot numeric array for that purpose using the One-Hot Encoding method. The categorical value represents the numerical value of the entry in the dataset. This encoding will create a binary column for each category and returns a matrix with the results.

Scaling Features:

Taking a look at the dataset above, it contains features highly varying in magnitudes, units and range. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

To supress this effect, we need to bring all features to the same level of magnitudes. This can be acheived by scaling.

Training Data:

The dataset will be split to two datasets, the training dataset and test dataset. The data is usually tend to be split inequality because training the model usually requires as much data-points as possible.The common splits are 70/30 or 80/20 for train/test.

The training dataset is the intial dataset used to train ML algorithm to learn and produce right predictions. (70% of dataset is training dataset)

The test dataset, however, is used to assess how well ML algorithm is trained with the training dataset. You can't simply reuse the training dataset in the testing stage because ML algorithm will already "know" the expected output, which defeats the purpose of testing the algorithm. (30% of dataset is testing dataset)

The evaluation metric is set based on **R^2 (coefficient of determination)** regression score function, that will represents the proportion of the variance for items (crops) in the regression model. **R^2** score shows how well terms (data points) fit a curve or line.

R^2 is a statistical measure between 0 and 1 which calculates how similar a regression line is to the data it's fitted to. If it's a 1, the model 100% predicts the data variance; if it's a 0, the model predicts none of the variance.

From results viewd above, **Decision Tree Regressor** has the highest R^2 score 0f **96%**, **GradientBoostingRegressor** comes second.

I'll also calculate **Adjusted R^2** also indicates how well terms fit a curve or line, but adjusts for the number of terms in a model. If you add more and more useless variables to a model, adjusted r-squared will decrease. If you add more useful variables, adjusted r-squared will increase. Adjusted R2 will always be less than or equal to R2.

The crop being potatoes has the highest importance in the decision making for the model, where it's the highest crops in the dataset. Cassava too, then as expected we see the effect of

pesticides, where its the third most important feature, and then if the crop is sweet potatoes, we see some of the highest crops in features importance in dataset.

If the crop is grown in India, makes sense since India has the largest crops sum in the dataset. Then comes rainfall and temperature. The first assumption about these features were correct, where they all significantly impact the expected crops yield in the model.At last, we are using matplotlib to display the Graphical values.

K. Sharma and A. S. Rajawat, "Crop Yield Prediction using Hybrid Deep Learning Algorithm for Smart Agriculture," 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS), Coimbatore, India, 2022, pp. 330-335, doi: 10.1109/ICAIS53314.2022.9743001.

Using IoT and cloud computing technology to enhance agricultural processes and boost yield production constitutes smart agriculture. These technologies can be used to automate irrigation, control weather, monitor crops and environmental conditions, and give precise agricultural planning.

Crop production prediction is one of the main uses of smart agriculture. This can be done by combining data from sensors, RFID, GPRS, 3G wireless internet, and other sources to create data that is processed and analyzed by a cloud-based server.

A layered method that includes a front-end layer with sensors and cameras to monitor crop conditions, a gateway layer that delivers data to the cloud, and a back-end layer that enables end-user access to data can be used to execute smart agriculture.

To assess the data gathered from multiple sources and provide insights into crop conditions, including physical and chemical demands, illnesses, and acceptable pesticides and fungicides, machine learning techniques like Deep Learning and Multilayer Perceptron can be utilized.

Installing infrared camera sensors that can distinguish between intruders and farmers on the ground can help farmers safeguard their crops from intruders and pests. Moreover, crop development factors during the harvesting stage, including nutrient data, plant weight, and humidity levels, can be monitored using RFID technology.

By providing real-time information on crop conditions, weather, and other variables that affect yield production, smart agriculture can, in general, assist farmers in producing yields that are successful under a wide range of geographical, environmental, land, and plant variables.

The system for collecting and analyzing agricultural data in real-time to improve crop yield forecast is discussed in the text. The information gathering layer, transportation layer, and application layer are the three levels that make up the system. The information-gathering layer gathers physical data from agricultural sensors and converts it into processable digital data. The sensor layer processes the agricultural data that was collected and summarized by the transportation layer, and the application layer analyses and processes the data to improve crop yield forecast.

For feature extraction and picture retrieval tasks, the system suggests a hybrid deep learning strategy employing a backpropagation neural network and a convolutional neural network. The authors suggest employing a deep CNN model that has already been trained to understand features and show that the suggested strategy yields higher performance than traditional handcrafted features.

Big Data and AI Revolution in Precision Agriculture

To achieve sustainable agriculture production, advanced technologies such as blockchain, IoT, and AI need to be integrated into the agriculture sector. By embracing data-driven agriculture with these technologies, we can take the most promising approach to address current and

future challenges. By collecting enormous amounts of data from farms and thus utilize it in agricultural decision-making. The use of IoT technology in agriculture allows for data collection throughout the various stages of production. As a result, it would be beneficial to analyze the large amounts of data generated during farming, processing, logistics, and marketing using big data analytics.

To acquire field data accurately, IoT devices, remote sensing, and other sensor networks are deployed. The data collected from these networks on soil, crops, weather, and the surrounding environment is stored on either local or cloud storage. ML-based big data algorithms are then used to extract important information from this data and aid farmers in making informed decisions. Based on the recommendations provided by the decision-making system, advanced machinery equipped with an intelligent control system physically executes the required actions. This process continues systematically until the harvesting stage. Software tools for decision-making in precision agriculture are highly valued for their ability to improve management efficiency compared to other tools. However, there is still progress to be made in creating innovation-based tools that are attractive, user-friendly, and intuitive enough for farmers to adopt. Furthermore, producers need to receive appropriate training before these technological tools can be easily managed.

There are many prospects of Big Data and AI in making precision agriculture such as Big Data-Based Decision Support System for Crop Selection, Crop Management, Growth Monitoring, and Produce Quality, Sustainable Use of Resources, Reduce Pesticide Usage, Plant Disease Detection. At the same time it has many challenges such as Data Collection, Big Data Analysis Techniques, Managing Growing Data and Real-Time Scalability.

A data-driven system is beneficial for every stakeholder involved in the agriculture business, from farmers to consumers, financial institutions, food processing industries, and others. While the system's full potential for value generation is yet to be explored, it has already brought about enormous change in the agriculture industry. AI and big data offer a variety of benefits, including

the development of healthier and superior products through new plant genome sequencing techniques, precision agriculture methods that aid in informed decision-making, and the use of IoT sensor devices and analytics techniques to prevent food wastage and food-borne diseases.

Effective use of Big Data in Precision Agriculture

The continuous decline in agricultural growth is a cause for concern, as there was a notable increase in food grain production during the early years of the green revolution. However, due to the limited understanding of science and digital resources, food grain production could not be further improved beyond a certain point. With the rapid advancement of technology, it has become more accessible even in rural areas. Both the government and farmers are collaborating to enhance crop production and support small-scale businesses related to farming. The government has implemented several initiatives, such as agriculture apps, to assist farmers. Through mobile messaging services, farmers can receive information regarding their agricultural queries, access vendor information, weather forecasts, soil content analysis, and locate nearby markets to supply their crop items.

Major issues in the agriculture are weather, soil, crop cutting, pests management and intercropping

Problem Statement: Initially, research was conducted on a limited number of crops, regions, and seasons, focusing on a few parameters such as temperature, soil content, weather conditions, and precipitation. Although this information proved to be useful for the specific regions, expanding the research to cover multiple regions, numerous seasons, and crops that have a direct impact on farmers' livelihoods, along with a wider range of parameters, could benefit a greater number of people and help alleviate the food grain crisis. By increasing the scope of the experiment, more precise predictions can be made. Utilizing big data analytics tools and methodologies alongside recent data can enhance the classification and prediction of information.

Possible Solution: The solution is divided into two phases construction phase and operational phase. Processing of data is done in construction phase and a decision is made in operational phase using decision tree generator algorithm.

Integrating technology with agricultural data can boost crop yield production despite adverse factors such as weather, soil content, pests, and so on. It is necessary to conduct practical experiments on both small and large scales to assess the effectiveness of the technology, and the results obtained can assist farmers in adopting new techniques. The land can be divided into sub-units based on factors such as soil texture, land contour, and water level, and different types of seeds can be sown in each sub-unit depending on the land's variability. This approach can enhance crop yield production.

A. Dataset collection

Analysis has done on the following dataset:

Crops versus Seasonal Production Crops versus Temperature

B. Data preprocessing

In the research WEKA tool has been used for preprocessing of data. Preprocessing of data can be performed on the following way, firstly it split the data into train and test data, and secondly it trains the classifier. Big data analytics tools such as WEKA can be used to perform above operations on data.

C. Decision tree generation

In this module Attribute Selection Measure (ASM) is used**.**

D. Data classification and clustering

Decision tree classification is a supervised learning algorithm. It can solve both regression and classification problems**.**

E. Methodology

In this paper the Linear Regression and Multiple Linear Regression technique of predictive analysis has been used for crop yield prediction.

4. Dataset

4.1. Temperature dataset

Region wise temperature data is taken which is recorded from 1990 to 2022. It has 5 columns, time, tavg, tmin, tmax, percpitation

Time: Date of the day temperature is recorded. It is in the format of dd-MM-yyyy

Tmin: Minimum temperature recorded for a particular day

Tmax: Maximum temperature recorded for a particular day

Tavg: Average of maximum and minimum temperature recorded in a day

Prec: Precipitation recorded in a day

4.2. Rainfall dataset

Region wise rainfall data is taken which is recorded from 1901 to 2015. It has 19 columns,

Subdivision: Name of the place where the data is recorded

year: Year in which the data is recorded

Jan: Rainfall in the month of January in mm

Feb: Rainfall in the month of February in mm

Mar: Rainfall in the month of March in mm

Apr: Rainfall in the month of April in mm

May: Rainfall in the month of May in mm

Jun: Rainfall in the month of June in mm

Jul: Rainfall in the month of July in mm

Aug: Rainfall in the month of August in mm

Sep: Rainfall in the month of September in mm

Oct: Rainfall in the month of October in mm

Nov: Rainfall in the month of November in mm

Dec: Rainfall in the month of December in mm

Annual: Yearly average rainfall recorded in a year

Jan-Feb: Average of rainfall in months of January and February

Mar-May: Average of rainfall in months of March to May

Jun-Sep: Average of rainfall in months of June to September

Oct-Dec: Average of rainfall in months of October to December

4.3. Crop recommendation dataset.

At last, we are obtaining the values of temperature and rainfall. We are comparing them with the Crop Recommendation Dataset and finally we are obtaining the result.

N – Nitrogen Ratios

P – Phosphorous Ratios.

K- Potassium Ratios.

Temperature – Temperature Ratios.

Humidity – Humidity Ratios

PH – Soil PH Values

Rainfall –Rainfall Values

Label – It describes the name of the Crop that we are growing

5. Design of features

5.1. Temperature: As mentioned in the objectives, the temperature could affect the growth of the crop, so this could be measured and used as a feature.

5.2. Rainfall: The amount of rainfall could be measured and used as a feature, as this is important for determining the suitability of the environment for the crop.

5.3. Location: The location where the crop is being grown could also be considered as a feature, as different regions may have different ideal conditions for crop growth.

5.4. Recommendation Dataset: At last, we are obtaining the values of temperature and rainfall. We are comparing them with the Crop Recommendation Dataset.

5.5. Year: We have taken this to collect annual average Temperature and Rainfall values.

5.6. SUBDIVISION: In this feature we are describing the area or a particular Region.

5.7. Annual: By using this we can collect all the month's data by averaging the values.

5.8. Seasonal Average: By using this we can extract the average Rainfall for each quarter.

5.9. Tavg_annual: new feature is added to temperature dataset which is average of temperature recorded in a particular year.

5.10. Tavg_janfeb: new feature is added to temperature dataset which is average of temperature recorded in months of January and February.

5.11. Tavg_marmay: new feature is added to the temperature dataset which is average of temperature recorded in months of March and May.

5.12. Tavg_junsep: new feature is added to the temperature dataset which is an average of temperature recorded in the months of June and September.

5.13. Tavg_octdec: new feature is added to the temperature dataset which is an average of temperature recorded in the months of October and December.

6. Analysis

In this section we are performing the analysis on temperature data set and rainfall data set we have collected both data sets and cleared the null values and obtained the final predicted values based on the year, month and region wise. In the Temperature data set we obtained predicted values based on the year and month this will helped us to do more analysis on the temperature values and In the same way we collected data for the Rainfall from there we have predicted few values. Recommendation Data set In the Recommendation we are already obtained the values of the earlier predicted values now we are going to compare our predicted values from Temperature and Rainfall values with the Recommended Data set. If the provided values nearly match with the Recommended Data set then we are going to get the details which crop is suitable based on the two data sets.

7. Implementation

Since the temperature data has null values and is recorded daily and is missing the columns of annual average and average of the temperature in the seasons, the null values are removed from the dataset using spark. Spark's imputer as shown in figs 7.1 to 7.3, imputer is an estimator that replaces the missing values to do so it might use different modes like replacing null/missing values with mean or median. In our case we have replaced the null values with mean of the column. Similarly the rainfall dataset is cleaned.

After cleaning the datasets, hive tables are created, and the cleaned data is loaded into those tables shown in fig 7.4 and 7.9. Table **temperaturedatasource** has the data of pre-processed temperature data and the **processedrainfalldata** table contains the data of the rainfall. From analysis we found that rainfall data is recorded monthly and has annual average and seasonal average of rainfall which are missing in the temperature dataset and the temperature data is recorded on daily basis sor firstly, the daily temperature data should be converted into monthly data to fill the missing columns (annual average & seasonal average).

To convert the temperature data into monthly data, a query is written as shown in fig 7.2, that selects the time column in format of yyyy-MM and average of temperature of month as **tavg** a new table is created with the quired data. After converting data, the is transformed into same way as rainfall dataset.

Once the dataset is converted, to add new missing columns, a new table **temperaturedata2** is created and using the query in fig 7.7 which computes the average of temperature of all months and stores as **tavg_annual** column and similarly seasonal average of temperature is calculated and stored as **tavg_janfeb** etc.

By joining the **temperaturedata** & **temperaturedata2** as shown in fig 7.8, we get the final temperature dataset which is stored as **processedtemperaturedata** and then joining the the

processedtemperaturedata and **processedrainfalldata** tables as shown in fig 7.10 we get the combined dataset which has the data of rainfall and temperature which has all the features that will be used to train and test the machine learning model.

Since this is a time series problem, we have considered 3 models Linear regression, Decision Tree Classifier and Random Forest Regressor.

Linear Regression:

Linear regression is a machine learning algorithm based on supervised learning. It finishes a regression task. Based on independent variables, regression generates a target prediction value. It is primarily used to determine the relationship between variables and forecasting. The number of independent variables used in regression models varies, as does the type of relationship considered between the dependent and independent variables. The dependent variable in a regression is known by many different names. It is also known as a criterion variable, an outcome variable, an endogenous variable, or a regressand. Exogenous variables, predictor variables, and regressors are other terms for independent variables.

Decision Tree:

A non-supervised machine learning approach used for both classification and regression problems. The algorithm divides the input data into subsets based on the values of the features recursively to arrive at a final prediction or judgment, which is represented by a tree-like structure. At each node of the decision tree, the technique chooses the feature with the greatest information gain. The degree of reduction in entropy or impurity of the dataset caused by splitting the data based on a given feature is referred to as information gain. The procedure is then repeated for each subset of data, with a new branch created for each possible value of that characteristic. The tree can then be used to classify new data by moving from the root to a leaf node and making a final judgment or prediction based on the majority class of the instances in that node for classification tasks or the mean value of the instances for regression tasks.

Random Forest Regressor:

Random Forest is a machine learning algorithm that can be used for classification, regression, and other purposes. It is an ensemble learning method that creates and combines multiple decision trees to make predictions.

A single decision tree can be prone to overfitting or underfitting, resulting in unreliable predictions. Random Forest solves this problem by generating a large number of decision trees, each of which is trained on a subset of the data and employs a subset of the features. This yields a collection of diverse trees, each with its own predictions, and the final prediction is determined by taking the average or majority vote of all the predictions.

To build and train models, firstly the joined data of temperature and rainfall is loaded in a spark data frame then the data of each column is transformed into integer type and the rainfall data is changed from mm to cm so that it can be suitable for all type of models that were considered. Once the data is ready, a vector assembler is created with output column as year since the prediction should be made by year and this vector will be stored in a column named **features**. Next this assembler is applied on the data frame and stored in a new variable **output.** From the output data frame, a new data frame is created from output df for each target column with features and the respective column values. The data in these new data frames are split into training and test data which will be used to train and evaluate the models.

Coming to building of the models, since there are multiple target columns, a training models are created for each of the target column, and they were fitted on to the column's training data. Once the model is trained, it is evaluated on its test data. This process is shown in fig 7.16 to fig 7. below

```
[1] !pip install pyspark

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.3.2.tar.gz (281.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 4.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.5
  Downloading py4j-0.10.9.5-py2.py3-none-any.whl (199 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.7/199.7 KB 19.8 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.3.2-py2.py3-none-any.whl size=281824028 sha256=2687462789de81d518409dc5df132271d4c7dec765f57c557a01277aee6e4db0
  Stored in directory: /root/.cache/pip/wheels/6c/e3/9b/0525ce8a69478916513509d43693511463c6468db0de237c86
Successfully built pyspark
Installing collected packages: py4j, pyspark
  Attempting uninstall: py4j
    Found existing installation: py4j 0.10.9.7
    Uninstalling py4j-0.10.9.7:
      Successfully uninstalled py4j-0.10.9.7
Successfully installed py4j-0.10.9.5 pyspark-3.3.2

#Importing SparkSession from pysparkSQL
from pyspark.sql import SparkSession
from pyspark.ml.feature import Imputer


#Creating an app
spark = SparkSession.builder.appName('Project').getOrCreate()
```

Fig 7.1 - Creating spark application

```
[2] #Importing SparkSession from pysparkSQL
    from pyspark.sql import SparkSession
    from pyspark.ml.feature import Imputer


    #Creating an app
    spark = SparkSession.builder.appName('Project').getOrCreate()

[8] data_frame = spark.read.format("com.dtabricks.spark.csv")\
    .option("mode", "DROPMALFORMED").option("header" ,True)\
    .option("inferschema", True).csv("/content/Chennai_1990_2022.csv")

    data_frame.filter("tavg is null").show()

+----------+----+----+----+----+
|      time|tavg|tmin|tmax|prcp|
+----------+----+----+----+----+
|27-09-1990|null|null|null|null|
|28-09-1990|null|null|null|null|
|20-10-1990|null|null|null|null|
|21-10-1990|null|null|null|null|
|22-10-1990|null|null|null|null|
|24-10-1990|null|null|null|null|
|02-11-1990|null|null|null|null|
|04-11-1990|null|null|null|null|
|05-11-1990|null|null|null|null|
|20-11-1990|null|null|null|null|
|01-12-1990|null|null|null|null|
|02-12-1990|null|null|null|null|
|14-12-1990|null|null|null|null|
|15-12-1990|null|null|null|null|
|21-12-1990|null|null|null|null|
|27-12-1990|null|null|null|null|
```

Fig 7.2 -Creating a data frame from temperature data and displaying null values

```
[10] imputer = Imputer(
         inputCol = data_frame.columns[1],
         outputCol = "tavg"
     )
     data_frame = imputer.fit(data_frame).transform(data_frame)

[11] data_frame.filter("tavg is null").show()

+----+----+----+----+----+
|time|tavg|tmin|tmax|prcp|
+----+----+----+----+----+
+----+----+----+----+----+


[12] data_frame.write.option("header",True).csv("/content/Chennai_Temperature_Processed")
```

Fig 7.3 - Replacing null values with mean of the column

```
hive> create table temperaturedatasource (time string, tavg float, tmin float, tmax float, precp int) r
ow format delimited fields terminated by ',' tblproperties("skip.header.line.count" = "1");
OK
Time taken: 0.279 seconds
hive> load data local inpath '/home/cloudera/Downloads/Chennai_Temperature_data_processed.csv' into tab
le temperaturedatasource;
Loading data to table project.temperaturedatasource
Table project.temperaturedatasource stats: [numFiles=1, totalSize=366919]
OK
Time taken: 0.875 seconds
```

Fig7.4: Creating table with pre-processed temperature data

```
hive> create table chennaidataprocessed as select date_format(time, 'yyyy-MM') as date, round(AVG(tavg)
,2) as tavg_month from temperaturedatasource group by date_format(time, 'yyyy-MM');
Query ID = cloudera_20230404141313_2ab00f87-3f61-4f39-9602-4ef6178ec4ae
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1680633808737_0005, Tracking URL = http://quickstart.cloudera:8088/proxy/application
_1680633808737_0005/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1680633808737_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-04-04 14:13:59,183 Stage-1 map = 0%,  reduce = 0%
2023-04-04 14:14:37,065 Stage-1 map = 67%,  reduce = 0%, Cumulative CPU 8.79 sec
2023-04-04 14:14:38,318 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 9.84 sec
2023-04-04 14:15:16,215 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 17.23 sec
MapReduce Total cumulative CPU time: 17 seconds 230 msec
Ended Job = job_1680633808737_0005
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/chennaidataprocessed
Table project.chennaidataprocessed stats: [numFiles=1, numRows=391, totalSize=5434, rawDataSize=5043]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 17.23 sec   HDFS Read: 376392 HDFS Write: 5521 SUCCE
SS
Total MapReduce CPU Time Spent: 17 seconds 230 msec
OK
```

Fig 7.5: Converting data from daily to monthly

```
hive> create table temperaturedata (year string, tavg_jan float, tavg_feb float, tavg_mar float, tavg_april float, tavg_may float, tavg_jun f
loat, tavg_jul float, tavg_aug float, tavg_sep float, tavg_oct float, tavg_nov float, tavg_dec float) row format delimited fields terminated
by ',' tblproperties("skip.header.line.count" = "1");
OK
Time taken: 0.287 seconds
hive> load data local inpath '/home/cloudera/Downloads/Chennai_monthly_data.csv' into table temperaturedata;
Loading data to table project.temperaturedata
Table project.temperaturedata stats: [numFiles=1, totalSize=2536]
OK
Time taken: 1.449 seconds
```

Fig 7.6: Loading monthly temperature data into new table

```
hive> create table temperaturedata2 as select year as year, round((tavg_jan+tavg_feb+tavg_mar+tavg_april+tavg_may+tavg_jun+tavg_jul+tavg_aug+
tavg_sep+tavg_oct+tavg_nov+tavg_dec)/12, 2) as tavg_annual, round((tavg_jan+tavg_feb)/2, 2) as tavg_janfeb, round((tavg_mar+tavg_april+tavg_m
ay)/3, 2) as tavg_marmay, round((tavg_jun+tavg_jul+tavg_aug+tavg_sep)/4, 2) as tavg_junsep, round((tavg_oct+tavg_nov+tavg_dec)/3, 2) as tavg_
octdec from temperaturedata;
Query ID = cloudera_20230405212727_be21346b-a772-4534-bf2c-3ab26073952c
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1680738764523_0004, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1680738764523_0004/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1680738764523_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2023-04-05 21:27:50,670 Stage-1 map = 0%,  reduce = 0%
2023-04-05 21:28:29,703 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 6.86 sec
MapReduce Total cumulative CPU time: 6 seconds 860 msec
Ended Job = job_1680738764523_0004
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/.hive-staging_hive_2023-04-05_21-27-07_455_6853626673760093137
-1/-ext-10001
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/temperaturedata2
Table project.temperaturedata2 stats: [numFiles=1, numRows=32, totalSize=1107, rawDataSize=1075]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 6.86 sec   HDFS Read: 10404 HDFS Write: 1189 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 860 msec
OK
Time taken: 84.594 seconds
```

Fig 7.7: creating a new table with the annual average and seasonal average

```
hive> create table processedtemperaturedata as select temperaturedata.*, temperaturedata2.tavg_annual, temperaturedata2.tavg_janfeb, temperat
uredata2.tavg_marmay, temperaturedata2.tavg_junsep, temperaturedata2.tavg_octdec from temperaturedata join temperaturedata2 on temperaturedat
a2.year = temperaturedata.year;
Query ID = cloudera_20230405214545_f04d6826-57ac-4517-a58f-2b7f9c55f25f
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20230405214545_f04d6826-57ac-4517-a58f-2b7f9c55f25f.log
2023-04-05 09:46:16     Starting to launch local task to process map join;      maximum memory = 1013645312
2023-04-05 09:46:24     Dump the side-table for tag: 1 with group count: 32 into file: file:/tmp/cloudera/fd7eced1-6eaf-4ef2-ad84-4cddc0c83a8
a/hive_2023-04-05_21-45-47_679_4767242412784442918-1/-local-10003/HashTable-Stage-4/MapJoin-mapfile31--.hashtable
2023-04-05 09:46:24     Uploaded 1 File to: file:/tmp/cloudera/fd7eced1-6eaf-4ef2-ad84-4cddc0c83a8a/hive_2023-04-05_21-45-47_679_476724241278
4442918-1/-local-10003/HashTable-Stage-4/MapJoin-mapfile31--.hashtable (2284 bytes)
2023-04-05 09:46:24     End of local task; Time Taken: 8.43 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1680738764523_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1680738764523_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1680738764523_0007
Hadoop job information for Stage-4: number of mappers: 1; number of reducers: 0
2023-04-05 21:47:20,101 Stage-4 map = 0%,  reduce = 0%
```

Fig 7.8: Joining the monthly data table and the missing columns data table

```
hive> create table processedrainfalldata (state string, year string, rf_jan float, rf_feb float, rf_mar float, rf_apr
float, rf_may float, rf_jun float, rf_jul float, rf_aug float, rf_sep float, rf_oct float, rf_nov float, rf_dec float,
 rf_annual float, rf_janfeb float, rf_marmay float, rf_junsep float, rf_octdec float) row format delimited fields term
inated by ',' tblproperties("skip.header.line.count" = "2");
OK
Time taken: 7.357 seconds
hive>
```

Fig 7.9: Loading pre-processed rainfall data into a new table

```
hive> create table processedjoindata as select processedtemperaturedata.*, processedrainfalldata.rf_jan, processedrainfalldata.rf_feb, processedrain
falldata.rf_mar, processedrainfalldata.rf_apr, processedrainfalldata.rf_may, processedrainfalldata.rf_jun, processedrainfalldata.rf_jul, processedra
infalldata.rf_aug, processedrainfalldata.rf_sep, processedrainfalldata.rf_oct, processedrainfalldata.rf_nov, processedrainfalldata.rf_dec, processed
rainfalldata.rf_annual, processedrainfalldata.rf_janfeb, processedrainfalldata.rf_marmay, processedrainfalldata.rf_junsep, processedrainfalldata.rf_
octdec from processedtemperaturedata join processedrainfalldata on processedrainfalldata.rf_year = processedtemperaturedata.year;
Query ID = cloudera_20230406091717_93992ab7-0a63-43c0-934d-9ca63b8da9b4
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20230406091717_93992ab7-0a63-43c0-934d-9ca63b8da9b4.log
2023-04-06 09:18:07     Starting to launch local task to process map join;      maximum memory = 1013645312
2023-04-06 09:18:15     Dump the side-table for tag: 0 with group count: 32 into file: file:/tmp/cloudera/d725e801-6ff2-4431-b1d7-9d48d3156aac/hive_
2023-04-06_09-17-37_489_5579736576664872275-1/-local-10003/HashTable-Stage-4/MapJoin-mapfile00--.hashtable
2023-04-06 09:18:15     Uploaded 1 File to: file:/tmp/cloudera/d725e801-6ff2-4431-b1d7-9d48d3156aac/hive_2023-04-06_09-17-37_489_5579736576664872275
-1/-local-10003/HashTable-Stage-4/MapJoin-mapfile00--.hashtable (3894 bytes)
2023-04-06 09:18:15     End of local task; Time Taken: 7.861 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1680796583419_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1680796583419_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1680796583419_0001
Hadoop job information for Stage-4: number of mappers: 1; number of reducers: 0
2023-04-06 09:19:20,506 Stage-4 map = 0%,  reduce = 0%
2023-04-06 09:20:01,346 Stage-4 map = 100%,  reduce = 0%, Cumulative CPU 6.89 sec
MapReduce Total cumulative CPU time: 6 seconds 890 msec
Ended Job = job_1680796583419_0001
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/processedjoindata
Table project.processedjoindata stats: [numFiles=1, numRows=26, totalSize=5077, rawDataSize=5051]
MapReduce Jobs Launched:
Stage-Stage-4: Map: 1   Cumulative CPU: 6.89 sec   HDFS Read: 25352 HDFS Write: 5160 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 890 msec
OK
Time taken: 148.587 seconds
```

Fig 7.10: Joining the processed temperature and rainfall tables to get combined data table

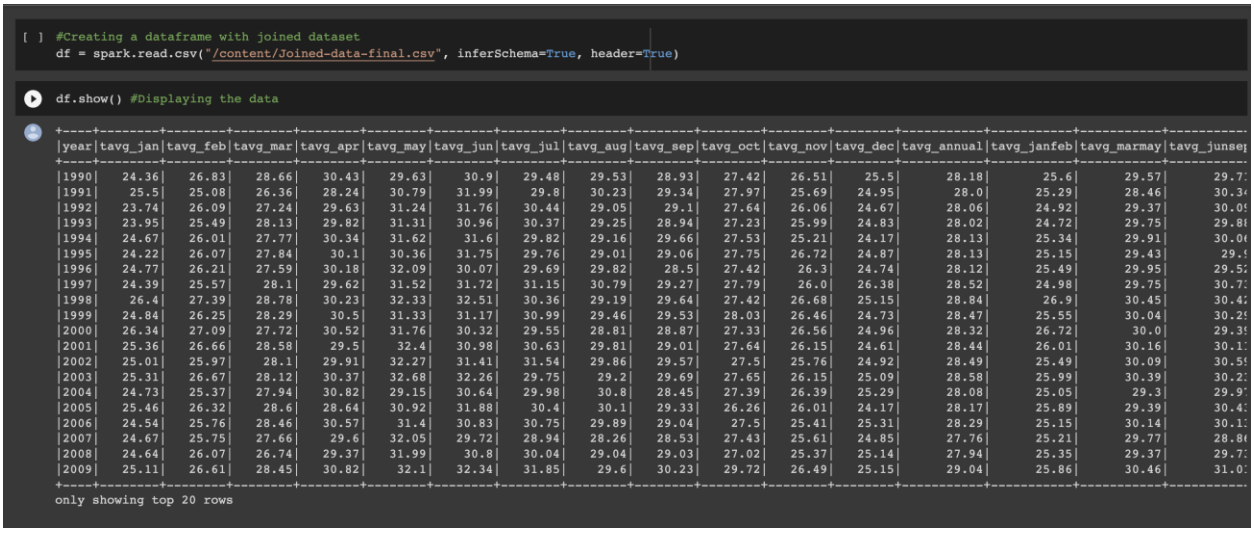
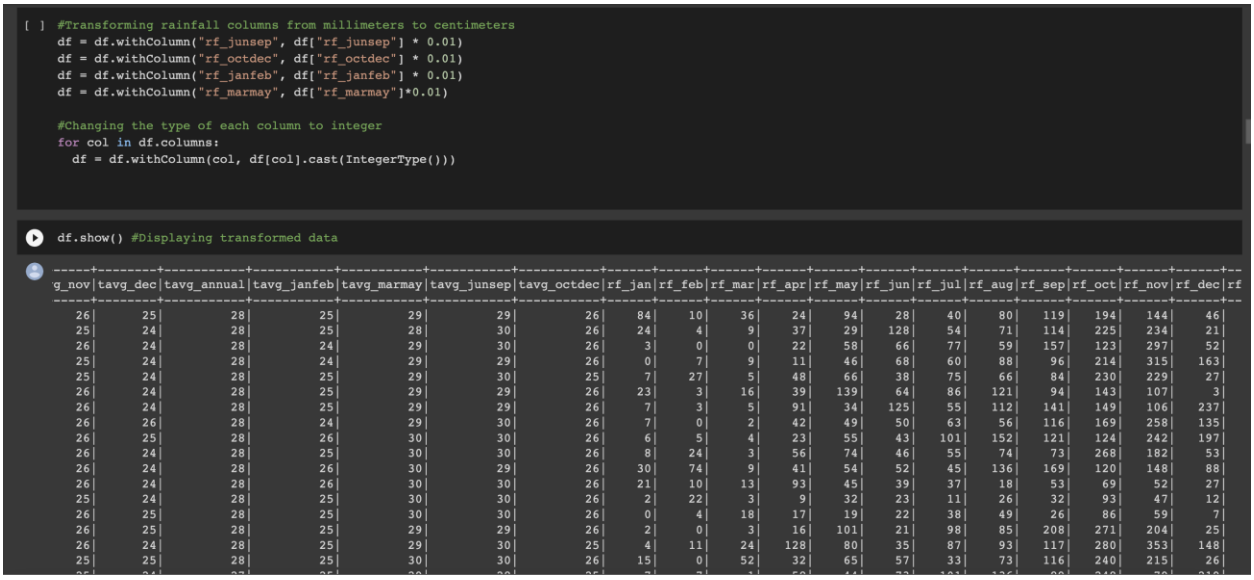


Fig 7.11: Creating a data frame with the joined dataset



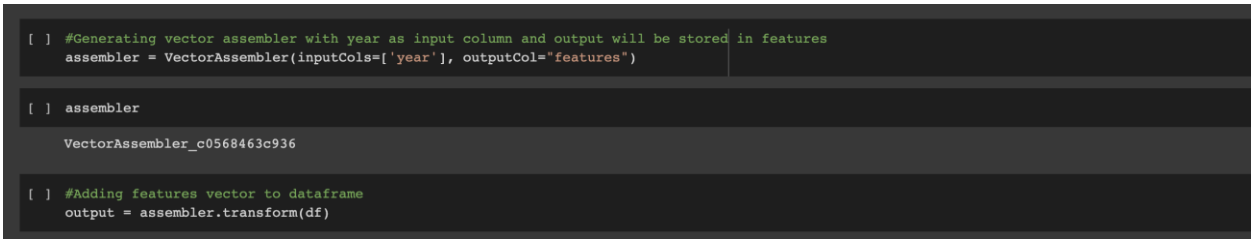
```
[ ] #Transforming rainfall columns from millimeters to centimeters
    df = df.withColumn("rf_junsep", df["rf_junsep"] * 0.01)
    df = df.withColumn("rf_octdec", df["rf_octdec"] * 0.01)
    df = df.withColumn("rf_janfeb", df["rf_janfeb"] * 0.01)
    df = df.withColumn("rf_marmay", df["rf_marmay"]*0.01)

    #Changing the type of each column to integer
    for col in df.columns:
      df = df.withColumn(col, df[col].cast(IntegerType()))
```

```
df.show() #Displaying transformed data
```

```
+-----+--------+-----------+-----------+-----------+-----------+-----------+------+------+------+------+------+------+------+------+------+------+------+------+--
g_nov|tavg_dec|tavg_annual|tavg_janfeb|tavg_marmay|tavg_junsep|tavg_octdec|rf_jan|rf_feb|rf_mar|rf_apr|rf_may|rf_jun|rf_jul|rf_aug|rf_sep|rf_oct|rf_nov|rf_dec|rf
+-----+--------+-----------+-----------+-----------+-----------+-----------+------+------+------+------+------+------+------+------+------+------+------+------+--
   26|      25|         28|         25|         29|         29|         26|    84|    10|    36|    24|    94|    28|    40|    80|   119|   194|   144|    46|
   25|      24|         28|         25|         28|         30|         26|    24|     4|     9|    37|    29|   128|    54|    71|   114|   225|   234|    21|
   26|      24|         28|         24|         29|         30|         26|     3|     0|     0|    22|    58|    66|    77|    59|   157|   123|   297|    52|
   25|      24|         28|         24|         29|         29|         26|     0|     7|     9|    11|    46|    68|    60|    88|    96|   214|   315|   163|
   25|      24|         28|         25|         29|         30|         25|     7|    27|     5|    48|    66|    38|    75|    66|    84|   230|   229|    27|
   26|      24|         28|         25|         29|         29|         26|    23|     3|    16|    39|   139|    64|    86|   121|    94|   143|   107|     3|
   26|      24|         28|         25|         29|         29|         26|     7|     3|     5|    91|    34|   125|    55|   112|   141|   149|   106|   237|
   26|      26|         28|         24|         29|         30|         26|     7|     0|     2|    42|    49|    50|    63|    56|   116|   169|   258|   135|
   26|      25|         28|         26|         30|         30|         26|     6|     5|     4|    23|    55|    43|   101|   152|   121|   124|   242|   197|
   26|      24|         28|         25|         30|         30|         26|     8|    24|     3|    56|    74|    46|    55|    74|    73|   268|   182|    53|
   26|      24|         28|         26|         30|         29|         26|    30|    74|     9|    41|    54|    52|    45|   136|   169|   120|   148|    88|
   26|      24|         28|         26|         30|         30|         26|    21|    10|    13|    93|    45|    39|    37|    18|    53|    69|    52|    27|
   25|      24|         28|         25|         30|         30|         26|     2|    22|     3|     9|    32|    23|    11|    26|    32|    93|    47|    12|
   26|      25|         28|         25|         30|         30|         26|     0|     4|    18|    17|    19|    22|    38|    49|    26|    86|    59|     7|
   26|      25|         28|         25|         29|         29|         26|     2|     0|     3|    16|   101|    21|    98|    85|   208|   271|   204|    25|
   26|      24|         28|         25|         29|         30|         25|     4|    11|    24|   128|    80|    35|    87|    93|   117|   280|   353|   148|
   25|      25|         28|         25|         30|         30|         26|    15|     0|    52|    32|    65|    57|    33|    73|   116|   240|   215|    26|
```

Fig 7.12: Transforming the columns in data frame



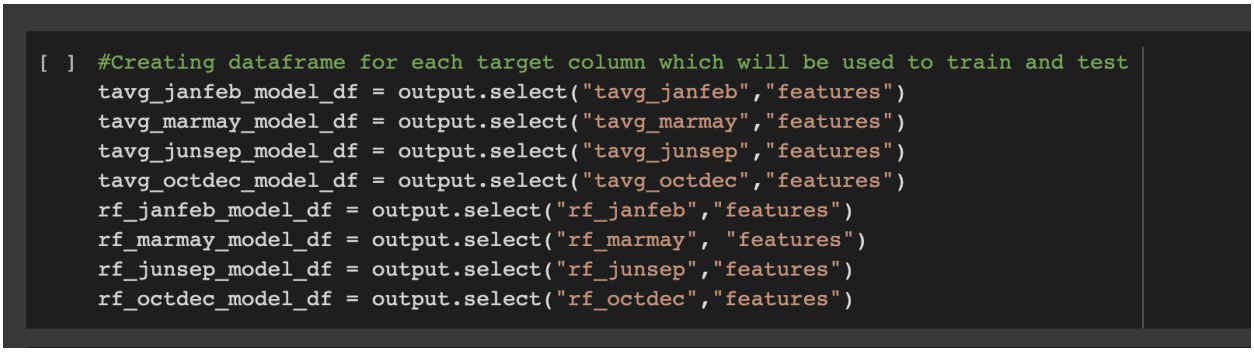
```
[ ] #Generating vector assembler with year as input column and output will be stored in features
    assembler = VectorAssembler(inputCols=['year'], outputCol="features")
```

```
[ ] assembler
```

```
VectorAssembler_c0568463c936
```

```
[ ] #Adding features vector to dataframe
    output = assembler.transform(df)
```

Fig 7.13: Generating vector assembler with year as input column



```
[ ] #Creating dataframe for each target column which will be used to train and test
    tavg_janfeb_model_df = output.select("tavg_janfeb","features")
    tavg_marmay_model_df = output.select("tavg_marmay","features")
    tavg_junsep_model_df = output.select("tavg_junsep","features")
    tavg_octdec_model_df = output.select("tavg_octdec","features")
    rf_janfeb_model_df = output.select("rf_janfeb","features")
    rf_marmay_model_df = output.select("rf_marmay", "features")
    rf_junsep_model_df = output.select("rf_junsep","features")
    rf_octdec_model_df = output.select("rf_octdec","features")
```

Fig 7.14: creating data frames from output of vector assembler

```
[ ]  #Splitting data in each training dataframe into taining and test data
     tavg_janfeb_training_df, tavg_janfeb_test_df = tavg_janfeb_model_df.randomSplit([0.7, 0.3])
     tavg_marmay_training_df, tavg_marmay_test_df = tavg_marmay_model_df.randomSplit([0.7, 0.3])
     tavg_junsep_training_df, tavg_junsep_test_df = tavg_junsep_model_df.randomSplit([0.6, 0.4])
     tavg_octdec_training_df, tavg_octdec_test_df = tavg_octdec_model_df.randomSplit([0.6, 0.4])
     training_rf_janfeb, test_rf_janfeb = rf_janfeb_model_df.randomSplit([0.7, 0.3])
     training_rf_marmay, test_rf_marmay = rf_marmay_model_df.randomSplit([0.7, 0.3])
     rf_junsep_training_df, rf_junsep_test_df = rf_junsep_model_df.randomSplit([0.7, 0.3])
     rf_octdec_training_df, rf_octdec_test_df = rf_octdec_model_df.randomSplit([0.7, 0.3])
```

Fig 7.15: Splitting data into training and test data

Linear Regression

```
✓ 0s [572] #Creating instance of linear regression model for each target column
     tavg_janfeb_lr = LinearRegression(featuresCol= "features", labelCol="tavg_janfeb")
     tavg_marmay_lr = LinearRegression(featuresCol= "features", labelCol="tavg_marmay")
     tavg_junsep_lr = LinearRegression(featuresCol= "features", labelCol="tavg_junsep")
     tavg_octdec_lr = LinearRegression(featuresCol= "features", labelCol="tavg_octdec")
     lr_janfeb = LinearRegression(featuresCol= "features", labelCol="rf_janfeb")
     lr_marmay = LinearRegression(featuresCol= "features", labelCol="rf_marmay")
     rf_junsep_lr = LinearRegression(featuresCol= "features", labelCol="rf_junsep")
     rf_octdec_lr = LinearRegression(featuresCol= "features", labelCol="rf_octdec")
```

```
✓ 2s [573] #Fitting training data on models created above
     tavg_janfeb_trained_model = tavg_janfeb_lr.fit(tavg_janfeb_training_df)
     tavg_marmay_trained_model = tavg_marmay_lr.fit(tavg_marmay_training_df)
     tavg_junsep_trained_model = tavg_junsep_lr.fit(tavg_junsep_training_df)
     tavg_octdec_trained_model = tavg_octdec_lr.fit(tavg_octdec_training_df)
     trained_model_janfeb = lr_janfeb.fit(training_rf_janfeb)
     trained_model_marmay = lr_marmay.fit(training_rf_marmay)
     rf_junsep_trained_model = rf_junsep_lr.fit(rf_junsep_training_df)
     rf_octdec_trained_model = rf_octdec_lr.fit(rf_octdec_training_df)
```

Fig 7.16: Creating linear regression models and fitting them on training data

```
[615] #Evaluating linear regression models
      tavg_janfeb_results = tavg_janfeb_trained_model.evaluate(tavg_janfeb_training_df)
      tavg_marmay_results = tavg_marmay_trained_model.evaluate(tavg_marmay_training_df)
      results_tavg_junsep = tavg_junsep_trained_model.evaluate(tavg_junsep_training_df)
      results_tavg_octdec = tavg_octdec_trained_model.evaluate(tavg_octdec_training_df)
      results_rf_janfeb = trained_model_janfeb.evaluate(training_rf_janfeb)
      results_rf_marmay = trained_model_marmay .evaluate(training_rf_marmay)
      rf_junsep_results = rf_junsep_trained_model.evaluate(rf_junsep_training_df)
      rf_octdec_results = rf_octdec_trained_model.evaluate(rf_octdec_training_df)

[616] #Displaying R squared value
      print("R squared for tavg janfeb",tavg_janfeb_results.r2)
      print("R squared for tavg marmay",tavg_marmay_results.r2)
      print("R squared for tavg junsep",results_tavg_junsep.r2)
      print("R squared for tavg octdec",results_tavg_octdec.r2)
      print(rf_junsep_results.r2)
      print(rf_octdec_results.r2)
      print(results_rf_janfeb.r2)
      print(results_rf_marmay.r2)

      R squared for tavg janfeb 0.005921504544565415
      R squared for tavg marmay 0.29168556904080567
      R squared for tavg junsep 0.0617929620034493
      R squared for tavg octdec 0.001787994891443101
      0.006727854437513225
      0.0017827431061324672
      0.02002877479933185
      0.002553722760289756
```

Fig 7.17: evaluating linear regression models and displaying R squared values

```
[618] #Selecting features of each test data
      tavg_janfeb_unlabeled_data = tavg_janfeb_test_df.select("features")
      tavg_janfeb_unlabeled_data.show()
      tavg_marmay_unlabeled_data = tavg_marmay_test_df.select("features")
      tavg_marmay_unlabeled_data.show()
      tavg_junsep_unlabeled_data = tavg_junsep_test_df.select("features")
      tavg_junsep_unlabeled_data.show()
      tavg_octdec_unlabeled_data = tavg_octdec_test_df.select("features")
      tavg_octdec_unlabeled_data.show()
      rf_janfeb_unlabeled_data = test_rf_janfeb.select("features")
      rf_marmay_unlabeled_data = test_rf_marmay.select("features")
      rf_janfeb_unlabeled_data.show()
      rf_marmay_unlabeled_data.show()
      rf_junsep_unlabeled_data = rf_junsep_test_df.select("features")
      rf_junsep_unlabeled_data.show()
      rf_octdec_unlabeled_data = rf_octdec_test_df.select("features")
      rf_octdec_unlabeled_data.show()

      |[1990.0]|
      |[1991.0]|
      |[1992.0]|
      |[1993.0]|
      |[1999.0]|
      |[2001.0]|
      |[2003.0]|
      |[2004.0]|
      |[2005.0]|
      |[2007.0]|
      |[2010.0]|
```

Fig 7.18: Selecting features of each test data and displaying the results

```
[619] #Trasforming test data on trained models to get predictions
      tavg_janfeb_predictions = tavg_janfeb_trained_model.transform(tavg_janfeb_unlabeled_data)
      tavg_marmay_predictions = tavg_marmay_trained_model.transform(tavg_marmay_unlabeled_data)
      tavg_junsep_predictions = tavg_junsep_trained_model.transform(tavg_junsep_unlabeled_data)
      tavg_octdec_predictions = tavg_octdec_trained_model.transform(tavg_octdec_unlabeled_data)
      rf_janfeb_predictions = trained_model_janfeb.transform(rf_janfeb_unlabeled_data)
      rf_marmay_predictions = trained_model_marmay.transform(rf_marmay_unlabeled_data)
      rf_junsep_predictions = rf_junsep_trained_model.transform(rf_junsep_unlabeled_data)
      rf_octdec_predictions = rf_octdec_trained_model.transform(rf_octdec_unlabeled_data)

[620] #Displaying predictions
      tavg_janfeb_predictions.show()
      tavg_marmay_predictions.show()
      tavg_junsep_predictions.show()
      tavg_octdec_predictions.show()
      rf_janfeb_predictions.show()
      rf_marmay_predictions.show()
      rf_junsep_predictions.show()
      rf_octdec_predictions.show()

      +--------+------------------+
      |features|        prediction|
      +--------+------------------+
      |[1992.0]|24.988394893754467|
      |[1994.0]|24.998559366122095|
      |[1996.0]| 25.00872383848972|
      |[2012.0]| 25.09003961743072|
      +--------+------------------+
```

Fig 7.19: transforming test data on trained models to get predictions and displaying the resulting predictions.

Decision Tree

```
#Creating instances of decision tree classifier for targeted columns training data
tavg_janfeb_df_classifier = DecisionTreeClassifier(labelCol="tavg_janfeb").fit(tavg_janfeb_training_df)
tavg_marmay_df_classifier = DecisionTreeClassifier(labelCol="tavg_marmay").fit(tavg_marmay_training_df)
tavg_junsep_df_classifier = DecisionTreeClassifier(labelCol="tavg_junsep").fit(tavg_junsep_training_df)
tavg_octdec_df_classifier = DecisionTreeClassifier(labelCol="tavg_octdec").fit(tavg_octdec_training_df)
rf_janfeb_df_classifier = DecisionTreeClassifier(labelCol="rf_janfeb").fit(training_rf_janfeb)
rf_marmay_df_classifier = DecisionTreeClassifier(labelCol="rf_marmay").fit(training_rf_marmay)
rf_junsep_classifier = DecisionTreeClassifier(labelCol="rf_junsep").fit(rf_junsep_training_df)
rf_octdec_classifier = DecisionTreeClassifier(labelCol="rf_octdec").fit(rf_octdec_training_df)

#Transforming decision tree models on respective test data
tavg_janfeb_df_predictions = tavg_janfeb_df_classifier.transform(tavg_janfeb_test_df)
tavg_marmay_df_predictions = tavg_marmay_df_classifier.transform(tavg_marmay_test_df)
tavg_junsep_df_predictions = tavg_junsep_df_classifier.transform(tavg_junsep_test_df)
tavg_octdec_df_predictions = tavg_octdec_df_classifier.transform(tavg_octdec_test_df)
df_predictions_janfeb = rf_janfeb_df_classifier.transform(test_rf_janfeb)
df_predictions_marmay = rf_marmay_df_classifier.transform(test_rf_marmay)
rf_junsep_predictions = rf_junsep_classifier.transform(rf_junsep_test_df)
rf_octdec_predictions = rf_octdec_classifier.transform(rf_octdec_test_df)
```

Fig 7.20: Creating instances of decision tree classifier for targeted columns training data and transforming decision tree models on respective test data

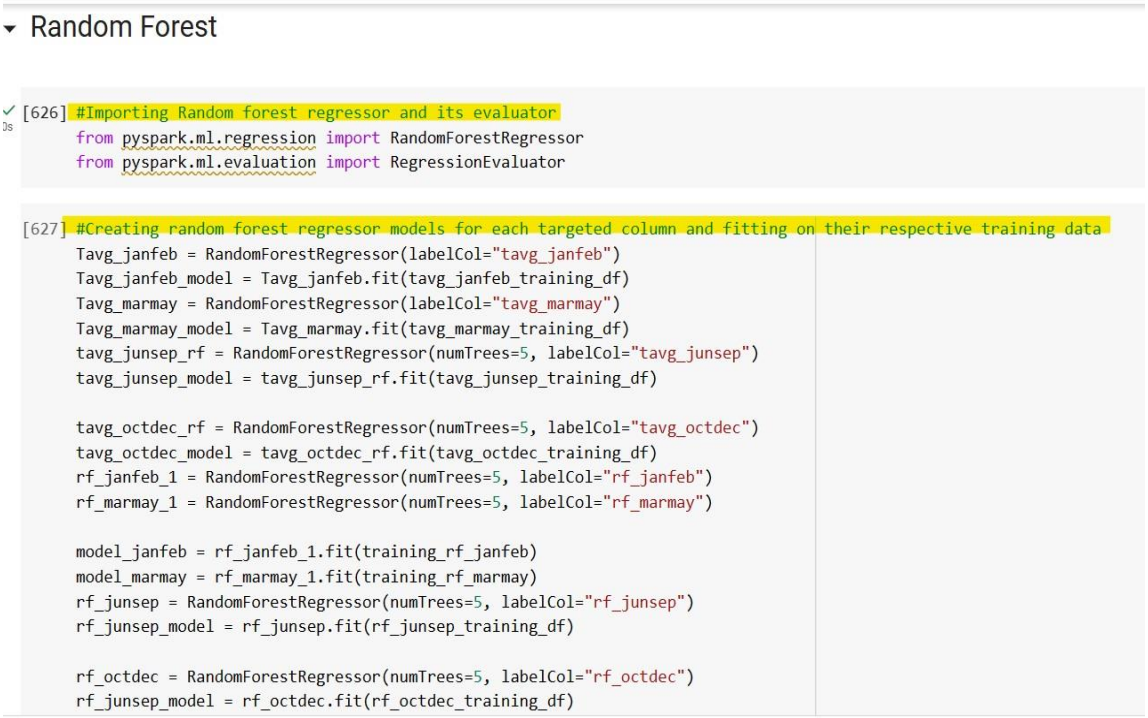


Fig 7.21: importing random forest regressor and evaluator and creating random forest regressor models for each targeted column and fitting on their respective training dataset.

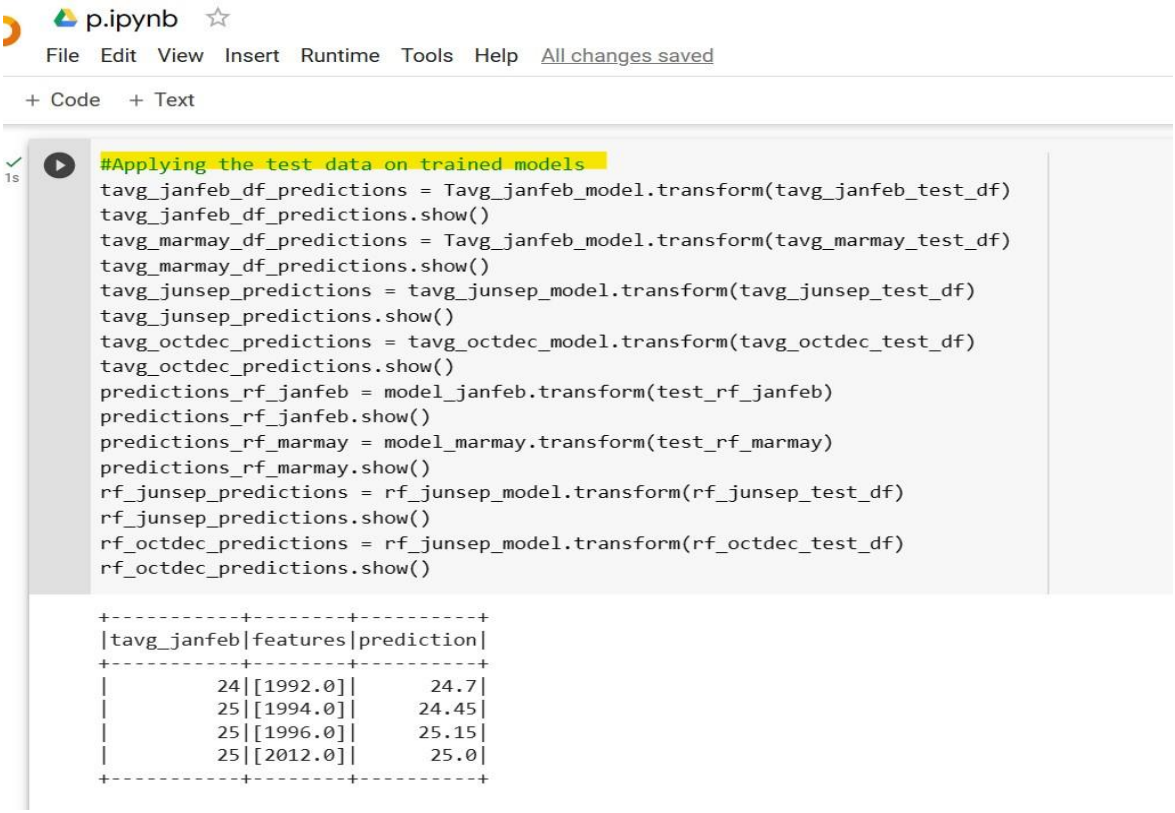


Fig 7.22: Applying the test data on trained random forest models

```
#Displaying RMSE value for each RF model
tavg_janfeb_model_evaluator = RegressionEvaluator(
    labelCol="tavg_janfeb", metricName="rmse")
print("Root Mean Squared Error for tavg_janfeb_model = %g" % tavg_janfeb_model_evaluator.evaluate(tavg_janfeb_df_predictions))
tavg_marmay_model_evaluator = RegressionEvaluator(
    labelCol="tavg_marmay", metricName="rmse")
print("Root Mean Squared Error for tavg_marmay_model = %g" % tavg_marmay_model_evaluator.evaluate(tavg_marmay_df_predictions))
tavg_junsep_model_evaluator = RegressionEvaluator(
    labelCol="tavg_junsep", metricName="rmse")
print("Root Mean Squared Error for tavg_junsep_model = %g" % tavg_junsep_model_evaluator.evaluate(tavg_junsep_predictions))
tavg_octdec_model_evaluator = RegressionEvaluator(
    labelCol="tavg_octdec", metricName="rmse")
print("Root Mean Squared Error for tavg_octdec_model = %g" % tavg_octdec_model_evaluator.evaluate(tavg_octdec_predictions))
rf_janfeb_evaluator = RegressionEvaluator(labelCol="rf_janfeb", metricName="rmse")
print("Root Mean Squared Error for rf_janfeb_evaluator = %g" % rf_janfeb_evaluator.evaluate(predictions_rf_janfeb))
rf_marmay_evaluator = RegressionEvaluator(labelCol="rf_marmay", metricName="rmse")
print("Root Mean Squared Error for rf_marmay_evaluator = %g" % rf_marmay_evaluator.evaluate(predictions_rf_marmay))
rf_junsep_model_evaluator = RegressionEvaluator(
    labelCol="rf_junsep", metricName="rmse")
print("Root Mean Squared Error (RMSE) on rf_junsep_model_evaluator= %g" %rf_junsep_model_evaluator.evaluate(rf_junsep_predictions) )
rf_octdec_model_evaluator = RegressionEvaluator(
    labelCol="rf_octdec", metricName="rmse")
print("Root Mean Squared Error (RMSE) on rf_octdec_model_evaluator= %g" %rf_octdec_model_evaluator.evaluate(rf_octdec_predictions) )
```

Fig 7.23: Displaying root mean squared error value for each random forest model

8. Preliminary Results

Visualization of rainfall data as scatter graph

From the chart in fig 8.1, we can see that the rainfall is higher in the months of October and November and is low from February to June.

Visualization of temperature data

From the chart in fig 8.2, we can see that the temperature is higher in months of march, April, May, June, August and October and was mostly higher in June.

Figs 8.3 & 8.5 are the heat maps of the monthly rainfall and temperature data, which display how the values are ranging and figs 8.4 & 8.6 are the heat maps of seasonal rainfall and temperature data

Heats maps are useful for correlating the data of rainfall and temperature and finding patterns between 2 datasets as we can see that from aug to

Evaluation of the linear regression models

For evaluation of the predicted results, we use R2 (R squared) value. R-squared is a goodness-of-fit metric for linear regression models. This statistic displays the percentage of the dependent variable's variance that the independent variables account for collectively. R-squared employs a simple 0-100% scale to assess the strength of the relationship between your model and the dependent variable. Here the R squared values are very less hence it is not an ideal model for our dataset.

Evaluation of the decision tree models

For evaluation of the predicted results, 2 metrics are used

1. Accuracy is defined as the ratio of correct predictions to total predictions made by the model.
2. Precision is defined as the ratio of true positives to the total of true positives and false positives. Precision assesses the model's ability to identify positive instances correctly.

As we can see from fig, the accuracy and precision on the test data is around 0.8 for most of the target columns so decision tree can be a choice.

Evaluation of the random forest models

For evaluation of the predicted results, RMSE value is considered, RMSE means Root Mean Squared Error It is used to evaluate the accuracy of the regression models. It measures the distance between the predicted value and the true value. Lower the value means better performance. Here the RMSE is low for all the target columns so Random forest regressor is ideal choice for the considered dataset

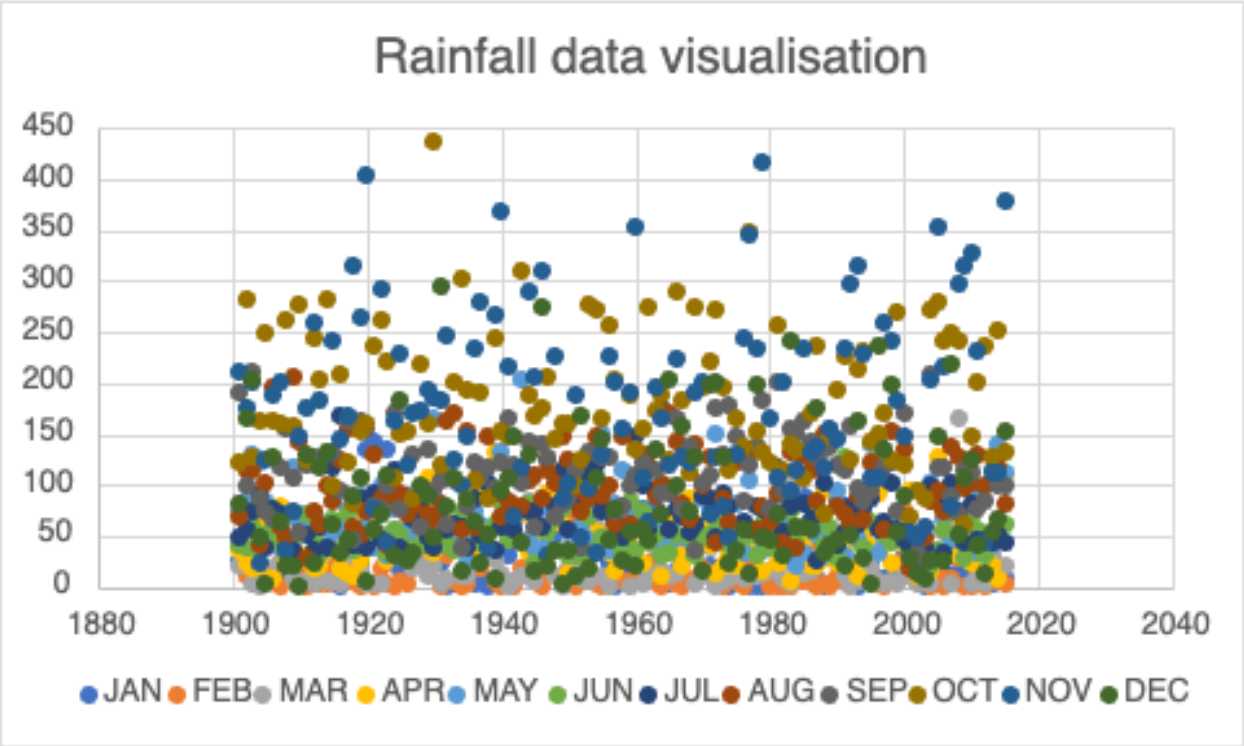


Fig 8.1 - Visualization of rainfall data as scattered chart

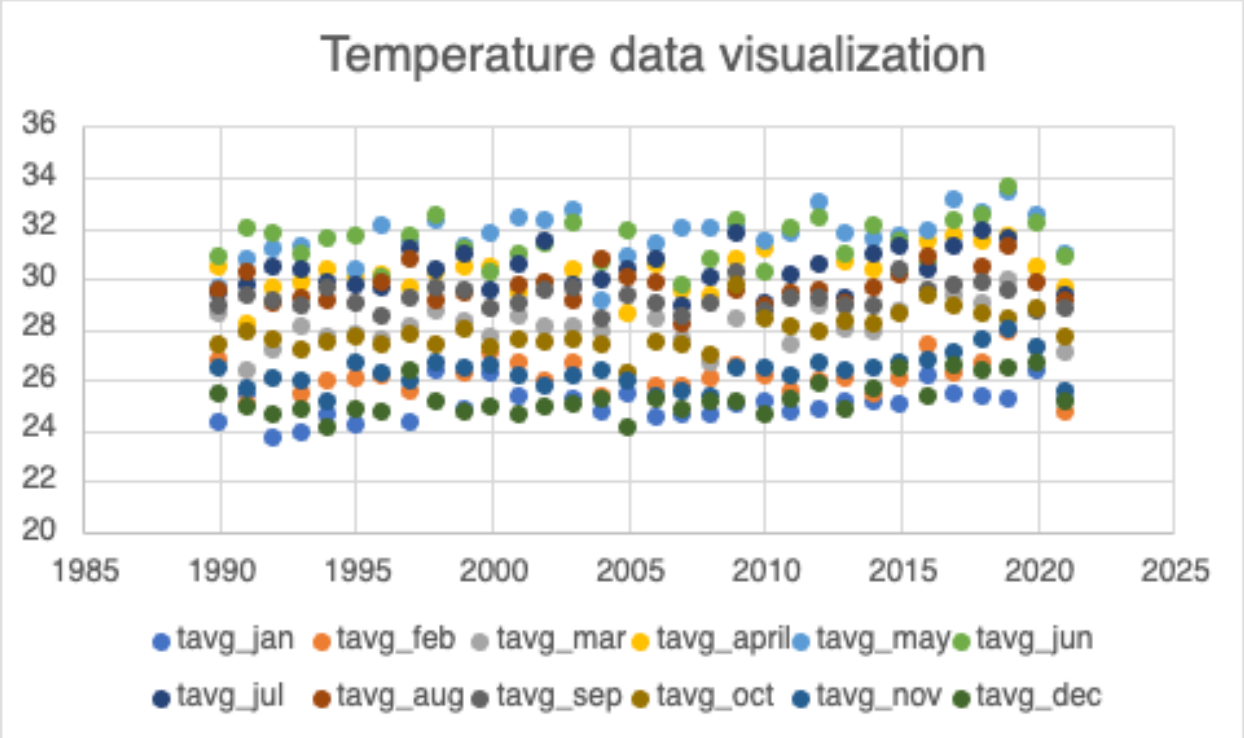


Fig 8.2 - Visualization of temperature data

year	tavg_jan	tavg_feb	tavg_mar	tavg_apr	tavg_may	tavg_jun	tavg_jul	tavg_aug	tavg_sep	tavg_oct	tavg_nov	tavg_dec
1990	24.36	26.83	28.66	30.43	29.63	30.9	29.48	29.53	28.93	27.42	26.51	25.5
1991	25.5	25.08	26.36	28.24	30.79	31.99	29.8	30.23	29.34	27.97	25.69	24.95
1992	23.74	26.09	27.24	29.63	31.24	31.76	30.44	29.05	29.1	27.64	26.06	24.67
1993	23.95	25.49	28.13	29.82	31.31	30.96	30.37	29.25	28.94	27.23	25.99	24.83
1994	24.67	26.01	27.77	30.34	31.62	31.6	29.82	29.16	29.66	27.53	25.21	24.17
1995	24.22	26.07	27.84	30.1	30.36	31.75	29.76	29.01	29.06	27.75	26.72	24.87
1996	24.77	26.21	27.59	30.18	32.09	30.07	29.69	29.82	28.5	27.42	26.3	24.74
1997	24.39	25.57	28.1	29.62	31.52	31.72	31.15	30.79	29.27	27.79	26	26.38
1998	26.4	27.39	28.78	30.23	32.33	32.51	30.36	29.19	29.64	27.42	26.68	25.15
1999	24.84	26.25	28.29	30.5	31.33	31.17	30.99	29.46	29.53	28.03	26.46	24.73
2000	26.34	27.09	27.72	30.52	31.76	30.32	29.55	28.81	28.87	27.33	26.56	24.96
2001	25.36	26.66	28.58	29.5	32.4	30.98	30.63	29.81	29.01	27.64	26.15	24.61
2002	25.01	25.97	28.1	29.91	32.27	31.41	31.54	29.86	29.57	27.5	25.76	24.92
2003	25.31	26.67	28.12	30.37	32.68	32.26	29.75	29.2	29.69	27.65	26.15	25.09
2004	24.73	25.37	27.94	30.82	29.15	30.64	29.98	30.8	28.45	27.39	26.39	25.29
2005	25.46	26.32	28.6	28.64	30.92	31.88	30.4	30.1	29.33	26.26	26.01	24.17
2006	24.54	25.76	28.46	30.57	31.4	30.83	30.75	29.89	29.04	27.5	25.41	25.31
2007	24.67	25.75	27.66	29.6	32.05	29.72	28.94	28.26	28.53	27.43	25.61	24.85
2008	24.64	26.07	26.74	29.37	31.99	30.8	30.04	29.04	29.03	27.02	25.37	25.14
2009	25.11	26.61	28.45	30.82	32.1	32.34	31.85	29.6	30.23	29.72	26.49	25.15
2010	25.15	26.14	28.74	31.18	31.48	30.25	29.09	28.94	28.74	28.46	26.54	24.66
2011	24.74	25.59	27.46	29.48	31.79	31.97	30.2	29.46	29.3	28.1	26.14	25.23
2012	24.85	25.97	28.91	30.62	33.04	32.45	30.6	29.6	29.23	27.96	26.71	25.87
2013	25.14	26.08	28.05	30.64	31.82	31.01	29.26	29.06	28.98	28.3	26.43	24.86
2014	25.12	25.51	27.92	30.42	31.61	32.16	30.95	29.71	28.92	28.22	26.45	25.63
2015	25.08	26.04	28.72	30.24	31.71	31.54	31.32	30.2	30.33	28.69	26.65	26.45

Fig 8.3 - Heatmap of monthly temperature data

year	tavg_janfeb	tavg_marmay	tavg_junsep	tavg_octdec
1990	25.6	29.57	29.71	26.48
1991	25.29	28.46	30.34	26.2
1992	24.92	29.37	30.09	26.12
1993	24.72	29.75	29.88	26.02
1994	25.34	29.91	30.06	25.64
1995	25.15	29.43	29.9	26.45
1996	25.49	29.95	29.52	26.15
1997	24.98	29.75	30.73	26.72
1998	26.9	30.45	30.42	26.42
1999	25.55	30.04	30.29	26.41
2000	26.72	30	29.39	26.28
2001	26.01	30.16	30.11	26.13
2002	25.49	30.09	30.59	26.06
2003	25.99	30.39	30.23	26.3
2004	25.05	29.3	29.97	26.36
2005	25.89	29.39	30.43	25.48
2006	25.15	30.14	30.13	26.07
2007	25.21	29.77	28.86	25.96
2008	25.35	29.37	29.73	25.84
2009	25.86	30.46	31.01	27.12
2010	25.65	30.47	29.25	26.55
2011	25.17	29.58	30.23	26.49
2012	25.41	30.86	30.47	26.85
2013	25.61	30.17	29.58	26.53
2014	25.32	29.98	30.43	26.77
2015	25.56	30.22	30.85	27.26

Fig 8.4 - Heatmap of seasonal temperature data

year	rf_jan	rf_feb	rf_mar	rf_apr	rf_may	rf_jun	rf_jul	rf_aug	rf_sep	rf_oct	rf_nov	rf_dec
1990	84.8	10.2	36.2	24.3	94.9	28.9	40	80.1	119.6	194.1	144	46.1
1991	24.2	4.6	9.7	37.4	29.5	128.1	54.7	71.7	114	225.1	234.4	21.4
1992	3.1	0.2	0	22.5	58.3	66.4	77.9	59.9	157.3	123.9	297.2	52.1
1993	0.1	7	9.9	11.2	46.6	68.2	60.3	88.8	96.7	214.5	315.9	163.1
1994	7	27.6	5.5	48.3	66.6	38.6	75.3	66.1	84.2	230.6	229.1	27
1995	23.8	3.3	16.7	39.5	139	64.6	86.7	121.1	94	143.1	107.9	3.7
1996	7.6	3.6	5	91.5	34	125.3	55.4	112.4	141.2	149.7	106	237.6
1997	7.7	0.2	2.3	42	49.2	50.4	63.5	56.4	116	169.5	258.1	135.2
1998	6.9	5.7	4.1	23.6	55.8	43	101.9	152.9	121.8	124.7	242	197.4
1999	8.3	24.1	3.9	56.9	74.5	46.4	55.3	74.8	73.1	268.9	182.8	53.6
2000	30.3	74.5	9.4	41.6	54.7	52.2	45.8	136.1	169.8	120.3	148.6	88.9
2001	21.8	10.9	13	93.8	45	39.9	37.2	18.7	53.8	69.5	52.3	27.6
2002	2.9	22.1	3.9	9.2	32.6	23.7	11.8	26.5	32.2	93.7	47.2	12.2
2003	0.3	4.1	18	17.1	19.8	22.5	38.7	49.3	26.1	86	59.5	7.1
2004	2.8	0.7	3	16.2	101.1	21.2	98.6	85.1	208.3	271.1	204.5	25
2005	4.1	11.1	24.4	128	80.6	35.7	87.9	93.3	117.9	280.5	353.4	148.5
2006	15.3	0.2	52.4	32.6	65.2	57.2	33.6	73.4	116.3	240.4	215.1	26.1
2007	7.2	7.5	1.8	58	44.9	73.1	101	136.5	89.1	248.9	79.2	219.9
2008	11.7	29.1	164.7	31.5	53.7	51.1	73.1	126.5	70.7	242.7	298.5	50.1
2009	7.9	0	41	41.4	74.8	27	42.1	96.9	114.4	62.1	314.7	106.2
2010	11.8	0.2	1.9	22.9	91.9	70	81.4	102.9	111.1	148.1	328.6	124.5
2011	4.3	11.2	8	91.5	33.4	56	45.5	128.9	76	200.4	230.5	41
2012	3	0.1	2.5	35.5	41.9	30.1	46.5	98	84.9	235.2	44.5	14
2013	3.9	30.9	30	20.3	42	54.6	42.7	110.7	113.5	127.9	112.3	53.2
2014	7.4	6.1	8.1	8.3	139.1	47.8	50.6	117.7	98.9	252.2	110.8	66
2015	8.3	2.3	21.7	108.8	112.4	62.4	43.5	81.6	98.4	132.6	379.8	152.8

Fig 8.5 - Heatmap of monthly rainfall data

year	tavg_janfeb	rf_marmay	rf_junsep	rf_octdec
1990	95	155.3	268.7	384.2
1991	28.9	76.6	368.5	480.9
1992	3.3	80.8	361.4	473.2
1993	7.1	67.7	314	693.5
1994	34.6	120.5	264.2	486.6
1995	27.1	195.2	366.4	254.7
1996	11.3	130.5	434.3	493.3
1997	7.9	93.5	286.3	562.8
1998	12.5	83.6	419.6	564
1999	32.3	135.3	249.6	505.3
2000	104.7	105.6	403.9	357.8
2001	32.7	151.8	149.5	149.3
2002	25	45.7	94.2	153.1
2003	4.4	54.8	136.6	152.6
2004	3.5	120.3	413.2	500.6
2005	15.2	233	334.8	782.3
2006	15.5	150.2	280.6	481.6
2007	14.7	104.7	399.7	548
2008	40.8	250	321.5	591.2
2009	7.9	157.2	280.4	482.9
2010	11.9	116.7	365.4	601.2
2011	15.5	132.8	306.4	471.8
2012	3.1	79.9	259.5	293.6
2013	34.8	92.2	321.5	293.4
2014	13.4	155.5	315.1	428.9
2015	10.6	242.8	285.9	665.3

Fig 8.6 - Heatmap of seasonal rainfall data

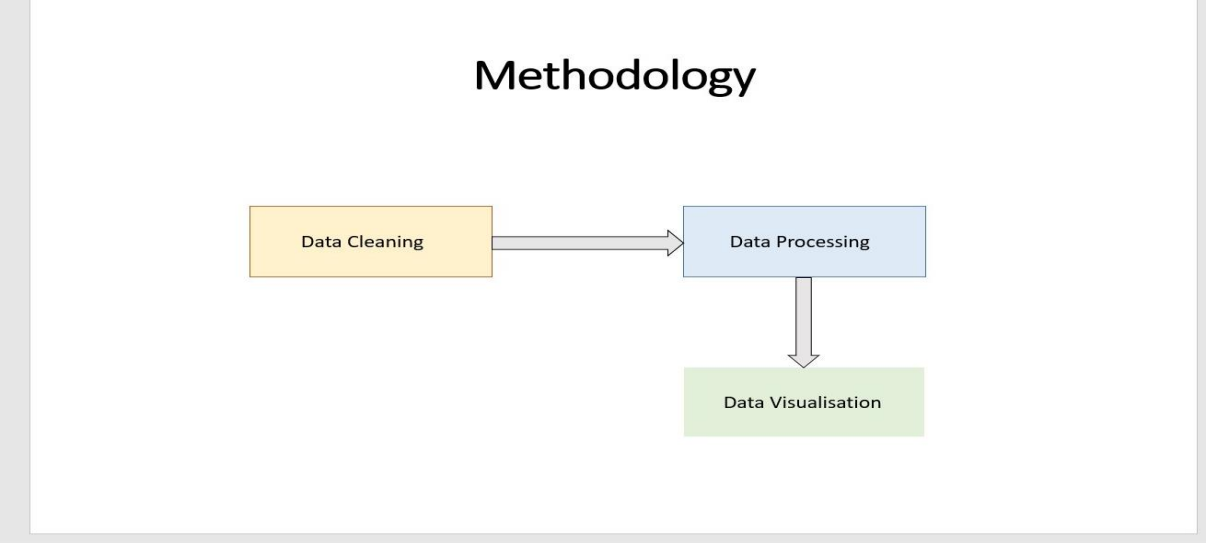


Fig 8.7 - Methodology steps

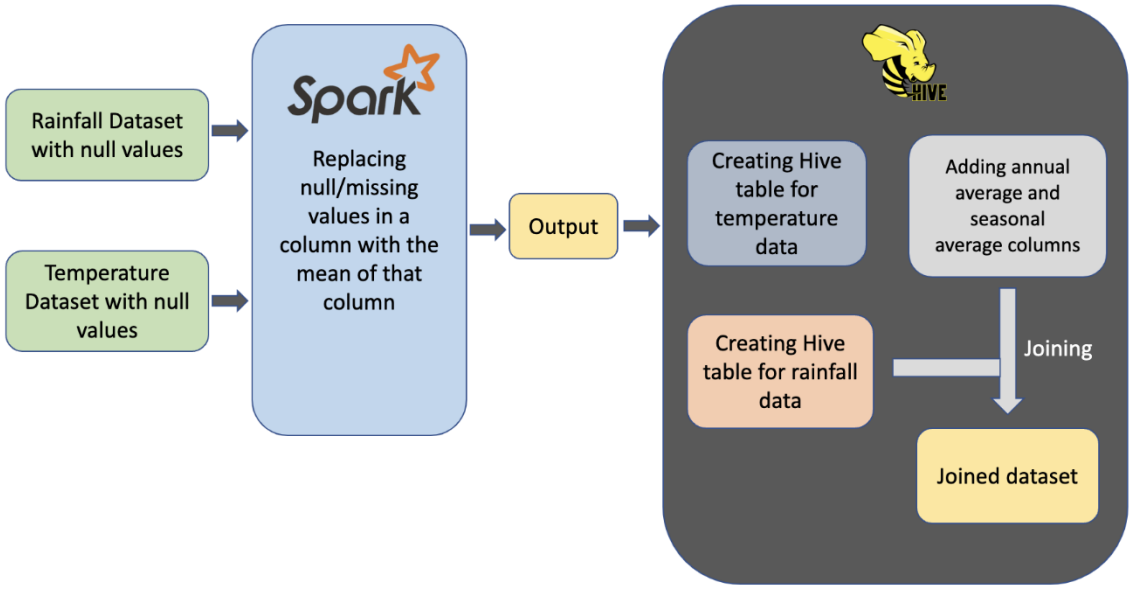


Fig 8.8 - Workflow

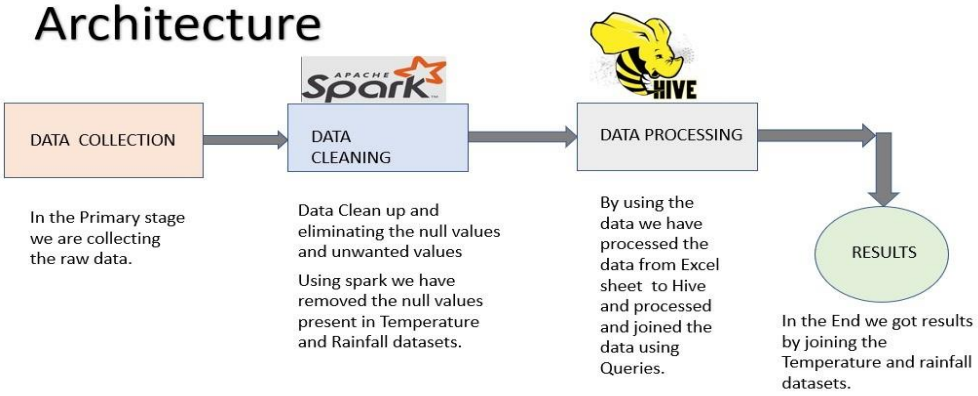


Fig 8.9 - Architecture

```
[617] #Displaying mean square error for linear regression models created
      print("Mean square error for tavg_janfeb: ", tavg_janfeb_results.meanSquaredError)
      print("Mean square error for tavg_marmay: ", tavg_marmay_results.meanSquaredError)
      print("Mean square error for tavg_junsep: ", results_tavg_junsep.meanSquaredError)
      print("Mean square error for tavg_octdec: ", results_tavg_octdec.meanSquaredError)
      print("Mean square error of rf_janfeb: ", results_rf_janfeb.meanSquaredError)
      print("Mean square error of rf_marmay: ", results_rf_marmay.meanSquaredError)
      print("Mean square error for rf_junsep: ", rf_junsep_results.meanSquaredError)
      print("Mean square error for rf_octdec: ", rf_octdec_results.meanSquaredError)

Mean square error for tavg_janfeb:  0.22387304959636836
Mean square error for tavg_marmay:  0.23561699046369913
Mean square error for tavg_junsep:  0.4158258340150909
Mean square error for tavg_octdec:  0.18716475095785434
Mean square error of rf_janfeb:  0.0695837556355504
Mean square error of rf_marmay:  0.3723799435028251
Mean square error for rf_junsep:  0.5587155818788987
Mean square error for rf_octdec:  2.0019648032996122
```

Fig 9.0: Displaying mean square error for linear regression models created for few columns

```
#Displaying predictions
tavg_janfeb_df_predictions.show()
tavg_marmay_df_predictions.show()
tavg_junsep_df_predictions.show()
tavg_octdec_df_predictions.show()
df_predictions_janfeb.show()
df_predictions_marmay.show()
rf_junsep_predictions.show()
rf_octdec_predictions.show()
```

```
+-----------+--------+--------------------+--------------------+----------+
|tavg_janfeb|features|       rawPrediction|         probability|prediction|
+-----------+--------+--------------------+--------------------+----------+
|         24|[1992.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      25.0|
|         25|[1994.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      24.0|
|         25|[1996.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      25.0|
|         25|[2012.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      25.0|
+-----------+--------+--------------------+--------------------+----------+

+-----------+--------+--------------------+--------------------+----------+
|tavg_marmay|features|       rawPrediction|         probability|prediction|
+-----------+--------+--------------------+--------------------+----------+
|         29|[1990.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      28.0|
|         29|[2011.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      30.0|
|         30|[1998.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      29.0|
|         30|[2002.0]|[0.0,0.0,0.0,0.0,...|[0.0,0.0,0.0,0.0,...|      30.0|
+-----------+--------+--------------------+--------------------+----------+
```

Fig 9.1: Displaying predictions for the decision tree model

```
[624] #Evaluating accuracy of decision tree models
tavg_janfeb_df_accuracy = MulticlassClassificationEvaluator(labelCol="tavg_janfeb", metricName="accuracy").evaluate(tavg_janfeb_df_predictions)
print("tavg_janfeb_df_accuracy: ", tavg_janfeb_df_accuracy)
tavg_marmay_df_accuracy = MulticlassClassificationEvaluator(labelCol="tavg_marmay", metricName="accuracy").evaluate(tavg_marmay_df_predictions)
print("tavg_marmay_df_accuracy: ", tavg_marmay_df_accuracy)
tavg_junsep_df_accuracy = MulticlassClassificationEvaluator(labelCol="tavg_junsep", metricName="accuracy").evaluate(tavg_junsep_df_predictions)
print("tavg_junsep Accuracy: ", tavg_junsep_df_accuracy)
tavg_octdec_df_accuracy = MulticlassClassificationEvaluator(labelCol="tavg_octdec", metricName="accuracy").evaluate(tavg_octdec_df_predictions)
print("tavg_octdec Accuracy: ", tavg_octdec_df_accuracy)
df_accuracy_janfeb = MulticlassClassificationEvaluator(labelCol="rf_janfeb", metricName="accuracy").evaluate(df_predictions_janfeb)
df_accuracy_marmay = MulticlassClassificationEvaluator(labelCol="rf_marmay", metricName="accuracy").evaluate(df_predictions_marmay)
print("Accuracy: ", df_accuracy_janfeb)
print("Accuracy: ", df_accuracy_marmay)
rf_junsep_accuracy = MulticlassClassificationEvaluator(labelCol="rf_junsep", metricName="accuracy").evaluate(rf_junsep_predictions)
print("Accuracy: ", rf_junsep_accuracy)
rf_octdec_accuracy = MulticlassClassificationEvaluator(labelCol="rf_octdec", metricName="accuracy").evaluate(rf_octdec_predictions)
print("Accuracy: ", rf_octdec_accuracy)
```

```
tavg_janfeb_df_accuracy:  0.5
tavg_marmay_df_accuracy:  0.25
tavg_junsep Accuracy:  0.2857142857142857
tavg_octdec Accuracy:  0.2
Accuracy:  0.9230769230769231
Accuracy:  0.36363636363636365
Accuracy:  0.1
Accuracy:  0.2857142857142857
```

Fig 9.2: Evaluating accuracy of decision tree models and displaying its results

```
[625] #Displaying precision for decision tree models
      tavg_janfeb_dt_precision = MulticlassClassificationEvaluator(labelCol="tavg_janfeb", metricName="weightedPrecision").evaluate(tavg_janfeb_df_predictions)
      print("Precision: ",tavg_janfeb_dt_precision)
      tavg_marmay_dt_precision = MulticlassClassificationEvaluator(labelCol="tavg_marmay", metricName="weightedPrecision").evaluate(tavg_marmay_df_predictions)
      print("Precision: ",tavg_marmay_dt_precision)
      tavg_junsep_dt_precision = MulticlassClassificationEvaluator(labelCol="tavg_junsep", metricName="weightedPrecision").evaluate(tavg_junsep_df_predictions)
      print("Precision: ", tavg_junsep_dt_precision)
      tavg_octdec_dt_precision = MulticlassClassificationEvaluator(labelCol="tavg_octdec", metricName="weightedPrecision").evaluate(tavg_octdec_df_predictions)
      print("Precision: ", tavg_octdec_dt_precision)
      rf_junsep_dt_precision = MulticlassClassificationEvaluator(labelCol="rf_junsep", metricName="weightedPrecision").evaluate(rf_junsep_predictions)
      print("Precision: ", rf_junsep_dt_precision)
      dt_precision_janfeb = MulticlassClassificationEvaluator(labelCol="rf_janfeb", metricName="weightedPrecision").evaluate(df_predictions_janfeb)
      dt_precision_marmay = MulticlassClassificationEvaluator(labelCol="rf_marmay", metricName="weightedPrecision").evaluate(df_predictions_marmay)
      print("Precision: ", dt_precision_janfeb)
      print("Precision: ", dt_precision_marmay)
      rf_octdec_dt_precision = MulticlassClassificationEvaluator(labelCol="rf_octdec", metricName="weightedPrecision").evaluate(rf_octdec_predictions)
      print("Precision: ", rf_octdec_dt_precision)

      Precision:  0.5
      Precision:  0.25
      Precision:  0.38095238095238093
      Precision:  0.5333333333333333
      Precision:  0.03333333333333333
      Precision:  1.0
      Precision:  0.47619047619047616
      Precision:  0.17142857142857143
```

Fig 9.3: Displaying precision for decision tree models

```
Root Mean Squared Error for tavg_janfeb_model = 0.70297
Root Mean Squared Error for tavg_marmay_model = 4.38504
Root Mean Squared Error for tavg_junsep_model = 0.811284
Root Mean Squared Error for tavg_octdec_model = 0.340588
Root Mean Squared Error for rf_janfeb_evaluator = 0.333333
Root Mean Squared Error for rf_marmay_evaluator = 0.418197
Root Mean Squared Error (RMSE) on rf_junsep_model_evaluator= 2.29468
Root Mean Squared Error (RMSE) on rf_octdec_model_evaluator= 2.11926
```

Fig 9.4: These are the results of the root mean squared error from the **fig7.23** code

9. Project Management

9.1. Implementation Status Report

Work completed

Task	Description	Contribution
Collecting datasets	Searching and gathering data related to rainfall and temperature	Surya and Anirudh
Analysis of Rainfall Data set	Analyzing the rainfall dataset and its features	All team members contributed equally
Analysis of Temperature Dataset	Analyzing the temperature dataset and its features	All team members contributed equally
Cleaning datasets	Replacing all the null values from the datasets with mean of the respective columns	Aditya and Mahaboob Ali
Processing Data sets	After cleaning the null values, we processed data by using hive.	Aditya and Mahaboob Ali
Merging the Rainfall and Temperature Data set values	We are merging the rainfall and temperature data sets by using the Joins.	Mahaboob Ali

9.2. Work to be completed

Task	Description	Contribution
Build the model	During this Phase we are building the models for predicting the values.	All team members contributed equally

Fitting training data on models	By using the models, we are going to train the models on joined dataset	All team members contributed equally
Predict the Data	By the End stage, we will predict the values from trained models.	All team members contributed equally

10. References

Best Fertilizer for Wheat: Organic, NPK and Application. (2022, February 23). Agri Farming. https://www.agrifarming.in/best-fertilizer-for-wheat-organic-npk-and-application

https://www.kaggle.com/code/atharvaingle/what-crop-to-grow#About-the-data

https://ieeexplore.ieee.org/document/9557312

https://ieeexplore.ieee.org/document/9510048

https://ieeexplore.ieee.org/document/9864482

https://ieeexplore.ieee.org/document/9853054/authors#authors

India_Crop_Production_Analysis | Kaggle

Big Data and AI Revolution in Precision Agriculture

https://ieeexplore.ieee.org/document/9743001

https://www.kaggle.com/code/kushagranull/crop-yield-prediction/notebook

Effective use of Big Data in Precision Agriculture

https://www.kaggle.com/datasets/rajanand/rainfall-in-india/discussion/132834

https://www.kaggle.com/datasets/vanvalkenberg/historicalweatherdataforindiancities

https://notebook.community/MingChen0919/learning-apache-spark/notebooks/ipynb/RandomForest