

OPTIMIZING CROP IRRIGATION USING MACHINE LEARNING ALGORITHM-BASED SOIL MOISTURE MONITORING

ABSTRACT

Efficient water management in agriculture is paramount for sustainable crop production, especially in regions facing water scarcity and environmental stressors. This study proposes a novel approach for optimizing crop irrigation practices by leveraging machine learning algorithms based on real-time soil moisture monitoring.

The core of this methodology involves deploying sensor networks within agricultural fields to continuously measure soil moisture levels at various depths. These data are then integrated into a machine learning framework, which analyses historical patterns and current environmental conditions to predict optimal irrigation schedules.

Key components of the proposed system include data preprocessing techniques to handle noise and outliers, feature selection methods to identify relevant variables affecting soil moisture, and the implementation of advanced machine learning algorithms such as decision trees, random forests, or neural networks.

By utilizing machine learning algorithms, the system can adaptively adjust irrigation schedules based on factors such as soil type, crop type, weather conditions, and water availability. The proposed system aims to minimise water wastage while maximising crop yield by delivering water precisely when and where it is needed.

The effectiveness of the proposed methodology will be demonstrated through field experiments and comparative analyses with traditional irrigation methods. The outcomes of this research have the potential to revolutionize agricultural water management practices, leading to increased efficiency, sustainability, and resilience in crop production systems.

1. INTRODUCTION

The world knows emergency problems, foremost among them being the scarcity of fresh water. Around 70% of freshwater used in agricultural activities such as irrigation and supply of nutrients for plant growth. The demand for fresh water is escalating with the increasing food demand of fast-growing population. However, traditional irrigation methods often result in water wastage, reduced crop yields and environmental degradation. The key to efficient irrigation lies in accurately monitoring soil moisture

levels, which is a complex task due to dynamic nature of soil properties and weather conditions.

Machine learning algorithms, combined with advanced sensor technologies, offer a promising solution to optimise crop irrigation. The project focuses on leveraging the K-Nearest Neighbours' (KNN) algorithm to predict requirement of water supply to the crops, monitoring soil moisture based on historical data.

The KNN algorithm is a simple, yet powerful, non-parametric method used for classification and regression tasks. By evaluating the similarities between new piece of land. And previously observed land, KNN can provide insights into the likelihood of water supply. This method is particularly advantageous due to simplicity and intuitive nature of its approach.

The primary objective of this project is to develop a predictive model using KNN that can effectively classify a given land into two categories: dry or wet. By analysing various features of given land, such as moisture, temperature and other relevant factors, the model aims to assist farmers in making more informed water managing decisions.

This project will cover the following key aspects:

- Defining the goal
- Collection of raw data
- Processing and cleaning of data.
- Exploratory data analysis.
- Model building.
- Model evaluation.
- Insights and interpretation.

1.1 MACHINE LEARNING

1.1.1 What is Machine Learning?

Machine Learning (ML) is a field of study in Artificial Intelligence (AI) concerned with the development and study of statistical algorithms that can learn from data and generalised to unseen data, and thus perform tasks without explicit instructions.

Machine learning is an application of AI that provides system the ability to automatically learn and improve from experience without being explicitly programmed.

1.1.2 Classification of Machine Learning Algorithms:

Machine learning algorithms are broadly classified into three types.

- i. Supervised Learning Algorithms.

- ii. Unsupervised Learning Algorithms.
- iii. Reinforcement Learning Algorithms.

i. Supervised Learning Algorithms:

Supervised learning is a type of machine learning in which the machine needs external supervision to learn. The supervised learning models are trained using the labelled dataset. Once the training and processing are done, the model is tested by providing a sample test data to check whether it predicts the correct output.

The goal of supervised learning is to map input data with the output data. Supervised learning is based on supervision, and it same as when a student learns things in the teacher's supervision.

Supervised learning can be divided further into two categories of problem:

- Classification
- Regression

Examples of some popular supervised learning algorithms are Simple Linear Regression, Decision tree, Logistic Regression, KNN algorithm, etc.

ii. Unsupervised Learning Algorithms:

It is a type of machine learning algorithm in which machine does not need any external supervision to learn from the data, hence called unsupervised learning. The unsupervised models can be trained using unlabelled dataset that is not classified, nor categorized, and the algorithm needs to act on that data without any supervision. In unsupervised learning, the model doesn't have a predefined output, and it tries to find useful insights from the huge amount of data. These are further classified into two categories of problems:

- Clustering.
- Association.

Examples of some unsupervised learning algorithms are K-means clustering, Apriori algorithm, Eclat etc.

iii. Reinforcement Learning Algorithms:

In Reinforcement learning, an agent interacts with its environment by producing actions, and learn with the help of feedback. The feedback is given to the agent in the form of rewards, such as for each good action, he gets a positive reward, and for each

bad action, he gets a negative reward. There is no supervision provided to the agent. **Q-Learning** is used in reinforcement learning.

1.2 Types of Machine Learning Algorithms:

1.2.1. Linear Regression:

Linear regression is one of the most fundamental and widely used algorithms in machine learning and statistics. It is supervised learning technique used for modelling the relationship between a dependent variable and one or more independent variables. The primary goal of linear regression is to predict the value of the dependent variable based on the values of the independent variables. Some common types of linear regression are simple linear regression, multiple linear regression, ridge regression, polynomial regression, etc.

The simple linear regression model is given by,

$$Y = \beta_0 + \beta_1 * x$$

Y – Response Variable

β_0 – Intercept

β_1 – Slope

x – Regressor Variable

1.2.2 Decision Tree:

A decision tree is a versatile and powerful machine learning algorithm used for both classification and regression tasks. A decision tree models the decision logics i.e., tests and corresponds outcomes for classifying data items into tree-like structure. The nodes of a decision tree normally have multiple levels where the first and top-most node is called the root node. All internal nodes represent tests on input variables or attributes. Depending on the test outcome, the classification algorithm branches towards the appropriate child node where the process of test and branching repeats until it reaches the leaf node. The leaf or terminal nodes corresponds to the decision. Its modes decisions and their possible consequences as a tree-like structure of nodes and branches, making it easy to interpret and understand. Decision trees are easy to visualize and interpret. The decision-making process is transparent.

1.2.3 Logistic Regression:

Logistic regression is a widely used statistical method for binary classification problems, where the goal is to predict the probability that a given input belongs to one of two possible classes. Despite its name, logistic regression is a classification algorithm rather than a regression algorithm. Logistic regression finds the best-fitting linear decision boundary that separates the two classes.

Logistic regression is a fundamental algorithm for binary classification tasks, valued for its simplicity, interpretability, and efficiency. It can be considered as an extension of ordinary regression and can model only a dichotomous variable which usually represent the occurrence or non-occurrence of an event. Logistic regression helps in finding the probability that a new instance belongs to a certain class.

1.2.4 Random Forest:

A random forest is an ensemble classifier and consisting of many decision trees like the way a forest is a collection of many trees. Decision trees that are grown very deep often cause over-fitting of the training data, resulting a high variation in classification outcome for a small change in the input data. They are very sensitive to their training data, which makes the error-prone to the test dataset. The different decision trees of a random forest are trained using the different parts of the training dataset. To classify a new sample, the input vector of that sample is required to pass down with each decision tree of the forest. Each decision tree then considers a different part of the input vector and gives a classification outcome.

1.2.5 Naïve Bayes:

Naïve Bayes (NB) is a classification technique based on the Bayes' theorem. These algorithms assume the presence of a particular feature in a class is independent of the presence of any other features, hence the term 'naïve'. This theorem can describe the probability of an event based on the prior knowledge of conditions related to that event. This classifier assumes that a particular feature in a class is not directly related to any other feature although features for that class could have interdependence among themselves.

1.2.6 K-Nearest Neighbour:

The K-nearest neighbour (KNN) algorithm is one of the simplest and earliest classification algorithms. KNN is a simple, non-parametric, and lazy learning algorithm used for both classification and regression tasks. The algorithm is based on the principle that similar data points are likely to have similar outcomes. It classifies a data point based on how its neighbours are classified. It can be thought a simpler version of an NB classifier. Unlike the NB technique, the KNN algorithm does not require to consider probability values. The 'K' in the KNN algorithm is the number of nearest neighbours considered to take 'vote' from. The selection of different values for 'K' can generate different classification results for the same sample object.

2. LITERATURE SURVEY

1. Intelligent irrigation system—An IOT based approach

[MN Rajkumar, S Abinaya, V Venkatesa Kumar-2017 - ieeexplore.ieee.org](#)

The Internet of Things (IOT) has been denoted as a new wave of information and communication technology (ICT) advancements. The IOT is a multidisciplinary

concept that encompasses a wide range of several technologies, application domains, device capabilities, and operational strategies, etc. The ongoing IOT research activities are directed towards the definition and design of standards and open architectures which still have the issues requiring a global consensus before the final deployment. This paper gives overview about IOT technologies and applications related to agriculture with comparison of other survey papers and proposed a novel irrigation management system. Our main objective of this work is to for Farming where various new technologies to yield higher growth of the crops and their water supply. Automated control features with latest electronic technology using microcontroller which turns the pumping motor ON and OFF on detecting the dampness content of the earth and GSM phone line is proposed after measuring the temperature, humidity, and soil moisture.

2. Fast and Intelligent Irrigation System Based on WS

[G. Oussama](#), [A. Rami](#), [F. Tarek](#), [Ahmed S. Alanazi](#), M. Abid

First published: 14 July 2022

One of the most challenging aspects of wireless sensor networks is their energy efficiency. Based on the water pipeline, this paper proposes an efficient way of utilizing the energy of wireless sensor networks for agriculture production. We are particularly interested in various monitoring techniques suitable for preserving the number of water pipelines as part of our work. To provide WSN with an information platform that can support a better role in agricultural production, the proposed model aims to improve on the shortcomings of existing WSN in the context of energy efficiency. Agriculture is a key factor impacting the global economy; therefore, WSN is important.

In the end, based on our solution, we conclude that 19% of water could be saved using this irrigation practice. Also, wireless sensor networks can improve accuracy and efficiency to achieve agricultural goals as well as reduce costs associated with wireless protocol systems. Agriculture can improve its management and production efficiency because of using an agricultural automation system.

3. Towards precision agriculture: IoT-enabled intelligent irrigation systems using deep learning neural network

PK Kashyap, S Kumar, [A Jaiswal](#), [M Prasad](#), 2021 - ieeexplore.ieee.org

Recently, precision agriculture has gained substantial attention due to the ever-growing world population demands for food and water. Consequently, farmers will need water and arable land to meet this demand. Due to the limited availability of both resources, farmers need a solution that changes the way they operate. Precision irrigation is the solution to deliver bigger, better, and more profitable yields with fewer resources. Several machine learning-based irrigation models have been proposed to use water more efficiently. Due to the limited learning ability of these models, they are not well suited to unpredictable climates. In this context, this paper proposes a deep learning neural network-based Internet of Things (IoT)-enabled intelligent irrigation system for precision agriculture (DLISA). This is a feedback integrated system that keeps its functionality better in the weather of any region for any period of time. DLISA

utilizes a long short-term memory network (LSTM) to predict the volumetric soil moisture content for one day ahead, irrigation period, and spatial distribution of water required to feed the arable land. It is evident from the simulation results that DLISA uses water more wisely than state-of-the-art models in the experimental farming area.

3. IoT-based intelligent irrigation management and monitoring system using arduino

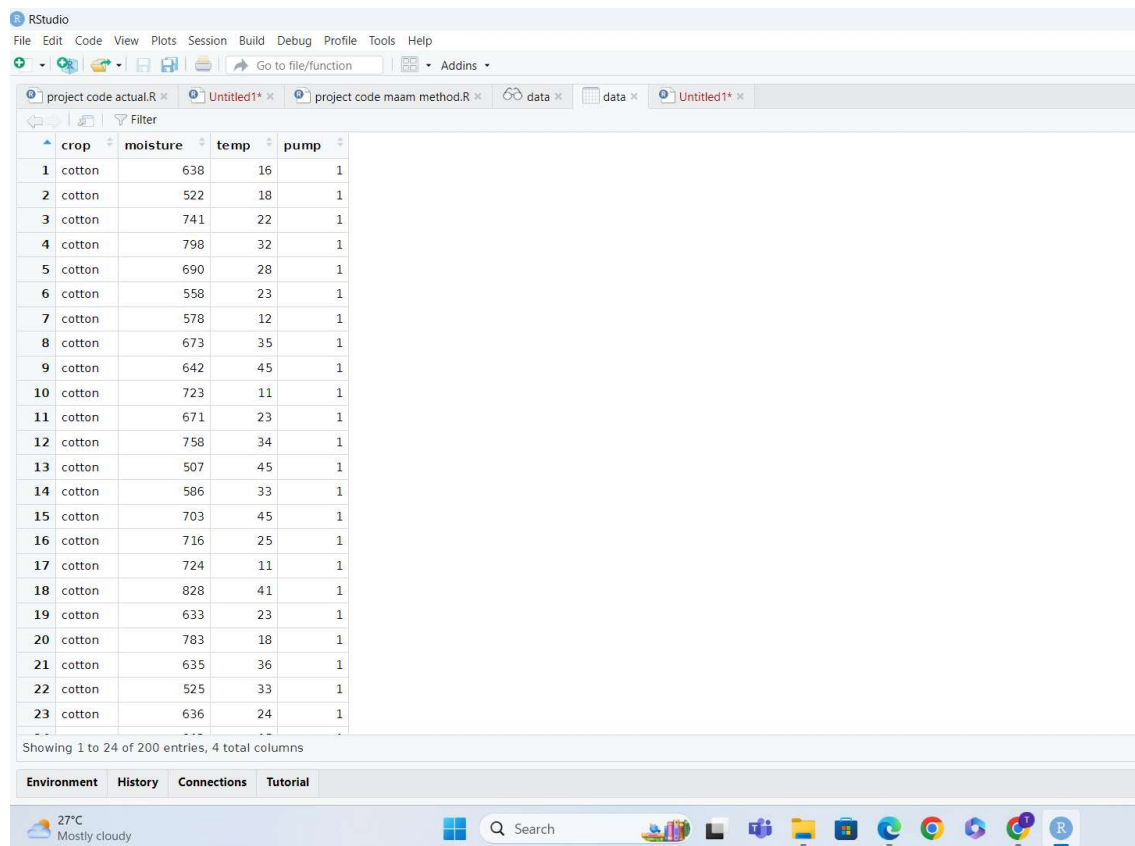
Fidaus Kamaruddin, Nik Noordini Nik Abd Malik, Noor Asniza Murad, Nurul Mu'azzah Abdul Latiff, Sharifah Kamilah Syed Yusof, Shipun Anuar Hamzah

Plants, flowers and crops are living things around us that makes our earth more productive and beautiful. In order to growth healthy, they need water, light and nutrition from the soil in order to effect cleaning air naturally and produce oxygen to the world. Therefore, a technology that manage to brilliantly control plants watering rate according to its soil moisture and user requirement is proposed in this paper. The developed system included an Internet of Things (IoT) in Wireless Sensor Network (WSN) environment where it manages and monitors the irrigation system either manually or automatically, depending on the user requirement. This proposed system applied Arduino technology and NRF24L01 as the microprocessor and transceiver for the communication channel, respectively. Smart agriculture and smart lifestyle can be developed by implementing this technology for the future work. It will save the budget for hiring employees and prevent from water wastage in daily necessities.

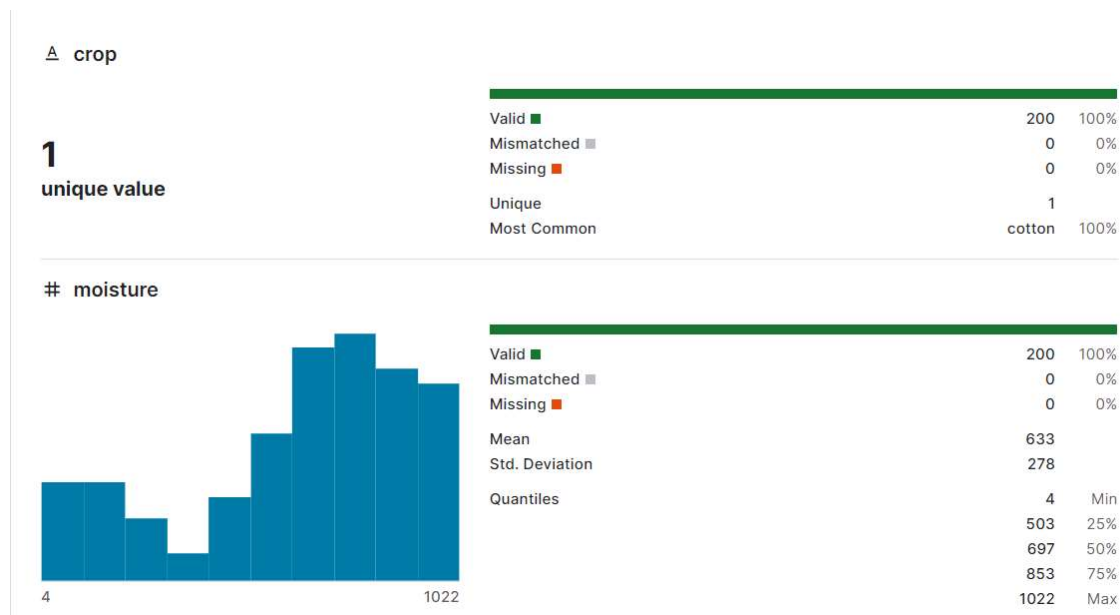
3. EXPLORATORY ANALYSIS ON THE DATASET:

The dataset consists of 200 entries with 4 features related to irrigated land. Here is a sample of data.

- Crop: denotes the type of cultivation in the land
- Moisture: denotes dampness in the soil
- Temperature: denotes temperature in the area
- Pump: 1-denotes that the land requires water supply
0-denotes that the land doesn't require water supply



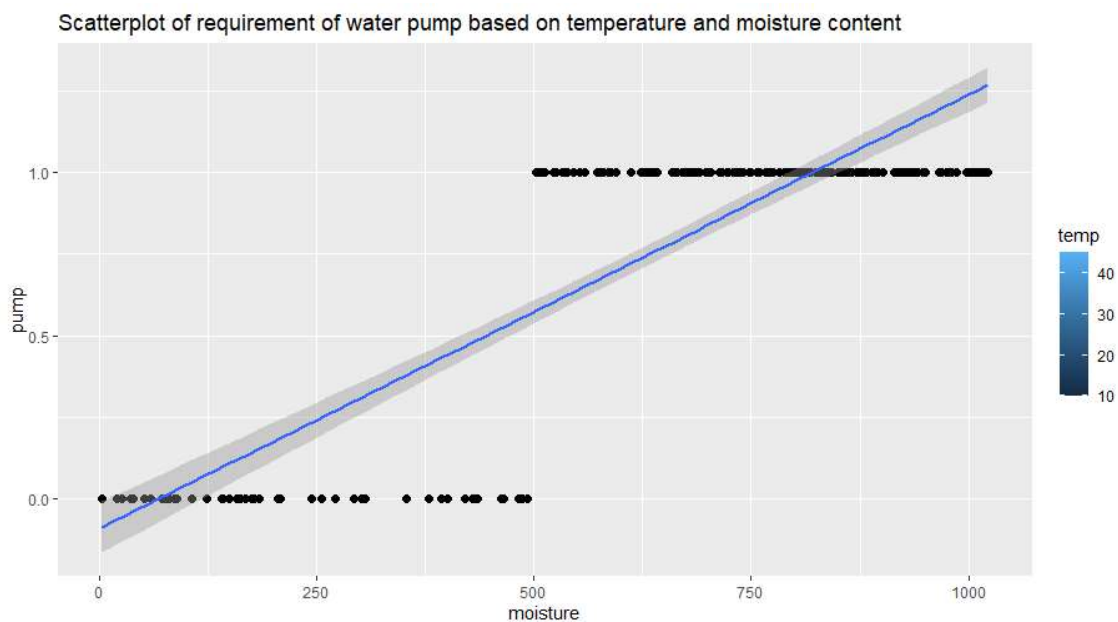
3.1 SUMMARY STATISTICS:





3.2 VISUALIZATION OF THE DATASET

3.2.1. SCATTERPLOT: This plot represents the requirement of water supply to the land cultivating the cotton crop by '1' and no need of water supply by '0'



4.METHODOLOGY

4.1. MISSING VALUES IN THE DATASET:

Missing values in data can pose significant challenges in data analysis and modelling. In R, handling missing values efficiently is crucial to ensure the quality and reliability of the results. Below is a note on missing values, including some common techniques to handle them along with corresponding R code examples.

Missing values, often represented as NA in R, can occur for various reasons such as data entry errors, measurement issues, or unrecorded observations. Handling these missing values appropriately is essential to maintain the integrity of the analysis. Below are some common methods to deal with missing values in R. To identify missing values in a dataset, we can use functions like `is.na()` and `sum()` to get a count of missing values.

Handling missing values is a critical step in data preprocessing. Depending on the nature and extent of missing data, one can choose to either remove or impute the missing values. Simple imputation methods like mean or median substitution are easy to implement but may not be suitable for all datasets. Advanced techniques such as those provided by the `mice` and `missForest` packages offer robust alternative for handling missing data, especially in complex datasets.

Upon examining the dataset for missing values using the provided R code, it is observed that there are no missing values in any of the columns. Every column has complete data without any null entries. This indicates that the dataset is fully populated and ready for further analysis without the need for handling missing values.

4.2. SAMPLING TECHNIQUES USED IN THIS ANALYSIS:

Simple random sampling is a fundamental statistical method used to select a subset of individual from a large population. The basic idea is to ensure that every individual in the population has an equal chance of being selected. This method is crucial for reducing bias and ensuring that the sample accurately represents the population, allowing for valid and reliable statistical inferences. In our data, we divided the total population into 75% as the training dataset and 25% as the test dataset by using simple random sampling technique.

4.2.1 Steps in Simple Random Sampling:

1. **Define the Population:** The first step is to clearly define the population from which you want to draw a sample. This could be dataset or a list of elements
2. **Determine the Sample Size:** Decide the number of samples (n) you want to draw from the population. This can be based on the study's requirements, resources, or statistical guidelines.
3. **Select the Sample:** Use R's built-in functions to randomly select samples. The `'sample ()'` function is used to randomly select rows from the dataset. The `'set.seed (123)'` function ensures that the results are reproducible by setting a seed for the random number generator.

4.3 VALIDATION METHODS

To discuss validation methods for your dataset, we'll first need to load the data and understand its structure. Once we have a grasp on the data, we can explore various validation techniques to ensure the robustness of our models.

Validation methods are essential to evaluate the performance of a model and ensure that it generalizes well to unseen data. Here, we discuss a few common validation techniques:

1. Train -Test Split
2. K -Fold Cross -Validation
3. Leave -One -Out Cross -Validation (LOOCV)
4. Stratified K -Fold Cross -Validation

In our case, we used most commonly used validation technique Test -Train Split. The Train- Test Split validation technique is a method used in machine learning to evaluate the performance of a model. The dataset is divided into two parts: a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate its performance. This approach helps in understanding how well the model generalizes to new, unseen data.

Steps in Train-Test Split Validation:

- i. **Split the Data:** Divide the dataset into a training set and a testing set. A common split ratio is 80/20 or 70/30, where 80% (70%) of the data is used for training, and 20% (or 30%) is used for testing. In our analysis, we split the dataset into 75/25 ratio, 75% for training and 25% for test set.
- ii. **Train the model:** Use the training set to train the machine learning model.
- iii. **Test the model:** Evaluate the model's performance on the testing set.
- iv. **Assess Performance:** Calculate performance metrics such as accuracy, precision, recall, F1-score, etc., to understand the model's effectiveness. The confusion-Matrix function is used to evaluate the performance of the model by comparing the predicted labels with the true labels in the testing set.

The **caret** package is used for data partitioning and model evaluation, while the **e1071** package provides the implementation of the **SVM** (Support Vector Machine) model. This approach is essential for building robust and generalizable machine learning models, as it helps in understanding the model's performance on unseen data.

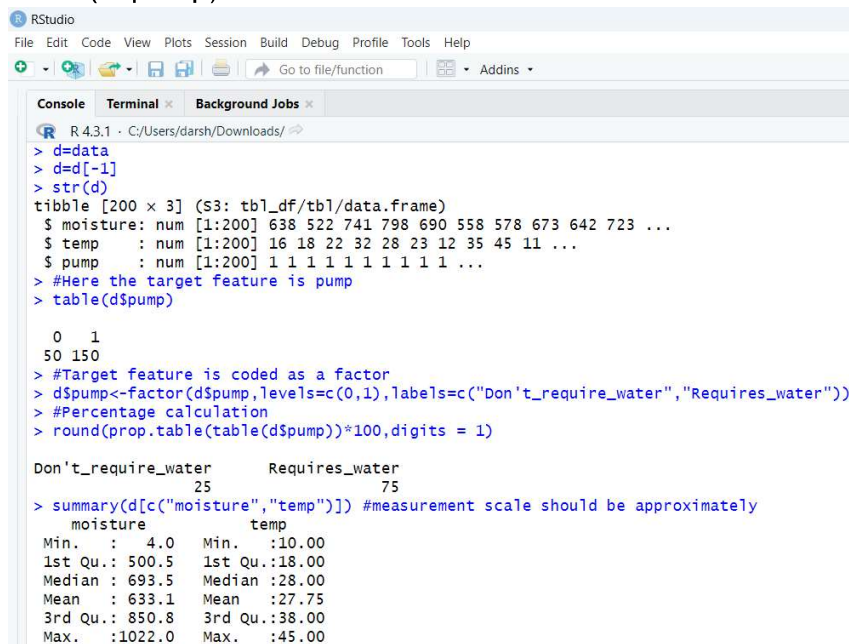
5. CODE

#Data Collection

- Library(readr)
- data <- read_csv("archive (6)/data.csv")
- View(data)
- d=data;d
- d=d[-1]
- str(d)

Here the target feature is “pump”

- table(d\$pump)



```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Console Terminal Background Jobs
R 4.3.1 - C:/Users/darsh/Downloads/
> d=data
> d=d[-1]
> str(d)
tibble [200 × 3] (S3: tbl_df/tbl/data.frame)
 $ moisture: num [1:200] 638 522 741 798 690 558 578 673 642 723 ...
 $ temp    : num [1:200] 16 18 22 32 28 23 12 35 45 11 ...
 $ pump    : num [1:200] 1 1 1 1 1 1 1 1 1 1 ...
> #Here the target feature is pump
> table(d$pump)

 0    1 
50   150 
> #Target feature is coded as a factor
> d$pump<-factor(d$pump,levels=c(0,1),labels=c("Don't_require_water","Requires_water"))
> #Percentage calculation
> round(prop.table(table(d$pump))*100,digits = 1)

Don't_require_water    Requires_water 
                25              75 
> summary(d[c("moisture","temp")]) #measurement scale should be approximately
moisture      temp
Min.   :   4.0   Min.   :10.00
1st Qu.: 500.5   1st Qu.:18.00
Median : 693.5   Median :28.00
Mean   : 633.1   Mean   :27.75
3rd Qu.: 850.8   3rd Qu.:38.00
Max.   :1022.0   Max.   :45.00

```

The Target Feature is Coded as a Factor

- d\$pump <- factor(d\$pump,levels=c (0,1),
labels=c("Don't_require_water","Requires_water"))

Percentage Calculation

- round(prop.table(table(d\$pump))*100, digits = 1)
- summary (d [c ("moisture", temp)]) #measurement scale should be approximately same

#Normalizing the Variables

- normalize<-function(x) {
 return((x-min(x))/(max(x)-min(x)))
}
- d_n<-as.data.frame(lapply(d[1:2], normalize))
- summary(d_n\$moisture)

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Console Terminal Background Jobs
R 4.3.1 · C:/Users/darsh/Downloads/
> normalize<-function(x){
+   return((x-min(x))/(max(x)-min(x)))
+ }
> d_n<-as.data.frame(lapply(d[1:2],normalize))
> summary(d_n$moisture)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.0000  0.4877   0.6773   0.6180  0.8318   1.0000

```

#Data Preparation

- `sample_size=floor (0.75*nrow(d));sample_size`
- `set.seed(123)`
- `train_ind=sample(seq_len(nrow(d)), size=sample_size);train_ind`

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Console Terminal Background Jobs
R 4.3.1 · C:/Users/darsh/Downloads/
> train_ind=sample(seq_len(nrow(d)),size=sample_size);train_ind
[1] 159 179 14 195 170 50 118 43 198 194 153
[12] 90 91 188 185 92 137 99 72 26 7 196
[23] 184 164 78 81 193 103 117 76 143 32 109
[34] 180 178 74 23 155 53 135 162 163 34 69
[45] 182 171 63 141 97 187 38 21 41 189 60
[56] 16 116 94 6 86 190 39 191 197 158 4
[67] 13 157 127 52 22 89 169 110 25 87 35
[78] 40 112 30 12 31 132 121 64 183 192 93
[89] 96 71 67 177 79 85 37 8 51 165 166
[100] 98 173 140 200 84 46 17 62 122 54 124
[111] 108 24 125 167 147 168 181 128 88 27 42
[122] 5 70 145 129 83 149 55 36 139 186 9
[133] 48 56 111 1 123 172 154 131 126 77 101
[144] 176 11 68 20 144 3 29

```

- `d_train=d_n[train_ind,];d_train #Trainind Dataset`

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
New File
Console Terminal Background Jobs
R 4.3.1 · C:/Users/darsh/Downloads/
> d_train=d_n[train_ind,];d_train #training dataset
  moisture      temp
159 0.84577603 0.20000000
179 0.98330059 0.94285714
14  0.57170923 0.65714286
195 0.89390963 0.71428571
170 0.97740668 0.57142857
50  0.51964637 0.91428571
118 0.15127701 0.20000000
43  0.68369352 0.80000000
198 0.87426326 0.91428571
194 0.99312377 0.71428571
153 0.95972495 0.74285714
90  0.53143418 0.54285714
91  0.83104126 0.17142857
188 0.95284872 0.51428571
185 0.86836935 0.97142857
92  0.70530452 0.45714286
137 0.08153242 0.05714286
99  0.57367387 0.22857143
72  0.62180747 0.54285714
26  0.81434185 0.08571429
7   0.56385069 0.05714286
196 0.92043222 0.08571429
184 0.86345776 0.85714286
```

➤ `d_test=d_n[-train_ind,];d_test`

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Console Terminal Background Jobs
R 4.3.1 · C:/Users/darsh/Downloads/
> d_test=d_n[-train_ind,];d_test
  moisture      temp
2  0.50884086 0.22857143
10 0.70628684 0.02857143
15 0.68664047 1.00000000
18 0.80943026 0.88571429
19 0.61787819 0.37142857
28 0.69744597 0.11428571
33 0.74165029 0.00000000
44 0.73182711 0.80000000
45 0.52357564 0.97142857
47 0.80550098 0.77142857
49 0.72298625 0.60000000
57 0.52161100 0.20000000
58 0.79960707 0.14285714
59 0.49017682 0.97142857
61 0.50000000 0.48571429
65 0.62573674 0.71428571
66 0.66011788 0.57142857
73 0.75147348 0.08571429
75 0.55992141 0.62857143
80 0.74263261 0.22857143
82 0.74852652 0.82857143
95 0.81139489 0.97142857
100 0.71611002 0.45714286
102 0.07072692 0.71428571
104 0.47445972 0.17142857
105 0.48035363 0.22857143
106 0.40962672 0.82857143
107 0.36836935 0.37142857
113 0.29666012 0.42857143
```

- `d_train_labels=d[train_ind,3];d_train_labels`
- `library(dplyr)`
- `d_train_label=d_train_labels%>%pull ()`

➤ `d_test_labels=t(d[-train_ind,3]);d_test_labels`

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins Project

Console Terminal Background Jobs
R 4.3.1 - C:/Users/darsh/Downloads/
> d_train_labels=d[train_ind,3];d_train_labels
# A tibble: 150 x 1
  pump
  <fct>
1 Requires_water
2 Requires_water
3 Requires_water
4 Requires_water
5 Requires_water
6 Requires_water
7 Don't_require_water
8 Requires_water
9 Requires_water
10 Requires_water
# i 140 more rows
# i Use `print(n = ...)` to see more rows
> library(dplyr)
> d_train_label=d_train_labels%>%pull()
> d_test_labels=t(d[-train_ind,3]);d_test_labels
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
pump "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water"
  [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18]
pump "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water"
  [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
pump "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Don't_require_water" "Don't_require_water" "Don't_require_water"
  [,27] [,28] [,29] [,30] [,31] [,32] [,33]
pump "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water"
  [,34] [,35] [,36] [,37] [,38] [,39] [,40]
pump "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water" "Don't_require_water"
  [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48]
pump "Don't_require_water" "Don't_require_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water" "Requires_water"
  [,49] [,50]
pump "Requires_water" "Requires_water"

```

#Initialize a vector to store accuracy for each k

- `k_values<-1:30`
- `k_nn=0`
- `accuracy<-numeric(length(k_values))`
- `library(class)`

#Loop through each k value and perform k-NN

- ```
for(k in k_values){
 pred<-knn(train=d_train,test=d_test,cl=d_train_label,k)
 incorrect=sum(d_test_labels!=pred)
 mis_rate=sum(incorrect)/length(d_test_labels)
 cat(k,mis_rate,"\n")
 k_nn[k]=mis_rate
 accuracy[k]<-sum(pred==d_test_labels)/length(pred)
}
```

**#Choosing the k which has minimum misclassification rate**

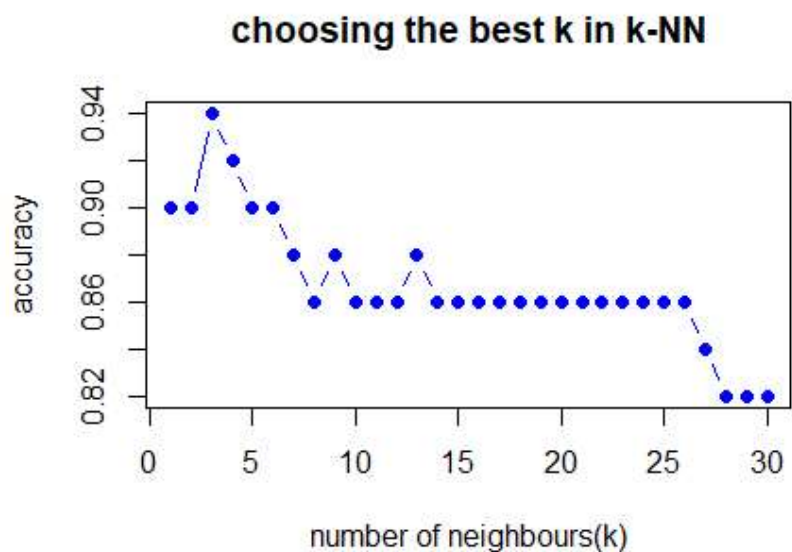
- `k1=which.min(k_nn);k1`



```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Project: (None)
Console Terminal Background Jobs
R 4.3.1 - C:/Users/darsh/Downloads/
1 v.1
2 0.08
3 0.06
4 0.08
5 0.1
6 0.14
7 0.12
8 0.14
9 0.12
10 0.12
11 0.14
12 0.14
13 0.12
14 0.14
15 0.14
16 0.14
17 0.14
18 0.14
19 0.14
20 0.14
21 0.14
22 0.14
23 0.14
24 0.14
25 0.14
26 0.16
27 0.16
28 0.18
29 0.18
30 0.18
> k1=which.min(k_nn);k1
[1] 3
```

### #Plot of Accuracy v/s K

- `plot(k_values,accuracy,type="b",col="blue",pch=19, xlab="number of neighbours(k)",ylab="accuracy",main="choosing best k in k-NN")`



### #Training a Model on the Data

- `library(class)`
- `d_test_pred=knn(d_train,d_test,d_train_label,k=k1);d_test_pred`



```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Console Terminal Background Jobs
R 4.3.1 - C:/Users/darsh/Downloads/
+ xlab="number of neighbours(k)",ylab="accuracy",
+ main="choosing the best k in k-NN")
> library(cclass)
> d_test_pred=knn(d_train,d_test,d_train_label,k=k1);d_test_pred
[1] Requires_water Requires_water
[3] Requires_water Requires_water
[5] Requires_water Requires_water
[7] Requires_water Requires_water
[9] Requires_water Requires_water
[11] Requires_water Requires_water
[13] Requires_water Requires_water
[15] Requires_water Requires_water
[17] Requires_water Requires_water
[19] Requires_water Requires_water
[21] Requires_water Requires_water
[23] Requires_water Don't_require_water
[25] Don't_require_water Don't_require_water
[27] Requires_water Don't_require_water
[29] Don't_require_water Don't_require_water
[31] Don't_require_water Requires_water
[33] Don't_require_water Requires_water
[35] Don't_require_water Don't_require_water
[37] Don't_require_water Don't_require_water
[39] Don't_require_water Don't_require_water
[41] Don't_require_water Don't_require_water
[43] Requires_water Requires_water
[45] Requires_water Requires_water
[47] Requires_water Requires_water
[49] Requires_water Requires_water
Levels: Don't_require_water Requires_water

```

## #Evaluating Model Performance

- library(gmodels)
- ct=CrossTable(d\_test\_labels,d\_test\_pred,prop.chisq=FALSE);ct

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Console Terminal Background Jobs
R 4.3.1 - C:/Users/darsh/Downloads/
> ct=CrossTable(d_test_labels,d_test_pred,prop.chisq=FALSE);ct

Cell Contents
-----|
| N / Row Total |
| N / Col Total |
N / Table Total

Total Observations in Table: 50

d_test_labels	d_test_pred	Requires_water	Row Total
Don't_require_water	16	3	19
0.842	0.158	0.380	
1.000	0.088		
0.320	0.060		
-----	-----	-----	-----
Requires_water	0	31	31
0.000	1.000	0.620	
0.000	0.912		
0.000	0.620		
-----	-----	-----	-----
Column Total	16	34	50
0.320	0.680		
-----	-----	-----	-----

```

- counts=ct\$counts

$$\begin{matrix}
 & \text{X} & & \text{Y} \\
 & \text{Don't\_require\_water} & & \text{Requires\_water} \\
 \text{Don't\_require\_water} & 16 & & 3
 \end{matrix}$$

- TP=counts [1,1]; TP #True Positives  
16
- TN=counts [2,2]; TN #True Negatives  
31
- FP=counts [1,2]; FP #False Positives  
3
- FN=counts [2,1]; FN #False Negatives  
0

### Interpretation of Cross-Table:

Cross-Table in R is a powerful tool for exploring and analyzing the relationships between categorical variables. By offering detailed statistics and proportions, it helps in understanding the underlying patterns and dependencies within the data. This function is particularly useful for preliminary data analysis and hypothesis testing in categorical data analysis.

- The top left cell indicates True Positive (TP), means, land don't require water supply and the k-NN correctly identified it as such
- The bottom right cell indicates True Negative (TN), means, land requires water supply and k-NN correctly identified it as such
- The bottom left cell indicates False Negative (FN), means, land requires water supply and k-NN identified it as land don't require water
- The top right cell indicates False Positive (FP), means, land don't require water and k-NN identified it as land requires water

## 6. EVALUATION METRICS

**Accuracy:** Accuracy is a common metric used to evaluate the performance of classification models in machine learning. It measures the proportion of correctly predicted instances out of the total instances in the dataset. Accuracy is defined as the ratio of the number of correct predictions to the total number of predictions.

**Accuracy =  $(TP+TN) / (TP+TN+FP+FN)$**

- `accuracy<-(TP+TN)/(TP+TN+FP+FN); accuracy`  
[1] 0.94

**Precision:** Precision is a performance metric for classification models, especially useful in scenarios with imbalanced classes. It measures the accuracy of positive predictions, specifically the ratio of true positives to the sum of true positives and false positives.

**Precision =  $TP / (TP+FP)$**

➤  $\text{precision} = \text{TP} / (\text{TP} + \text{FP})$ ; precision  
[1] 0.8421053

**Recall:** Recall, also known as sensitivity or true positive rate, measures the proportion of actual positives that are correctly identified by the model. It is particularly useful in situations where it is important to capture as many positives as possible.

**Recall =  $\text{TP} / (\text{TP} + \text{FN})$**

➤  $\text{recall} = \text{TP} / (\text{TP} + \text{FN})$ ; recall  
[1] 1

**F1- Score:** The F1-Score is a performance metric that combines precision and recall into a single measure. It is the harmonic mean of precision and recall, providing a balance between the two metrics.

**F1-Score =  $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$**

➤  $\text{f1\_score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$ ; f1\_score  
[1] 0.9142857

**Interpretation:**

- Accuracy is 94%, it means that 94% of the predictions made by the model are correct. In other words, out of all instances it predicted, 94% were classified correctly as either positive or negative. Since the accuracy is relatively high, it suggests that the model's performance is very reliable.
- Precision is 84.21%, this indicates a low number of false positives, which means the model is correctly predicting

**REFERENCE:**

1. "Intelligent Irrigation System Using kNN and IoT" by S.S.Rao et al. (2020)
2. "KNN-Based Smart Irrigation System for Water Conservation" by A.K.Singh et al. (2019)
3. "Irrigation Scheduling Using KNN and Weather Forecast Data" by M.A.Ahmed et al. (2018)
4. "Smart Irrigation System Using KNN, IoT, and Sensors" by R.S.Yadav et al. (2019)