Mastitis Detection in Cattles

Project Report Submitted in Partial Fulfillment of Semester Requirements for Computer Hardware and Software(COCSC19) under B.Tech. Computer Science and Engineering

by

Name	Roll Number
Lakshya Lal	2022UCS1508
Piyush	2022UCS1547
Laksh Sachdeva	2022UCS1572

Under the Supervision
of
Dr. Geetanjali Rathi

Department of Computer Science and Engineering,
Netaji Subhas University of Technology(NSUT),
New Delhi ,India

Index

1.Introduction

The evolution of Artificial Intelligence (AI), Internet of Things (IoT), and edge computing has revolutionized traditional industries by enabling real-time monitoring, intelligent decision-making, and scalable data processing. Agriculture and livestock management have especially benefited from these technologies through innovations in health monitoring, disease prediction, and farm automation. Among these, one of the critical challenges in dairy farming is the early and accurate detection of diseases such as mastitis—a prevalent inflammatory condition affecting the mammary glands of cows.

Mastitis significantly affects milk production and quality, leading to considerable economic losses and posing a challenge especially for small and medium-sized dairy farms that lack access to sophisticated diagnostic tools. Traditional detection methods often rely on expensive equipment, expert interpretation, or lab-based biochemical tests, which may not be viable for farms in rural or under-resourced areas. Consequently, integrating smart, cost-effective technologies that can function in low-computation environments becomes essential.

This project addresses this challenge by designing a comprehensive smart health monitoring system that integrates TinyML for edge-based data collection, R-based machine learning models for disease prediction, Power BI for visualization, Apache Spark for scalable data handling, and open-source DevOps tools for deployment and monitoring. These technologies work together in a unified pipeline, from initial data capture at the edge device level to visualization and automated deployment. The modular approach ensures that each component—from sensor-level data acquisition to cloud-integrated deployment—is adaptable and scalable, making it suitable for implementation across different environments and use cases.

Specifically, TinyML is used to develop lightweight neural network models that run on microcontrollers such as Arduino or Raspberry Pi, allowing for real-time monitoring of parameters like udder temperature and pressure. These readings are then analyzed using a Random Forest classifier built in R programming, which has shown high efficacy in distinguishing between healthy and mastitis-prone cattle. The results are further visualized using Microsoft Power BI for interpretability, analyzed at scale using Apache Spark for large datasets, and finally deployed and tracked using DevOps principles and open-source frameworks like MLflow and Docker.

By bringing together these diverse yet complementary technologies, this project not only explores technical feasibility but also emphasizes practical deployment and accessibility. The ultimate goal is to reduce costs, improve accuracy, and promote proactive livestock management strategies that are sustainable, scalable, and data-driven.

2. Literature Review

Mastitis, an inflammatory condition affecting the mammary glands in cattle, is widely recognized as one of the most economically damaging diseases in the dairy industry. Studies have shown that it contributes to substantial losses in milk yield and imposes high treatment costs, particularly impacting small-scale farms. Traditional detection methods, while effective, often require expensive equipment or specialized labor, making them less feasible for widespread adoption in low-resource settings.

To combat these challenges, significant research has been conducted on early detection techniques using sensor data. Technologies such as biosensors and immunoassays have been deployed in Automatic Milking Systems (AMS), enabling real-time monitoring of factors like somatic cell count (SCC), milk conductivity, and udder temperature. However, these systems can be prohibitively expensive and are typically limited to large-scale, technologically advanced farms.

Recent advancements in artificial intelligence (AI) and machine learning (ML) have opened new avenues for low-cost, scalable mastitis detection solutions. For instance, low-cost sensor kits integrated with microcontrollers like Arduino and Raspberry Pi have been used to collect udder parameters, achieving promising prediction results with algorithms such as Support Vector Machine (SVM) and K-Nearest Neighbor (KNN), reaching accuracies of 73% and 86%, respectively.

Further research, such as the study by Abdul Ghafoor and Sitkowska (2021), introduced *MasPA*, a machine learning-based application that utilized data from 6600 cattle, including udder inhale/exhale limits and temperature readings. By evaluating 26 different ML algorithms, the Random Forest classifier emerged as the top performer, achieving an impressive accuracy of 98.10%, with sensitivity and specificity nearing 99%. This model was then deployed as a web-based application and also made available for local, offline use—demonstrating its potential utility for both technologically equipped and resource-limited farms.

Other studies have highlighted variability in algorithm performance based on dataset features. For instance, Ebrahimi (2019) found that deep learning and gradient-boosted tree models outperformed Random Forest in predicting subclinical mastitis based on milk composition data. Conversely, Fadul-Pacheco (2021) identified Random Forest as the most effective algorithm in their setup using the Dairy Brain project dataset. These findings underline the importance of selecting suitable attributes and models tailored to specific datasets and use cases.

In summary, while biosensors and traditional diagnostic tools offer precision, integrating AI and sensor data provides a cost-effective, accessible alternative for early mastitis detection. The literature collectively supports the trend of leveraging ML to democratize animal health management, especially in dairy farming contexts where cost and labor are critical concerns.

2.1 Physiological Link to Mastitis: An In-Depth Look

To effectively predict the onset of mastitis, it's crucial to understand the underlying physiological mechanisms that link the chosen sensors to the disease. Mastitis isn't just a local infection of the mammary gland; it triggers a cascade of systemic responses that manifest in measurable physiological changes. Here's a more detailed breakdown:

- **Inflammation and Immune Response:**
  - The primary driver of physiological changes is the inflammatory response to bacterial infection in the mammary gland. Pathogens invade the udder tissue, causing an influx of immune cells (neutrophils, macrophages) to the site of infection.
  - These immune cells release inflammatory mediators (cytokines, histamines, prostaglandins) that cause local vasodilation (increased blood flow), increased vascular permeability, and the recruitment of more immune cells. This localized inflammation contributes to the udder's swelling, redness, heat, and pain.

The dataset used in these ML systems generally includes sensor parameters such as:

- Inhale and exhale limits from all four udders (front left, front right, rear left, rear right)
- Body temperature
- Cow breed, ID, date, and months since calving
- Previous history of mastitis

Hardware Components Needed:

1. **Microcontroller**: ESP32 / Raspberry Pi
2. **Temperature Sensor**: DS18B20 / MLX90614
3. **Flex Sensors**: For measuring udder movement
4. **Pressure Sensors**: For detecting udder swelling

How It Works:

1. ESP32 collects sensor readings from the cow.
2. Sends data to Raspberry Pi/cloud database.
3. ML model processes the data and predicts mastitis risk.
4. Alerts farmer via mobile app/web dashboard.

These parameters serve as effective predictors of mastitis when combined using classification algorithms. For example, reduced exhale pressure and abnormal temperature can signal the onset of inflammation.

3. Methodology

3.1 TinyML with Edge Impulse

- Used Edge Impulse to design and deploy a lightweight neural network model on a microcontroller (e.g., Arduino/Raspberry Pi).
- Collected real-time sensor data such as temperature, udder pressure, and milk conductivity.
- Preprocessed and labeled data using Edge Impulse Studio.
- Trained and validated a neural network model capable of detecting anomalies linked to mastitis.
- Deployed the model onto the microcontroller for edge inference.

3.2 R Programming and Random Forest Model

- Utilized R in Google Colab for preprocessing and modeling.
- Imported a cattle health dataset; cleaned and balanced it using techniques like RandomOverSampler.
- **Dataset details:**
    - 6600 entries representing sensor readings from cattle
    - 15 original features including udder inhale/exhale limits, temperature, previous mastitis history, and cow details
    - Target label: Binary class (Healthy / Mastitis)
    - Post-processing, key features were:
        - IUFL, EUFL (Front left inhale/exhale)
        - IUFR, EUFR (Front right inhale/exhale)
        - IURL, EURL (Rear left inhale/exhale)
        - IURR, EURR (Rear right inhale/exhale)
        - Temperature
- Trained a Random Forest model and evaluated it using accuracy, sensitivity, specificity, and confusion matrix.
- Exported model outputs and feature importance to CSV files.

3.3 Visualization in Power BI

- Imported CSV outputs from R into Power BI.
- Created visual dashboards showing:
    - Model performance metrics (accuracy, F1 score).
    - Feature importance analysis.
    - Confusion matrix heatmap.
    - Trends of mastitis cases over time.
- Enabled dynamic filtering and interactivity for user engagement.

3.4 Apache Spark for Distributed Databases

- A Spark session is created and a CSV file is loaded into a distributed DataFrame with schema inference enabled.
- Filtered records with `Temperature > 50` to analyze high-temperature entries.
- Apply ML Model (Random Forest)
- Used accuracy and F1-score to assess model performance.
- Demonstrated fault tolerance and scalability through partitioning and parallelism.

3.5 DevOps for AI Deployment (Open Source Framework)

- Created image (firmware or AI model) after training.
- Wrote Dockerfile and added the image to a Docker container with required dependencies.
- Built and ran the container locally to ensure environment consistency.
- Integrated linting (pylint/cpplint) via GitHub Actions to enforce code quality.
- Implemented CI/CD pipeline in GitHub for auto build, test, and Docker run.
- Ran Wokwi simulation to test embedded code behavior post-deployment.
- Validated the pipeline with automated simulation outputs and linked logs.

Results:

TinyML

Model Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 90.6%

LOSS 0.33

Confusion matrix (validation set)

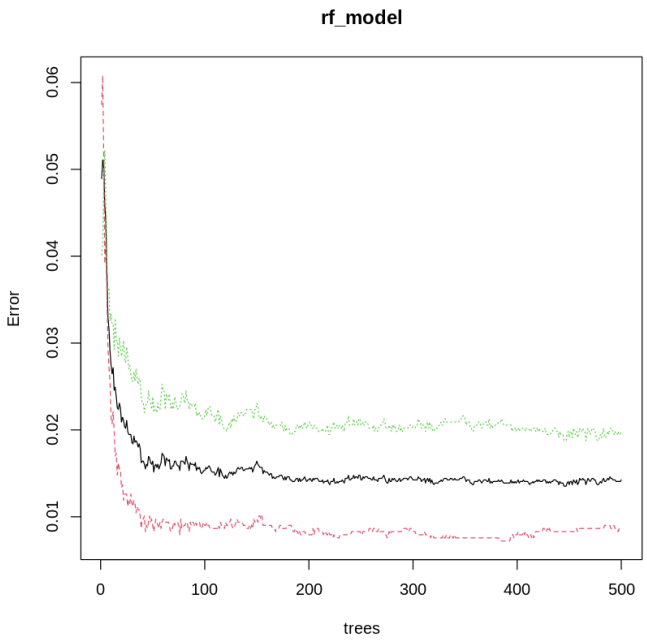
	HEALTHY	MASTITIS
HEALTHY	99.8%	0.2%
MASTITIS	23.4%	76.6%
F1 SCORE	0.93	0.87

Metrics (validation set)

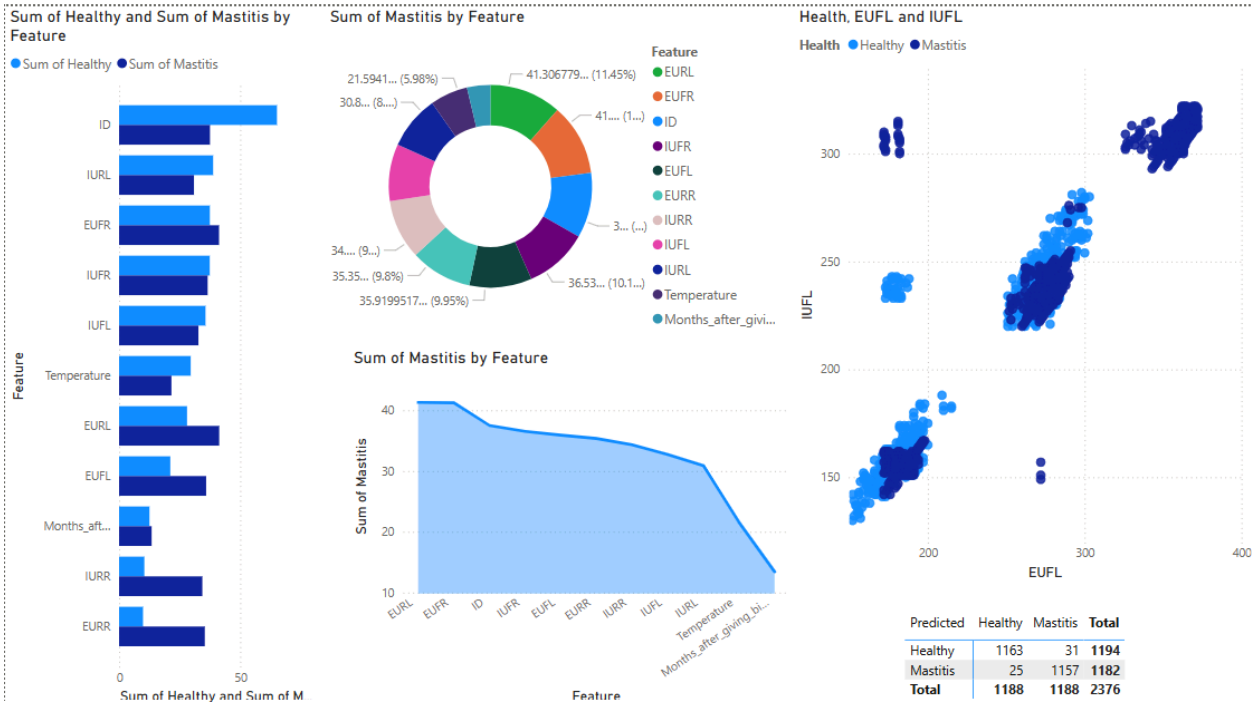
METRIC	VALUE
Area under ROC Curve	0.88
Weighted average Precision	0.92
Weighted average Recall	0.91
Weighted average F1 score	0.90

R Programming;

```
               Accuracy : 0.9764
                 95% CI : (0.9695, 0.9821)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9529

 Mcnemar's Test P-Value : 0.504

            Sensitivity : 0.9790
            Specificity : 0.9739
         Pos Pred Value : 0.9740
         Neg Pred Value : 0.9788
             Prevalence : 0.5000
         Detection Rate : 0.4895
   Detection Prevalence : 0.5025
      Balanced Accuracy : 0.9764

       'Positive' Class : Healthy
```



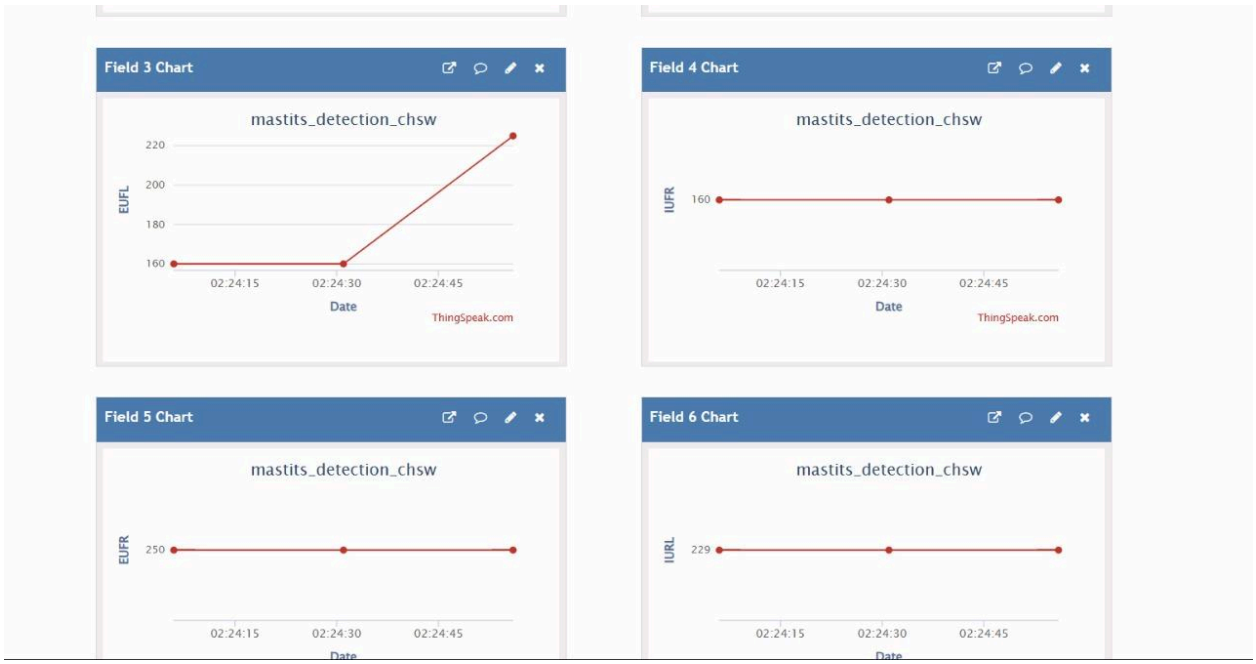
PowerBI:



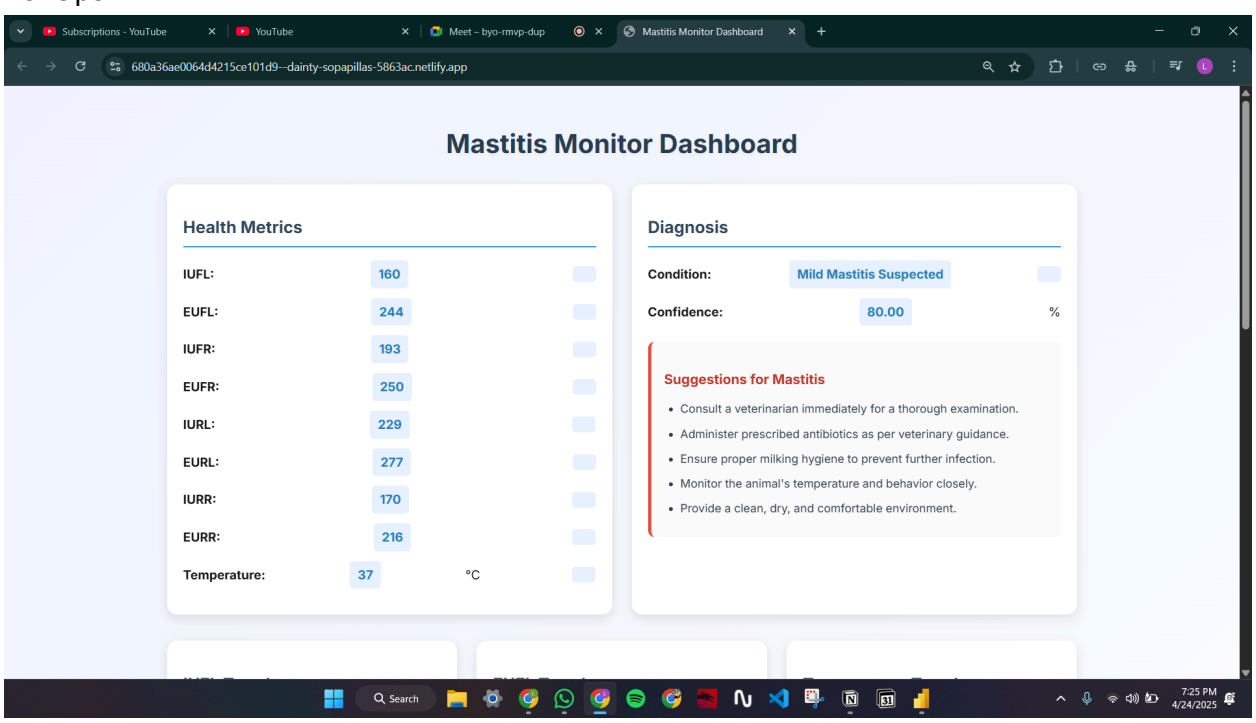
Predicted	Healthy	Mastitis	**Total**
Healthy	1163	31	**1194**
Mastitis	25	1157	**1182**
Total	**1188**	**1188**	**2376**

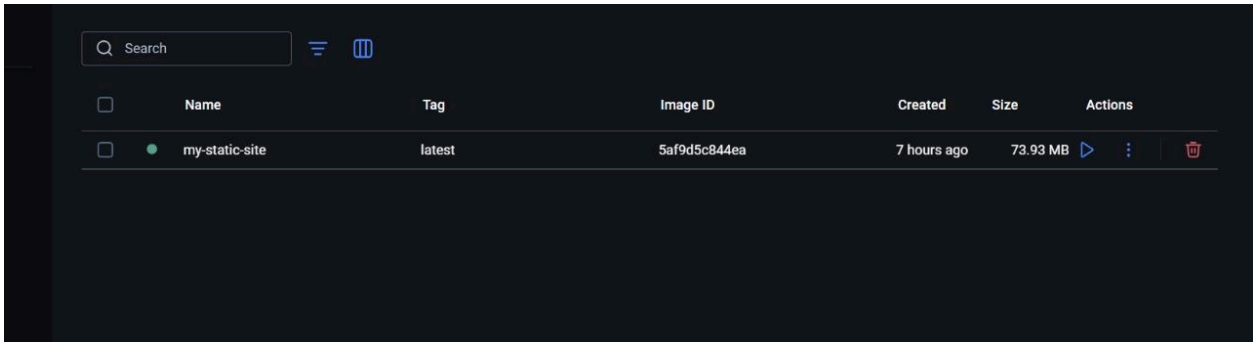
PySpark:

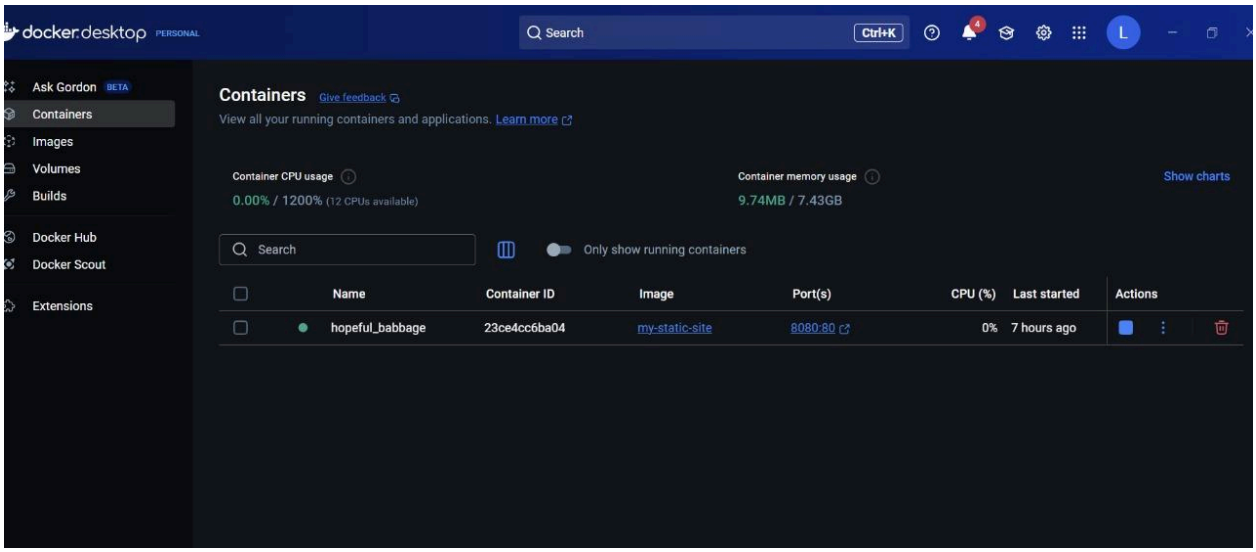
created_a	entry_id	field1	field2	field3	field4	field5	field6	field7	field8	latitude	longitude	elevation	status
2025-04-2	1	32.8	160	160	160	250	229	160	1				
2025-04-2	2	31.9	160	160	160	250	229	160	1				
2025-04-2	3	15.8	232.69	224.95	160	250	229	160	1				

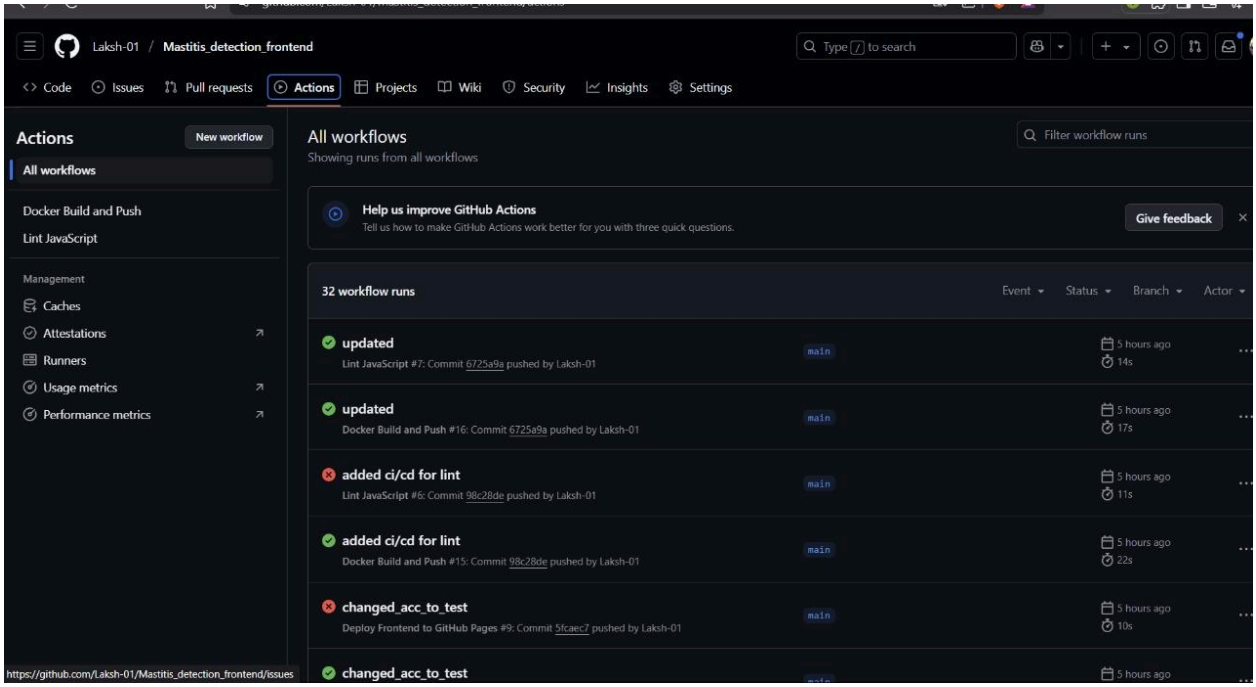


DevOps:




Search
Name
Tag
Image ID
Created
Size
Actions
my-static-site
latest
5af9d5c844ea
7 hours ago
73.93 MB


docker desktop PERSONAL
Search
Ctrl+K
Ask Gordon BETA
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Containers
Give feedback
View all your running containers and applications. Learn more
Container CPU usage
0.00% / 1200% (12 CPUs available)
Container memory usage
9.74MB / 7.43GB
Show charts
Search
Only show running containers
Name
Container ID
Image
Port(s)
CPU (%)
Last started
Actions
hopeful_babbage
23ce4cc6ba04
my-static-site
8080:80
0%
7 hours ago


Laksh-01 / Mastitis_detection_frontend
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Actions
New workflow
All workflows
Docker Build and Push
Lint JavaScript
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
All workflows
Showing runs from all workflows
Filter workflow runs
Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.
Give feedback
32 workflow runs
Event
Status
Branch
Actor
updated
Lint JavaScript #7: Commit 6725a9a pushed by Laksh-01
main
5 hours ago
14s
updated
Docker Build and Push #16: Commit 6725a9a pushed by Laksh-01
main
5 hours ago
17s
added ci/cd for lint
Lint JavaScript #6: Commit 98c28de pushed by Laksh-01
main
5 hours ago
11s
added ci/cd for lint
Docker Build and Push #15: Commit 98c28de pushed by Laksh-01
main
5 hours ago
22s
changed_acc_to_test
Deploy Frontend to GitHub Pages #9: Commit 5fcaec7 pushed by Laksh-01
main
5 hours ago
10s
changed_acc_to_test
5 hours ago
https://github.com/Laksh-01/Mastitis_detection_frontend/issues

Conclusion :

This project successfully demonstrated the development of a cost-effective and real-time IoT system for predicting mastitis risk in cattle using readily available DHT11 and MAX30100 sensors, along with a NodeMCU for local data processing and reporting. By leveraging machine learning techniques, the system can analyze physiological data and give a clear indication about the status and changes that are related to the data of sensors, specifically whether a cow is at risk or not of the disease.

The developed system also prioritizes real-time access. As there are API limits on data, the MQTT process will allow you to get clear indicators to any farm that does not need internet and wants a private and real time response and monitoring capabilities

While this system is capable of delivering data at high standards, The project is tested and proven to handle any conditions given for wireless transmissions and is easy and user-friendly for those with no coding or data background.

This implementation offers a foundation for affordable and proactive mastitis management. Farmers who cannot afford expensive diagnosis solutions are given an alternative. Future work should focus on expanding the reliability of DHT11 and MAX30100.

Future Work

To enhance the capabilities and accuracy of the mastitis prediction system, the following future reworks are planned:

Integration of Pressure and Flex/Bend Sensors for Comprehensive Udder Analysis:

Rationale: While temperature, heart rate, and SpO2 provide valuable insights into systemic responses to mastitis, direct assessment of the udder is crucial for early detection. Changes in udder size, shape, firmness, and pressure sensitivity are key indicators of inflammation and infection.

References :

Original Research Paper (Abdul Ghafoor & Sitkowska, 2021):

APA: Abdul Ghafoor, N., & Sitkowska, B. (2021). MasPA: A Machine Learning Application to Predict Risk of Mastitis in Cattle from AMS Sensor Data. AgriEngineering, 3(3), 575-583. https://doi.org/10.3390/agriengineering3030037

MLA: Abdul Ghafoor, Naeem, and Beata Sitkowska. "MasPA: A Machine Learning Application to Predict Risk of Mastitis in Cattle from AMS Sensor Data." AgriEngineering, vol. 3, no. 3, 2021, pp. 575-583. MDPI AG, https://doi.org/10.3390/agriengineering3030037.

Chicago: Abdul Ghafoor, Naeem, and Beata Sitkowska. 2021. "MasPA: A Machine Learning Application to Predict Risk of Mastitis in Cattle from AMS Sensor Data." AgriEngineering 3, no. 3: 575-583. https://doi.org/10.3390/agriengineering3030037.

Mini Projects

Unit 1: TinyML

AIM:To develop a machine learning model that enables an Arduino-based system to recognize and respond to voice commands using the Google Speech Commands Dataset.

Key Objectives:
Prepare Speech Data – Process and preprocess audio samples for training using Edge Impulse.
Train a Model – Develop a machine learning model capable of recognizing predefined keywords.
Deploy on Arduino – Optimize and deploy the trained model on an Arduino-compatible microcontroller.
Real-Time Command Execution – Enable the Arduino to recognize spoken commands and trigger actions (e.g., controlling LEDs, motors, or home automation devices).
Improve Accuracy – Fine-tune the model to enhance recognition accuracy, even with different accents and noise conditions.

Code

```
# remove the data dir if exists
!rm -r ./data
!rm -r ./edgeimpulse_dataset/


!rm -rf ./edgeimpulse_output_data

!wget http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

mkdir ./data

!tar xvf speech_commands_v0.01.tar.gz -C ./data

# Number of samples
num_samples = 1500
output_dir = './edgeimpulse_output_data'
words = ['one', 'two', 'three', 'four', 'noise', 'unknown']
```

```
import os
import random
import shutil
import librosa
import soundfile as sf
import numpy as np

# Download the zip dataset file
!wget https://cdn.edgeimpulse.com/datasets/keywords2.zip

# make the directory where to unzip the file
!mkdir ./edgeimpulse_dataset

# unzip the file
!unzip keywords2.zip -d ./edgeimpulse_dataset

!cp -r ./edgeimpulse_dataset/noise ./data/noise
!cp -r ./edgeimpulse_dataset/unknown/ ./data/unknown

# Copy num_samples samples to another directory
if not os.path.exists(output_dir):
  print("Making dir ["+output_dir+"]")
  os.mkdir(output_dir)


# Prepare the output directory structure
for word in words:
  dest = output_dir + '/' + word
  if not os.path.exists(dest):
    print("Create dest dir ", dest)
    os.mkdir(dest)


# Initialize random
random.seed();
for word in words:
  print("Selected word ["+word+"]")
  file_list = []
```

```
  for filename in os.listdir('./data/' + word):
   # print("Filename: ", filename)
    _, ext = os.path.splitext(filename)
    if (ext.lower() == '.wav'):
      # append the files
      file_list.append(filename)

  random.shuffle(file_list)
  # print("File size:", len(file_list))
  # Copy files from the origin directory to the output dir
  for i in range(num_samples):
    src = './data/' + word + '/' + file_list[i]
    dest = output_dir + '/' + word +'/' + word + '.' + file_list[i]
    # We can check if the file has the correct length
    s, sr = librosa.load(src, sr=16000, mono=True)
    # samle rate * sample time is in this case 16000
    if (len(s) < 16000):
      print("Padding the file...")
      s = np.append(s, np.zeros( int(16000 - len(s))))
      sf.write(dest, s, 16000)
    else:
      print("Copy file from ", src, " to ", dest)
      shutil.copyfile(src, dest)


print("Finished!")

# Check the number of sample in each directory
for word in words:
  dirname = output_dir + '/' + word
  print("Dir ", dirname, "Samples ", len(os.listdir(dirname)))

!npm install -g --unsafe-perm edge-impulse-cli

# API Key
api_key =
'ei_166cbf36b4923b2e8b67d71c164daf3305cbcaab4decf40a55ca3fb689775a91'
for word in words:
  sample_dir = output_dir + '/' + word + '/*.wav'
  print("Uploading files from ", sample_dir)
```

```
  cmd = 'edge-impulse-uploader --api-key ' + api_key + ' --label ' + word
+ ' ' + sample_dir
  os.system(cmd)


print("Done!");
```

Results:

Accuracy on Training Data:

Last training performance (validation set)

ACCURACY
83.8%

LOSS
0.53

Confusion matrix (validation set)

	FOUR	NOISE	ONE	THREE	TWO	UNKNOWN
FOUR	88.7%	2.9%	1.7%	1.3%	2.5%	2.9%
NOISE	0.4%	95.2%	0%	0.9%	0%	3.5%
ONE	3.1%	4.8%	87.7%	0.4%	0%	4.0%
THREE	0.4%	2.0%	0%	96.7%	0.4%	0.4%
TWO	1.6%	5.7%	0%	2.0%	89.9%	0.8%
UNKNOWN	12.9%	7.9%	8.3%	15.4%	10.4%	45.2%
F1 SCORE	0.86	0.87	0.88	0.89	0.89	0.58

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	0.97
Weighted average Precision	0.84

Accuracy on Training Data:

ACCURACY
74.66%

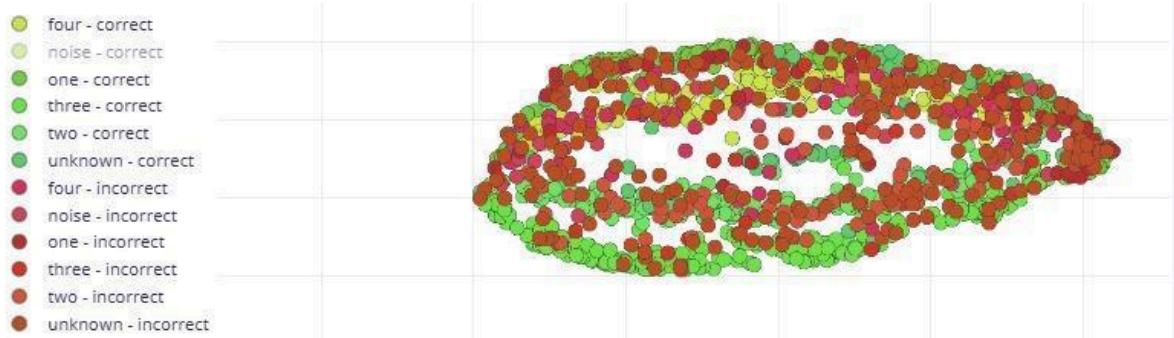
Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.97
Weighted average Precision	0.85
Weighted average Recall	0.85
Weighted average F1 score	0.85

Confusion matrix

	FOUR	NOISE	ONE	THREE	TWO	UNKNOWN	UNCERTAIN
FOUR	77.2%	1.3%	0.3%	0%	0%	0.3%	20.8%
NOISE	0.3%	91.7%	0%	0.3%	0%	0%	7.6%
ONE	1%	1.3%	82.7%	0%	0%	1%	14%
THREE	0%	2.3%	0%	92.3%	0%	0%	5.3%
TWO	0%	2.0%	0%	1.0%	75.6%	0%	21.4%
UNKNOWN	4.6%	4.3%	4.3%	5.3%	4.3%	25.3%	52.0%
F1 SCORE	0.84	0.91	0.88	0.93	0.84	0.40	

Feature explorer



Unit-2 Task(R-Programming)

Task 1 : Explain Basic Data Structure in R.

Basic Data Structures in R:
R has several basic data structures that are essential for organizing and manipulating data. Below are the key ones:

Vectors:
A vector is a sequence of data elements of the same type.
Example: x <- c(1, 2, 3, 4, 5) (numeric vector)
Matrices:
A matrix is a two-dimensional data structure where elements are arranged in rows and columns.
Example: matrix_data <- matrix(1:6, nrow = 2, ncol = 3)
Arrays:
An array is an extension of a matrix, with more than two dimensions.
Example: array_data <- array(1:8, dim = c(2, 2, 2))
Data Frames:
A data frame is a table or 2D array-like structure that can hold different types of data (numeric, character, etc.) in different columns.
Example: df <- data.frame(Name = c("John", "Alice"), Age = c(28, 24))
Lists:
A list is an ordered collection of elements that can be of different types (vectors, matrices, data frames, etc.).
Example: my_list <- list(name = "John", age = 28, scores = c(90, 85, 88))
Factors:
A factor is used to represent categorical data with a set of levels.
Example: gender <- factor(c("Male", "Female", "Male"))

Task 2: Implement Linear Regression in R and Visualize the results.

Step 1: Load the necessary library:
library(ggplot2)

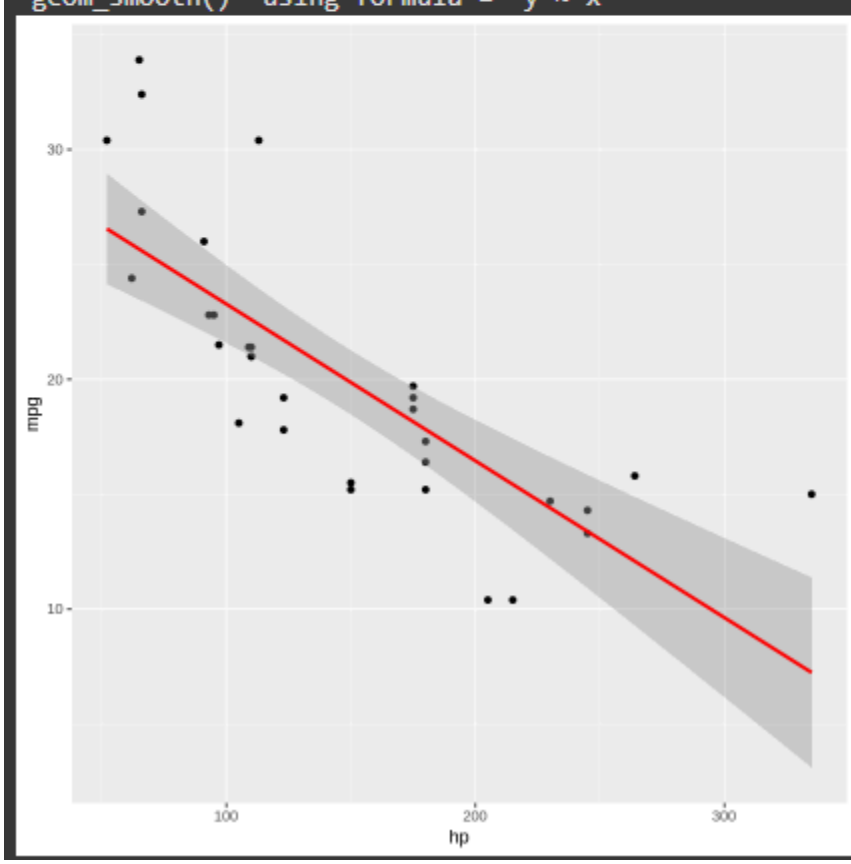
Step 2 :Create a simple dataset: For simplicity, let's use the mtcars dataset, which is built into R.
data(mtcars)

Step 3 : Fit a linear model (e.g., predicting mpg from hp):
lm_model <- lm(mpg ~ hp, data = mtcars)
summary(lm_model)

Step 4 : Visualize the results:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +  # Scatter plot
  geom_smooth(method = "lm", col = "red")  # Add regression line
```

```
Call:
lm(formula = mpg ~ hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-5.7121 -2.1122 -0.8854  1.5819  8.2360 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
hp          -0.06823    0.01012  -6.742 1.79e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6024,	Adjusted R-squared:  0.5892 
F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
`geom_smooth()` using formula = 'y ~ x'
```



Task 3 : Implement Logistic Regression in R and Visualize the results.

Step 1 :Load necessary libraries:

```
library(ggplot2)
```

Step 2 :Use the mtcars dataset: Let's convert a continuous variable to a factor for classification. We'll predict whether a car has mpg greater than 20 or not.

```
mtcars$mpg_high <- ifelse(mtcars$mpg > 20, 1, 0)
```

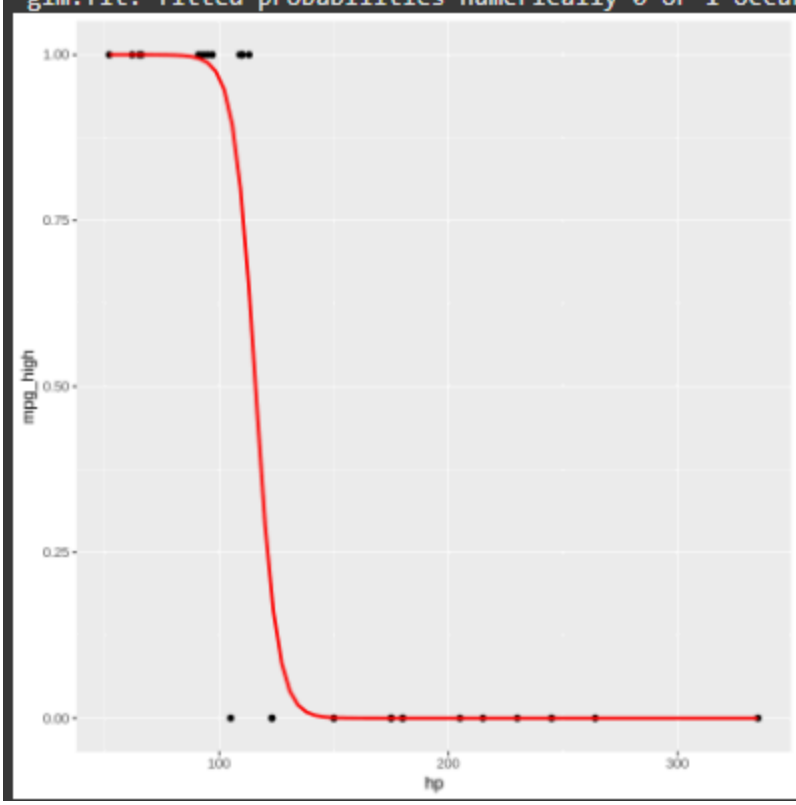
Step 3 :Fit the logistic regression model:

```
logit_model <- glm(mpg_high ~ hp + wt, data = mtcars, family = binomial)
summary(logit_model)
```

Step 4 :Visualize the logistic regression results:

```
ggplot(mtcars, aes(x = hp, y = mpg_high)) +
  geom_point() +
  stat_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE, col = "red")
```

```
Call:
glm(formula = mpg_high ~ hp + wt, family = binomial, data = mtcars)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    894.228 365884.162   0.002    0.998
hp              -2.021    858.062  -0.002    0.998
wt            -202.865  84688.218  -0.002    0.998

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4.3860e+01  on 31  degrees of freedom
Residual deviance: 1.1156e-08  on 29  degrees of freedom
AIC: 6

Number of Fisher Scoring iterations: 25
`geom_smooth()` using formula = 'y ~ x'
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
```



Task 4: Implement any Machine learning Algorithm along with feature selection and data visualization on any dataset of your choice.

Implementing a Random Forest model along with feature selection using the caret package. We'll use the famous Iris dataset.

```
Install necessary libraries
install.packages("caret")
install.packages("randomForest")
install.packages("ggplot2")
install.packages("dplyr")

Load the libraries
library(caret)
library(randomForest)
library(ggplot2)
library(dplyr)

Step 1: Load the Iris dataset
data(iris)
head(iris)

Step 2: Feature Selection using Recursive Feature Elimination (RFE)
ctrl <- rfeControl(functions=rfFuncs, method="cv", number=10)  # Cross-validation setup

Perform the RFE algorithm to select top features
rfe_result <- rfe(iris[, 1:4], iris[, 5], sizes=c(1:4), rfeControl=ctrl)

Output selected features
print(rfe_result)

Step 3: Split the Data into Training and Test Sets
set.seed(123)  # For reproducibility
trainIndex <- createDataPartition(iris$Species, p=0.8, list=FALSE)
train_data <- iris[trainIndex, ]
test_data <- iris[-trainIndex, ]

Step 4: Train a Random Forest Classifier
rf_model <- randomForest(Species ~ ., data=train_data)

Check the model's summary
print(rf_model)
```

Step 5: Evaluate the Model

```
rf_pred <- predict(rf_model, test_data)

Create a confusion matrix to evaluate accuracy
conf_matrix <- confusionMatrix(rf_pred, test_data$Species)

Print confusion matrix and accuracy
print(conf_matrix)
```

Step 6: Visualize Feature Importance

```
Plot feature importance
importance(rf_model)
varImpPlot(rf_model, main="Random Forest - Feature Importance")
```

Step 7: Visualize Decision Boundaries

```
We use only the first two features for the visualization
ggplot(test_data, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size=3) +
  stat_ellipse() +
  theme_minimal() +
  ggtitle("Decision Boundaries - Random Forest")
```

Outputs

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

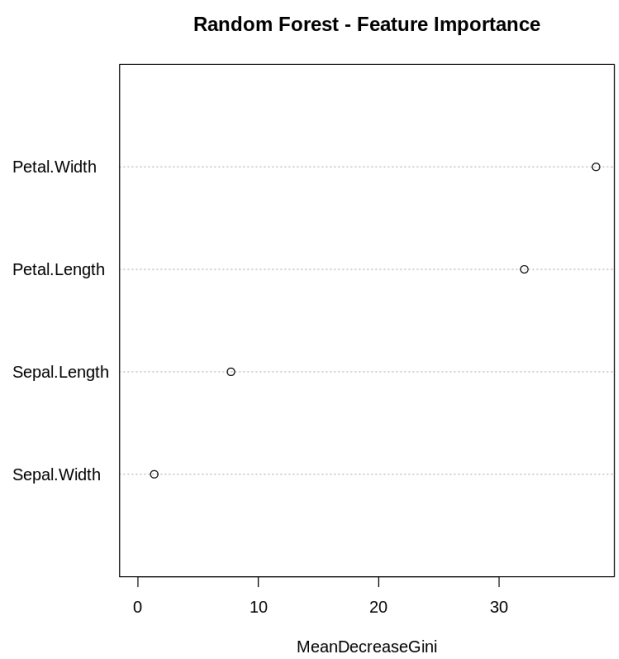
Recursive feature selection

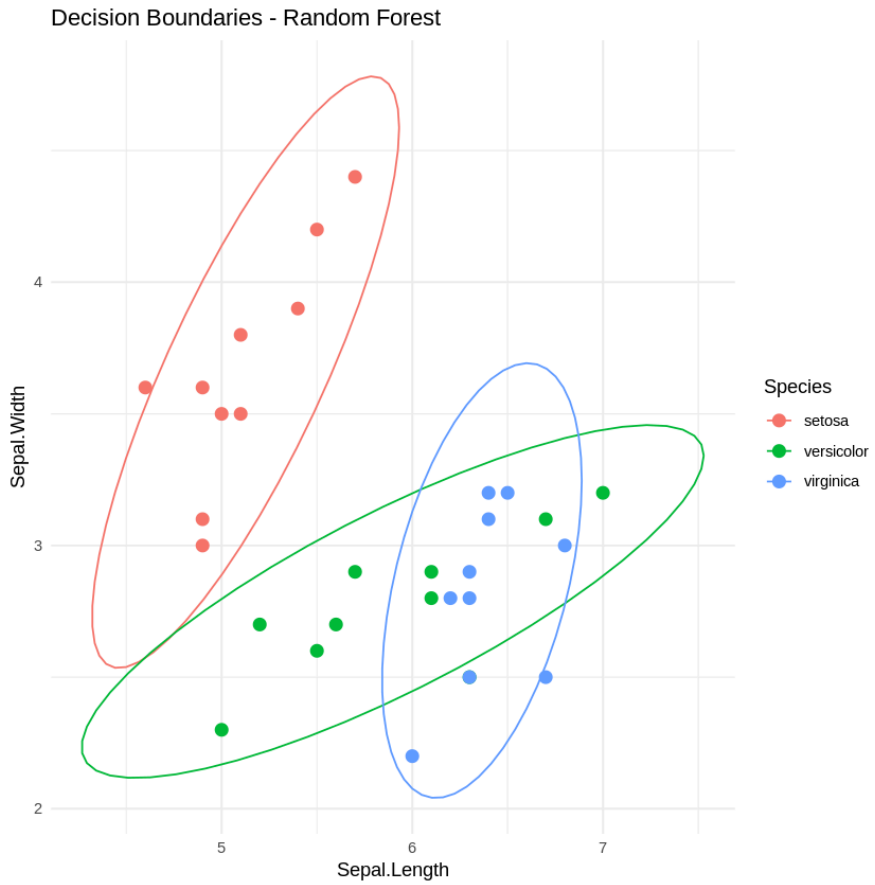
```
Outer resampling method: Cross-Validated (10 fold)

Resampling performance over subset size:

 Variables Accuracy Kappa AccuracySD KappaSD Selected
         1   0.9133  0.87    0.04500 0.06749
         2   0.9467  0.92    0.04216 0.06325        *
         3   0.9400  0.91    0.04919 0.07379
         4   0.9333  0.90    0.05443 0.08165

The top 2 variables (out of 2):
   Petal.Length, Petal.Width


Call:
 randomForest(formula = Species ~ ., data = train_data)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 5%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         40          0         0       0.000
versicolor      0         37         3       0.075
virginica       0          3        37       0.075
Confusion Matrix and Statistics

           Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.7793, 0.9918)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : 8.747e-12

                  Kappa : 0.9

 Mcnemar's Test P-Value : NA
```

Statistics by Class:

	Class: setosa	Class: versicolor	Class: virginica
Sensitivity	1.0000	1.0000	0.8000
Specificity	1.0000	0.9000	1.0000
Pos Pred Value	1.0000	0.8333	1.0000
Neg Pred Value	1.0000	1.0000	0.9091
Prevalence	0.3333	0.3333	0.3333
Detection Rate	0.3333	0.3333	0.2667
Detection Prevalence	0.3333	0.4000	0.2667
Balanced Accuracy	1.0000	0.9500	0.9000

	MeanDecreaseGini
Sepal.Length	7.729199
Sepal.Width	1.356956
Petal.Length	32.096689
Petal.Width	38.053952



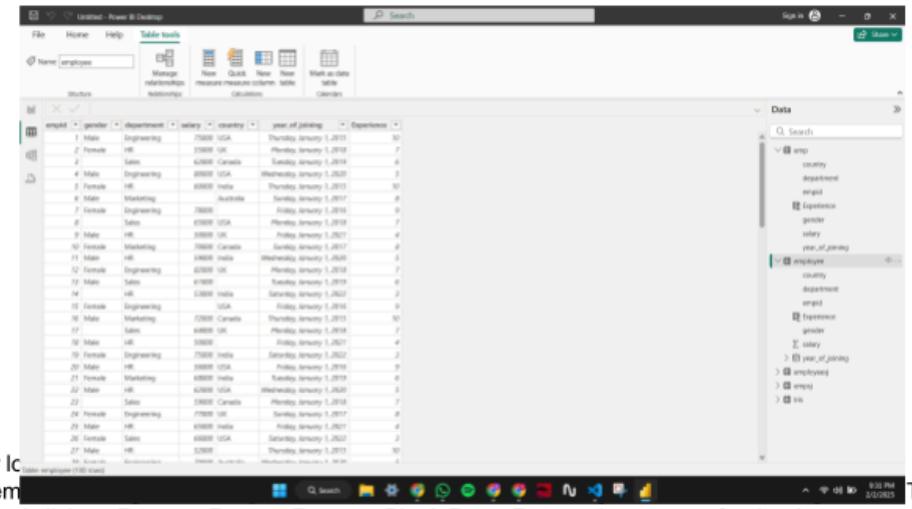

Decision Boundaries - Random Forest
Sepal.Width
4
3
2
5
6
7
Sepal.Length
Species
setosa
versicolor
virginica

Unit 3: Power BI

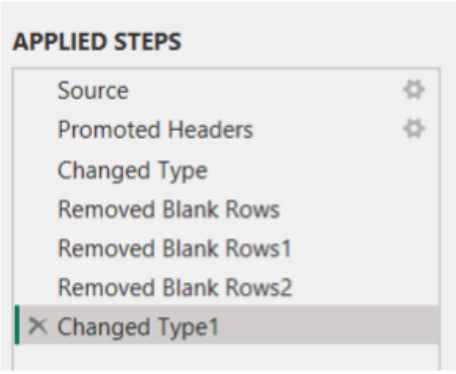
1.Explore Power View, Power Query

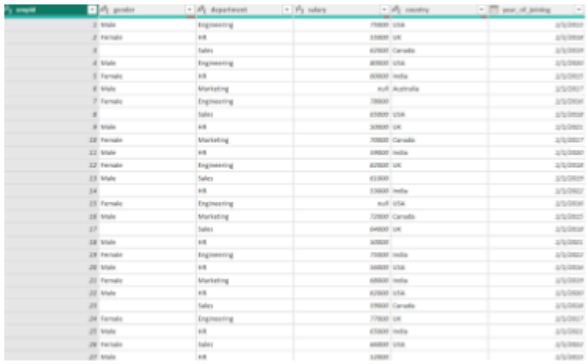
- **Create a table Employee(empid, gender, department, salary, country, year_of_joining) connect to Employee data file.**

1. Open Power BI Desktop.
2. Click on Home > Get Data.
3. Choose the type of file (e.g., Excel or CSV) where your Employee data is stored.
4. Browse and load the file with the Employee data containing columns like empid, gender, department, salary, country, and year_of_joining.

- 

After lo
To rem Transform tab and click on Remove Rows > Remove Blank Rows.Repeat the process for the department column.



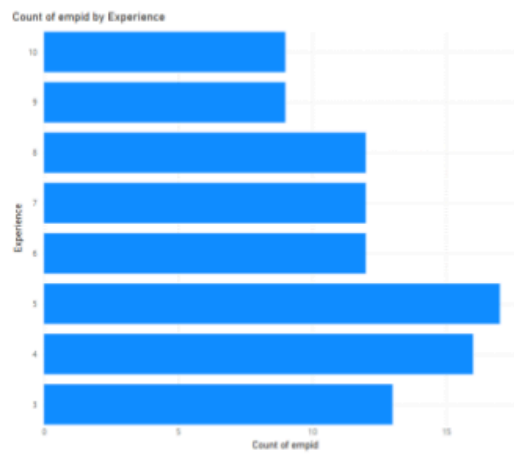


- Extract year_of_joining column and visualize number of employees w.r.t year of experience in the company

.

1. In Power BI, go to the **Data** view.
2. Create a new calculated column for **Experience** (years of experience). Assuming `year_of_joining` is in date format:
3. In the **Modeling** tab, click **New Column** and enter the formula:
   `Experience = YEAR(TODAY()) - YEAR(Employee[year_of_joining])`
4. This will calculate the years of experience for each employee based on the current year.
5. To visualize Go to the **Report** view.
6. Choose a **Bar Chart** or **Column Chart** visualization.
7. Drag the **Experience** field to the **Axis** and **empid** field (or a count of employee IDs) to the **Values** field.
8. You should now see a visualization showing the number of employees corresponding to each year of experience.

empid	gender	department	salary	country	year_of_joining	Experience
1	Male	Engineering	75000	USA	Thursday, January 1, 2015	10
2	Female	HR	55000	UK	Monday, January 1, 2018	7
3		Sales	62000	Canada	Tuesday, January 1, 2019	6
4	Male	Engineering	80000	USA	Wednesday, January 1, 2020	5
5	Female	HR	60000	India	Thursday, January 1, 2015	10
6	Male	Marketing		Australia	Sunday, January 1, 2017	8
7	Female	Engineering	78000		Friday, January 1, 2016	9
8		Sales	65000	USA	Monday, January 1, 2018	7
9	Male	HR	50000	UK	Friday, January 1, 2021	4
10	Female	Marketing	70000	Canada	Sunday, January 1, 2017	8
11	Male	HR	59000	India	Wednesday, January 1, 2020	5
12	Female	Engineering	82000	UK	Monday, January 1, 2018	7
13	Male	Sales	61000		Tuesday, January 1, 2019	6
14		HR	53000	India	Saturday, January 1, 2022	3
15	Female	Engineering		USA	Friday, January 1, 2016	9
16	Male	Marketing	72000	Canada	Thursday, January 1, 2015	10
17		Sales	64000	UK	Monday, January 1, 2018	7
18	Male	HR	50000		Friday, January 1, 2021	4
19	Female	Engineering	75000	India	Saturday, January 1, 2022	3
20	Male	HR	56000	USA	Friday, January 1, 2016	9
21	Female	Marketing	68000	India	Tuesday, January 1, 2019	6
22	Male	HR	62000	USA	Wednesday, January 1, 2020	5
23		Sales	59000	Canada	Monday, January 1, 2018	7
24	Female	Engineering	77000	UK	Sunday, January 1, 2017	8

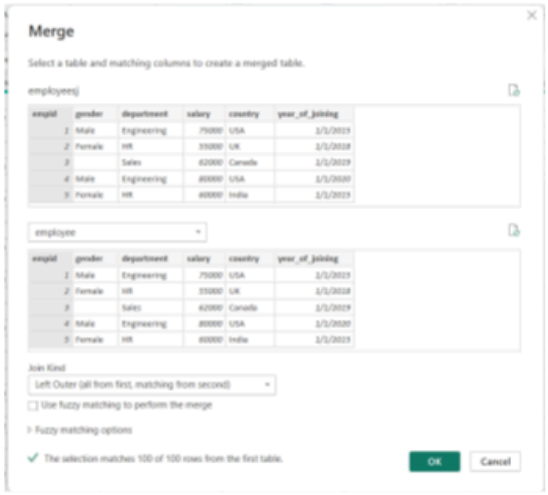


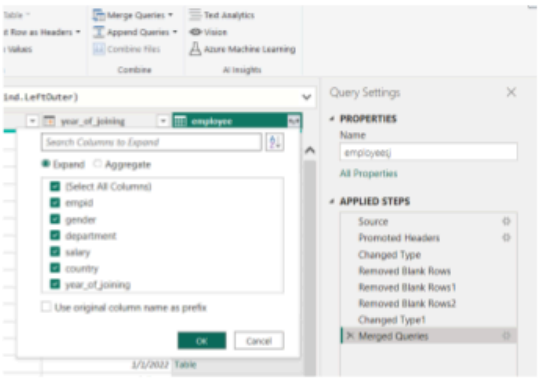
- Perform self-join using Power Query.

Go back to the **Power Query Editor**.
Right-click on the **Employee** query in the Queries pane and select **Duplicate** to create a copy of the table.
Rename the duplicate table as `Employee_Self_Join`.
Now, perform the self-join:

1. Go to the **Home** tab and click on **Merge Queries**.
2. Select the original `Employee` table and the `Employee_Self_Join` table.
3. Choose the columns that match between the two tables for the join (e.g., `empid`).
4. Select the type of join (Inner, Left Outer, etc.).

After merging, expand the newly merged columns to bring in additional information from the joined table (like salary, gender, etc.).

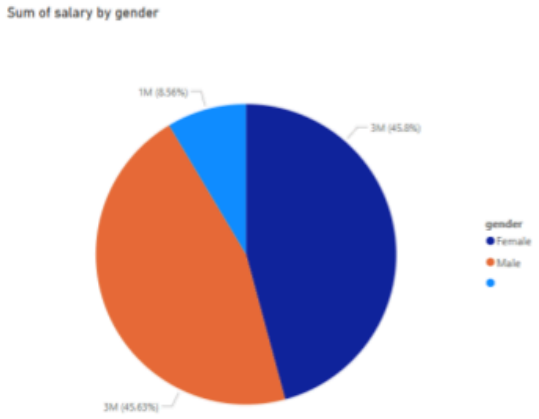




empid	gender	department	salary	country	year_of_joining	empid.1	gender.1	department.1	salary.1	country.1	year_of_joining.1
1	Male	Engineering	75000	USA	Thursday, January 1, 2015	1	Male	Engineering	75000	USA	Thursday, January 1, 2015
2	Female	HR	55000	UK	Monday, January 1, 2018	2	Female	HR	55000	UK	Monday, January 1, 2018
3		Sales	62000	Canada	Tuesday, January 1, 2019	3		Sales	62000	Canada	Tuesday, January 1, 2019
4	Male	Engineering	80000	USA	Wednesday, January 1, 2020	4	Male	Engineering	80000	USA	Wednesday, January 1, 2020
5	Female	HR	60000	India	Thursday, January 1, 2015	5	Female	HR	60000	India	Thursday, January 1, 2015
6	Male	Marketing		Australia	Sunday, January 1, 2017	6	Male	Marketing		Australia	Sunday, January 1, 2017
7	Female	Engineering	78000		Friday, January 1, 2016	7	Female	Engineering	78000		Friday, January 1, 2016
8		Sales	65000	USA	Monday, January 1, 2018	8		Sales	65000	USA	Monday, January 1, 2018
9	Male	HR	50000	UK	Friday, January 1, 2021	9	Male	HR	50000	UK	Friday, January 1, 2021
10	Female	Marketing	70000	Canada	Sunday, January 1, 2017	10	Female	Marketing	70000	Canada	Sunday, January 1, 2017
11	Male	HR	59000	India	Wednesday, January 1, 2020	11	Male	HR	59000	India	Wednesday, January 1, 2020
12	Female	Engineering	82000	UK	Monday, January 1, 2018	12	Female	Engineering	82000	UK	Monday, January 1, 2018
13	Male	Sales	61000		Tuesday, January 1, 2019	13	Male	Sales	61000		Tuesday, January 1, 2019
14		HR	53000	India	Saturday, January 1, 2022	14		HR	53000	India	Saturday, January 1, 2022
15	Female	Engineering		USA	Friday, January 1, 2016	15	Female	Engineering		USA	Friday, January 1, 2016
16	Male	Marketing	72000	Canada	Thursday, January 1, 2015	16	Male	Marketing	72000	Canada	Thursday, January 1, 2015
17		Sales	64000	UK	Monday, January 1, 2018	17		Sales	64000	UK	Monday, January 1, 2018
18	Male	HR	50000		Friday, January 1, 2021	18	Male	HR	50000		Friday, January 1, 2021
19	Female	Engineering	75000	India	Saturday, January 1, 2022	19	Female	Engineering	75000	India	Saturday, January 1, 2022
20	Male	HR	56000	USA	Friday, January 1, 2016	20	Male	HR	56000	USA	Friday, January 1, 2016
21	Female	Marketing	68000	India	Tuesday, January 1, 2019	21	Female	Marketing	68000	India	Tuesday, January 1, 2019
22	Male	HR	62000	USA	Wednesday, January 1, 2020	22	Male	HR	62000	USA	Wednesday, January 1, 2020
23		Sales	59000	Canada	Monday, January 1, 2018	23		Sales	59000	Canada	Monday, January 1, 2018
24	Female	Engineering	77000	UK	Sunday, January 1, 2017	24	Female	Engineering	77000	UK	Sunday, January 1, 2017
25	Male	HR	63000	India	Friday, January 1, 2021	25	Male	HR	63000	India	Friday, January 1, 2021
26	Female	Sales	66000	USA	Saturday, January 1, 2022	26	Female	Sales	66000	USA	Saturday, January 1, 2022
27	Male	HR	52000		Thursday, January 1, 2015	27	Male	HR	52000		Thursday, January 1, 2015

- Aggregate salary with gender and Visualize using Pie chart.

1. In **Report** view, choose a **Pie Chart** visualization.
2. Drag the **gender** field to the **Legend** and **salary** field to the **Values** field.
3. Power BI will automatically aggregate the salary by gender, and you will see the Pie Chart showing the proportion of total salary by gender.



2. Visualize the result of any Machine Learning algorithm on any dataset of your choice in PowerBI.

1. Load the Iris Dataset into Power BI

1. **Get the Iris Dataset**: You can download the Iris dataset as a CSV file or use it directly from a Python package like `sklearn`. For the CSV version, you can find it here on Kaggle.
2. **Import the Dataset into Power BI**:
   - Go to the **Home** tab in Power BI and click on **Get Data**.
   - Select **CSV** and browse to where you saved the Iris dataset file, then click **Open**.
   - After importing the dataset, click **Transform Data** to go to the **Power Query Editor**.

2. Run the Python Script to Apply the ML Algorithm

1. **Enable Python Script**:
   - In Power BI, go to **File** > **Options and Settings** > **Options**.
   - Under **Global** > **Python scripting**, make sure that you have Python installed and the correct path set to where Python is installed on your machine (for example, `C:\Python39`).
2. **Go to Transform Data**:
   - After loading the dataset, click on the **Transform Data** button to open the **Power Query Editor**.
3. **Run Python Script**:
   - In the **Power Query Editor**, go to the **Home tab** and click **Run Python Script**.
   - In the dialog box that opens, enter the following code:

```
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming your dataset is called 'dataset'
dataset = pd.DataFrame(dataset)  # Convert to DataFrame if necessary
X = dataset.drop('species', axis=1)  # Features (remove target column)
y = dataset['species']  # Target variable (species)

# Split the data into training and test sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict using the model
predictions = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, predictions)

# Add predictions to the dataset for visualization in Power BI
dataset_predictions = X_test.copy()
dataset_predictions['Actual'] = y_test
dataset_predictions['Predicted'] = predictions

# Return the dataset with predictions and accuracy
dataset_predictions['Accuracy'] = accuracy
```

This code:

- Splits the Iris dataset into features (`X`) and the target (`species`).
- Trains a **Random Forest Classifier** on the training data.
- Makes predictions on the test data.
- Calculates the accuracy of the model.
- Returns the dataset with the **actual** and **predicted** values, along with the **accuracy**.

2. **Click OK** to run the script. This will return the output dataset with the predicted values added to it.

Visualize the Results in Power BI

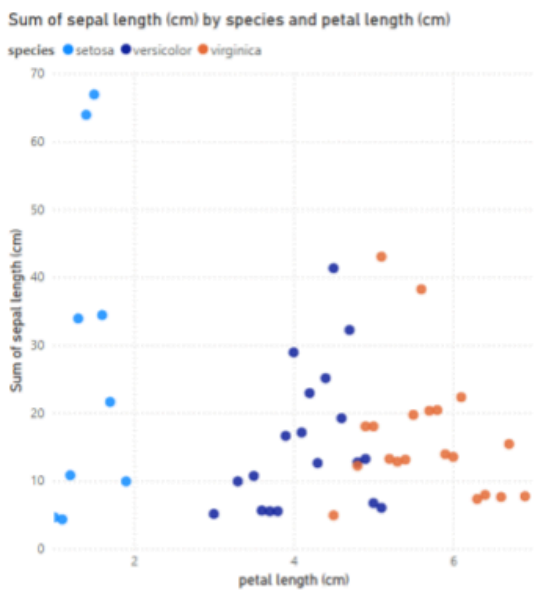
1. Visualize the Predicted vs. Actual Values (Classification)

Now that you have the model's predictions and accuracy, you can create visualizations to show the performance of the model.

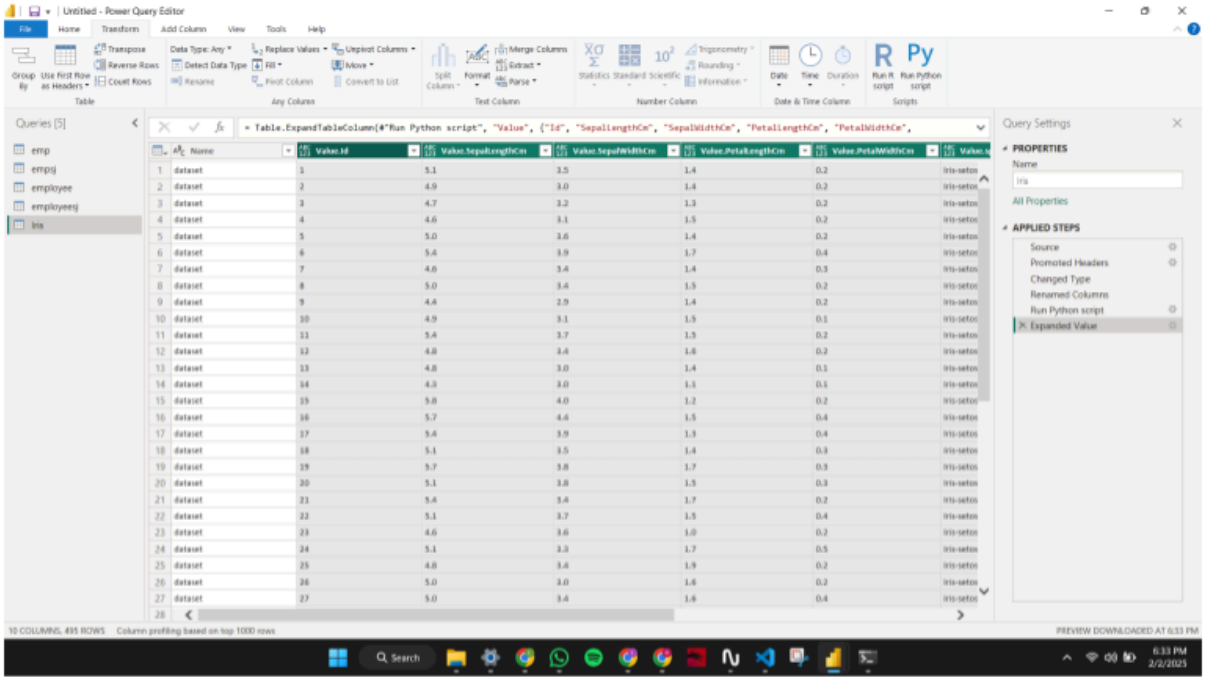
Scatter Plot (for Regression or Comparing True vs. Predicted Values):

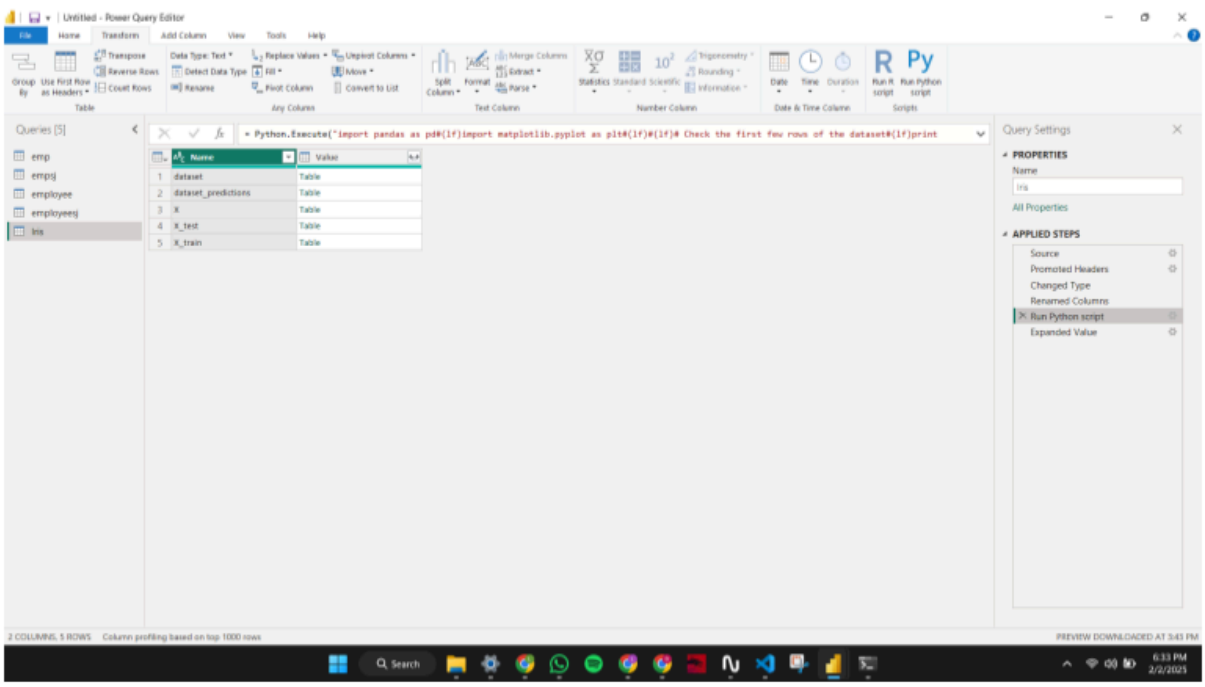
Create a **Scatter Plot** to show how close the predictions are to the actual values.

- On the **Visualizations** pane, select **Scatter Chart**.
- Add **Actual** values to the **Y-axis** and **Predicted** values to the **X-axis**.
- This will show a scatter plot comparing the predicted and actual values.



Column1	Sum of setosa	Sum of versicolor	Sum of virginica
setosa	10	0	0
versicolor	0	9	0
virginica	0	0	11
Total	**10**	**9**	**11**


Untitled - Power Query Editor
= Table.ExpandTableColumn(#"Run Python script", "Value", {"Id", "SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm",
Queries [5]
emp
empsj
employee
employeesj
Iris
Query Settings
PROPERTIES
Name
Iris
All Properties
APPLIED STEPS
Source
Promoted Headers
Changed Type
Renamed Columns
Run Python script
Expanded Value
10 COLUMNS, 495 ROWS
Column profiling based on top 1000 rows
PREVIEW DOWNLOADED AT 6:33 PM


Untitled - Power Query Editor
= Python.Execute("import pandas as pd#(lf)import matplotlib.pyplot as plt#(lf)#(lf)# Check the first few rows of the dataset#(lf)print
Queries [5]
emp
empsj
employee
employeesj
Iris
Name
Value
dataset
dataset_predictions
X
X_test
X_train
Table
Query Settings
PROPERTIES
Name
Iris
All Properties
APPLIED STEPS
Source
Promoted Headers
Changed Type
Renamed Columns
Run Python script
Expanded Value
2 COLUMNS, 5 ROWS
Column profiling based on top 1000 rows
PREVIEW DOWNLOADED AT 3:45 PM

Unit-4 Spark

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum
from google.colab import files

# Initialize the Spark session
spark = SparkSession.builder.appName("Employee Data Processing").getOrCreate()

# Step 1: Read the uploaded CSV file
file_path = "/content/emp.csv"  # Path to the uploaded file in Google Colab
sales_df = spark.read.csv(file_path, header=True, inferSchema=True)

# Step 2: Data Cleaning - Handling missing values and removing duplicates
sales_df_clean = sales_df.dropna()  # Drop rows with missing values
sales_df_clean = sales_df_clean.dropDuplicates()  # Remove duplicates

# Step 3: Calculate the total salary per department
total_salary_df = sales_df_clean.groupBy("department").agg(
    sum("salary").alias("total_salary")
)

# Step 4: Output the results to a new CSV file with overwrite mode (in case it exists)
output_path = "/content/total_salary_per_department.csv"
total_salary_df.write.csv(output_path, header=True, mode="overwrite")

# Step 5: Show the results (optional)
total_salary_df.show()

# Step 6: Download the resulting CSV file
import os
from google.colab import files

# List the files in the output directory
output_directory = "/content/total_salary_per_department.csv"
output_files = os.listdir(output_directory)

# Print the files in the output directory (for debugging)
print("Files in output directory:", output_files)

# Find the first part file (e.g., part-00000)
part_file = [f for f in output_files if f.startswith("part-")][0]
```

```
# Construct the full path to the part file
part_file_path = os.path.join(output_directory, part_file)

# Download the part file
files.download(part_file_path)

# Stop the Spark session
spark.stop()
```

OUTPUT

```
Files in output directory: ['._SUCCESS.crc',
'.part-00000-cdc9e647-a17b-4721-a3ef-2b212062c632-c000.csv.crc',
'part-00000-cdc9e647-a17b-4721-a3ef-2b212062c632-c000.csv', '_SUCCESS']
```

```
root
 |-- empid: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- department: string (nullable = true)
 |-- salary: integer (nullable = true)
 |-- country: string (nullable = true)
 |-- year_of_joining: integer (nullable = true)

+----------+------------+
|department|total_salary|
+----------+------------+
|        HR|      187000|
|   Finance|      266000|
| Marketing|      164000|
|        IT|      293000|
+----------+------------+
```

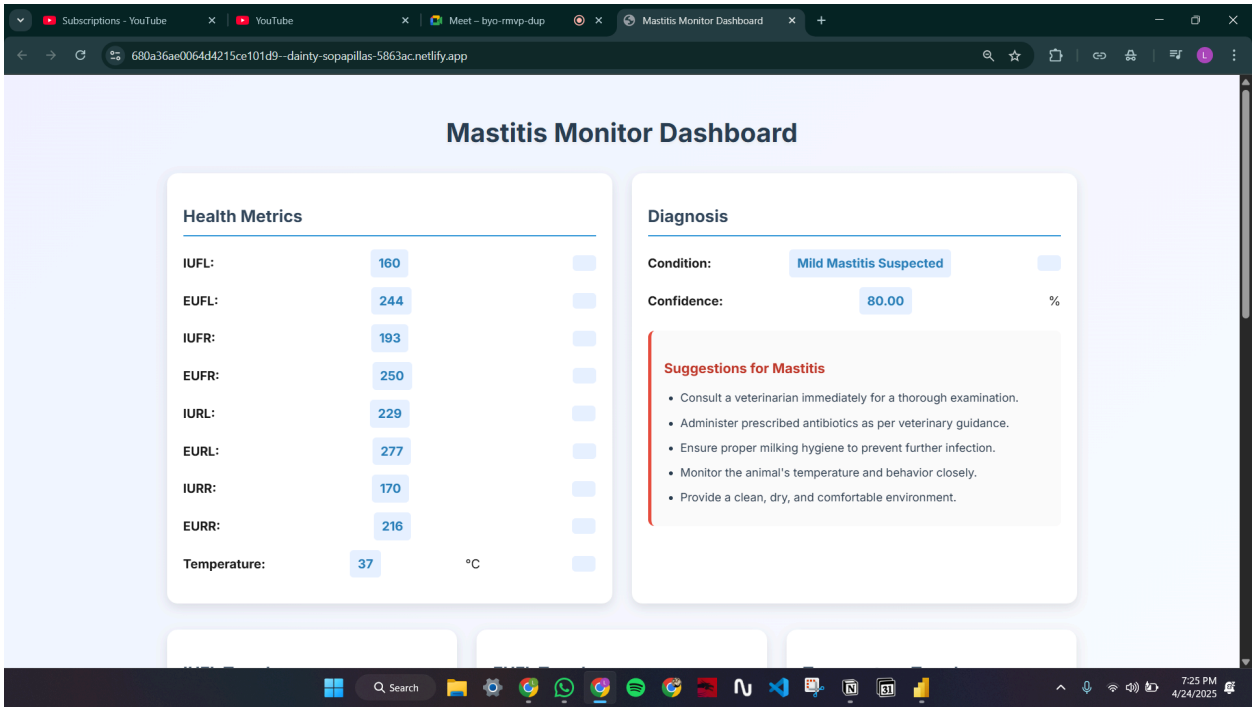
part-00000-cdc9e647-a17b-4721-a3ef-2b212062c632-c000.csv

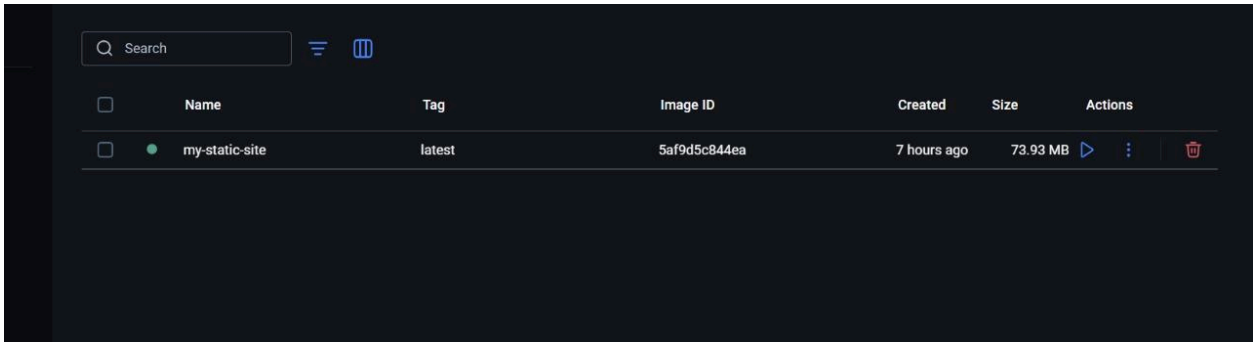
C: > Users > lalfa > OneDrive > Desktop > part-00000-cdc9e647-a17b-4721-a3ef-2b212062c632-c000.csv

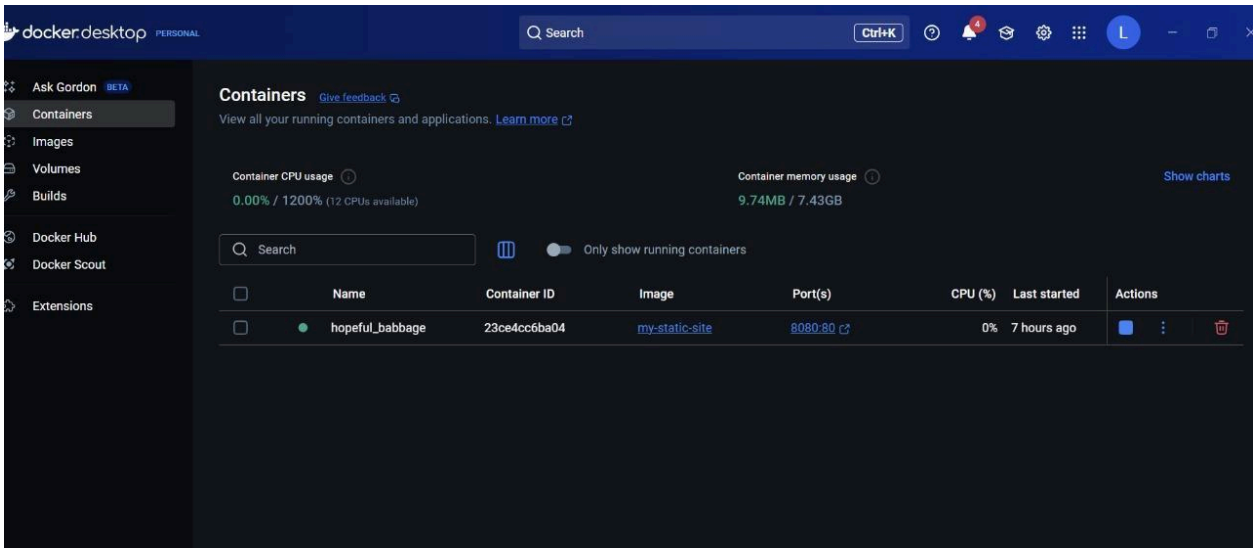
```
department,total_salary
HR,187000
Finance,266000
Marketing,164000
IT,293000
```

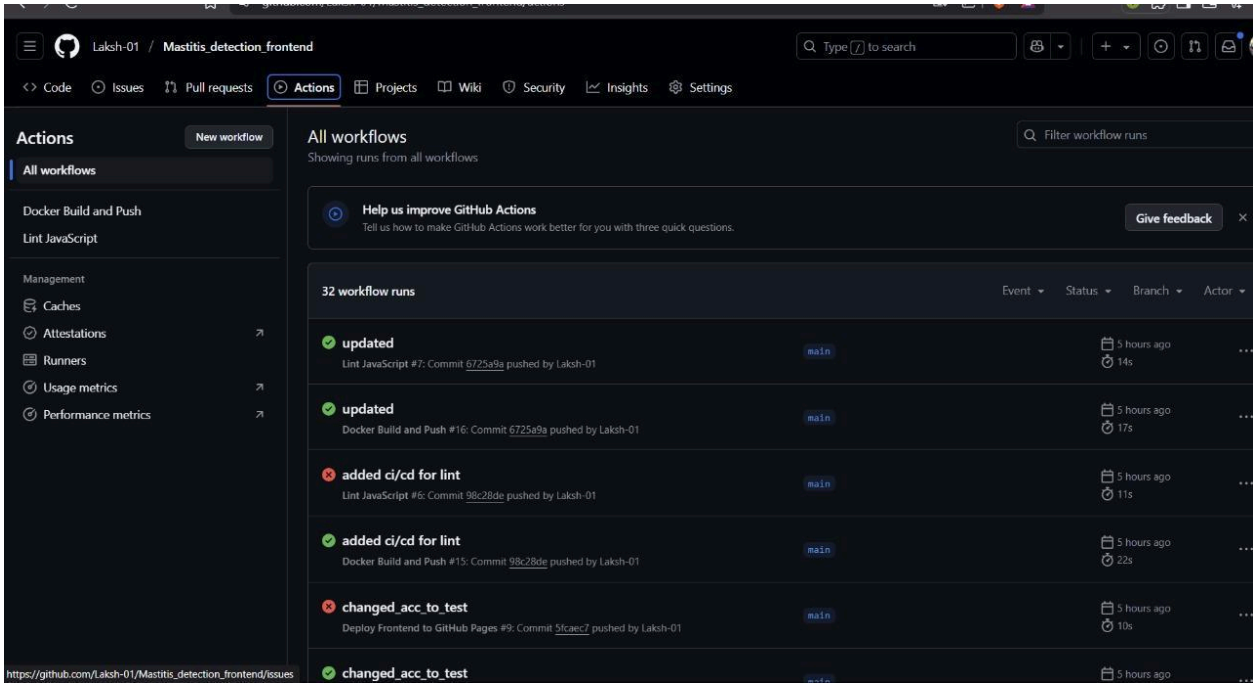
Unit 5 : DevOps

- Created image (firmware or AI model) after training.
- Wrote Dockerfile and added the image to a Docker container with required dependencies.
- Built and ran the container locally to ensure environment consistency.
- Integrated linting (pylint/cpplint) via GitHub Actions to enforce code quality.
- Implemented CI/CD pipeline in GitHub for auto build, test, and Docker run.
- Ran Wokwi simulation to test embedded code behavior post-deployment.
- Validated the pipeline with automated simulation outputs and linked logs.




Search
Name
Tag
Image ID
Created
Size
Actions
my-static-site
latest
5af9d5c844ea
7 hours ago
73.93 MB


docker desktop PERSONAL
Search
Ctrl+K
Ask Gordon BETA
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Containers
Give feedback
View all your running containers and applications. Learn more
Container CPU usage
0.00% / 1200% (12 CPUs available)
Container memory usage
9.74MB / 7.43GB
Show charts
Search
Only show running containers
Name
Container ID
Image
Port(s)
CPU (%)
Last started
Actions
hopeful_babbage
23ce4cc6ba04
my-static-site
8080:80
0%
7 hours ago


Laksh-01 / Mastitis_detection_frontend
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Actions
New workflow
All workflows
Docker Build and Push
Lint JavaScript
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
All workflows
Showing runs from all workflows
Filter workflow runs
Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.
Give feedback
32 workflow runs
Event
Status
Branch
Actor
updated
Lint JavaScript #7: Commit 6725a9a pushed by Laksh-01
main
5 hours ago
14s
updated
Docker Build and Push #16: Commit 6725a9a pushed by Laksh-01
main
5 hours ago
17s
added ci/cd for lint
Lint JavaScript #6: Commit 98c28de pushed by Laksh-01
main
5 hours ago
11s
added ci/cd for lint
Docker Build and Push #15: Commit 98c28de pushed by Laksh-01
main
5 hours ago
22s
changed_acc_to_test
Deploy Frontend to GitHub Pages #9: Commit 5fcaec7 pushed by Laksh-01
main
5 hours ago
10s
changed_acc_to_test
5 hours ago
https://github.com/Laksh-01/Mastitis_detection_frontend/issues