

NAAN MUDHALVAN PROJECT REPORT



PROJECT TITLE

Equipred– Advanced Equipment Failure Prediction

PROBLEM STATEMENT

An industrial facility wants to minimize downtime by predicting when equipment is likely to fail. The goal is to develop an AI-based predictive maintenance system that can analyze equipment data to forecast potential failures.

AIM

The aim of the project is to develop an AI-based predictive maintenance system for industrial equipment. This system aims to minimize downtime by forecasting potential equipment failures, enabling proactive maintenance actions. The project involves collecting and preprocessing equipment sensor data, training machine learning models for failure prediction, implementing real-time monitoring and alerting systems, and developing a dashboard for visualizing equipment health. Ultimately, the goal is to minimize downtime through proactive maintenance, predict potential equipment failures before they occur, and provide actionable insights for maintenance planning, thus enhancing operational efficiency and reliability in industrial settings.

OBJECTIVES

- Develop a predictive maintenance model using machine learning techniques to anticipate equipment failures and minimize downtime.
- Implement a real-time monitoring system for equipment health, enabling timely detection of anomalies and proactive maintenance actions.
- Generate alerts and recommendations for maintenance actions based on the analysis of equipment data, facilitating efficient resource allocation and maintenance planning.
- Develop a user-friendly dashboard for visualizing equipment health and performance metrics, providing actionable insights for informed decision-making.

TECH STACK



EXISTING WORK

In contrast to existing systems where Equipment Failure Prediction is primarily treated as a classification task, our proposed system expands beyond binary outcomes by accurately generating probabilities through a thorough analysis of contributing features to Equipment Failure. Additionally, an interactive dashboard has been developed to not only predict failure probabilities but also provide insights into Equipment health for proactive maintenance, a feature absent in existing systems. Furthermore, our system incorporates an Alert system, a crucial component for timely intervention and maintenance scheduling, thereby enhancing operational efficiency. Moreover, the accuracy of our predictive model has been enhanced through Competitive Model Analysis, surpassing the performance of models utilized in existing systems. Through these advancements, a more comprehensive, accurate, and actionable solution for Equipment Failure Prediction and Maintenance Management is provided.

PROPOSED WORK

The project involved the following steps:

1. Data Collection:

- Sensor data was collected from the IoT sensor dataset.

2. Data Preprocessing:

- Various preprocessing techniques were applied, including handling missing values, outlier detection, and normalization using StandardScaler.

- Important features were identified through correlation analysis.

3. Model Training:

- The dataset was trained using multiple machine learning models, such as Decision Tree, Random Forest, Logistic Regression, K-Nearest Neighbors (KNN), Gradient Boosting, and Stacking.
- Performance metrics (Accuracy, Precision, Recall, F1-Score) were compared across all models to determine the best-performing one.

4. Model Selection:

- KNN was selected as the predictive maintenance model based on its superior performance across all metrics.

5. Building KNN Model:

- The KNN model was constructed using the chosen features and trained on the entire dataset.

6. Integration with Flask:

- The trained KNN model was saved using pickle.
- It was then loaded into a Flask application to facilitate real-time monitoring using test data.

7. Frontend Development:

- A form was developed using HTML and CSS to gather test data from users.
- Flask functionality was implemented to receive test data, forward it to the KNN model for prediction, and display the results.

8. Alert System:

- An alert system was implemented in the Flask application to indicate equipment failure probability based on model calculations.

9. Dashboard Development:

- A dashboard was created using a visualization library like Plotly or Dash.
- It showcased the percentage of equipment in various failure probability categories through a pie chart and listed equipment requiring immediate maintenance in a table, along with their health status.

10. Testing:

- The entire system was rigorously tested with various test cases to ensure its functionality, reliability, and accuracy.

IMPLEMENTATION

1. Data Collection and Preprocessing:

Dataset Link: <https://drive.google.com/file/d/1TD0LrsfRdGSDYJ2gjqP02fc8-zet0GRU/view?usp=sharing>

The dataset consists of 9 Features that represents sensor readings and 1 Class label that indicates Equipment Failure.

The Features include:

- Footfall
- Ambient Temperature
- SelfLR
- ClinLR
- DoleLR
- Proportional Integral Derivative(PID)
- OutPressure
- InPressure
- Equipment Temperature

(i) Import necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
import scipy
import sklearn
import tensorflow as tf
import torch
```

(ii) Load the dataset

```
# Load dataset from a CSV file
df = pd.read_csv('/content/iot_sensor_dataset.csv')
```

(iii) Preprocessing

Replacing null values with mean:

```
# Perform Preprocessing
df.fillna(df.mean(), inplace=True)
```

Standardise the values in footfall, outpressure and inpressure using Standard Scaler:

```
from sklearn.preprocessing import StandardScaler

# Scale numerical features using StandardScaler
scaler = StandardScaler()
df[['footfall', 'outpressure', 'inpressure', 'temp']] = scaler.fit_transform(df[['footfall', 'outpressure', 'inpressure', 'temp']])
```

After Preprocessing:

(Subset of Preprocessed dataset)

df

	footfall	atemp	selfLR	ClinLR	DoleLR	PID	outpressure	inpressure	temp	Unnamed: 9
0	-0.283153	7	7	1	6	6	-0.672788	-0.979504	-2.567407	1
1	-0.107558	1	3	3	5	1	-1.647540	-0.353894	-2.567407	0
2	-0.254504	7	2	2	6	1	-1.403852	0.897327	-2.567407	0
3	-0.206446	4	3	4	5	1	-1.160164	0.897327	-2.567407	0
4	0.308326	7	5	6	4	0	1.276716	0.897327	-2.567407	0

2. Feature Selection based on Correlation:

```
correlation_matrix = df.corr()
relevant_features = correlation_matrix[correlation_matrix > 0.5].index.tolist()
df = df[relevant_features]
print(relevant_features)
```

```
['footfall', 'atemp', 'selfLR', 'ClinLR', 'DoleLR', 'PID', 'outpressure', 'inpressure', 'temp', 'Unnamed: 9']
```

3. Train test Splitting:

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, :-1]
y = df.iloc[:, -1]
# Split the dataset into training and test sets (75% - 25% ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

4. Best Model Selection:

(i) Logistic Regression

```

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Train the Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
lr_accuracy = accuracy_score(y_test, y_pred)
lr_precision = precision_score(y_test, y_pred)
lr_recall = recall_score(y_test, y_pred)
lr_f1 = f1_score(y_test, y_pred)

# Print the metrics
print("Accuracy:", lr_accuracy)
print("Precision:", lr_precision)
print("Recall:", lr_recall)
print("F1-score:", lr_f1)

```

```

# Print the metrics
print("Accuracy:", lr_accuracy)
print("Precision:", lr_precision)
print("Recall:", lr_recall)
print("F1-score:", lr_f1)

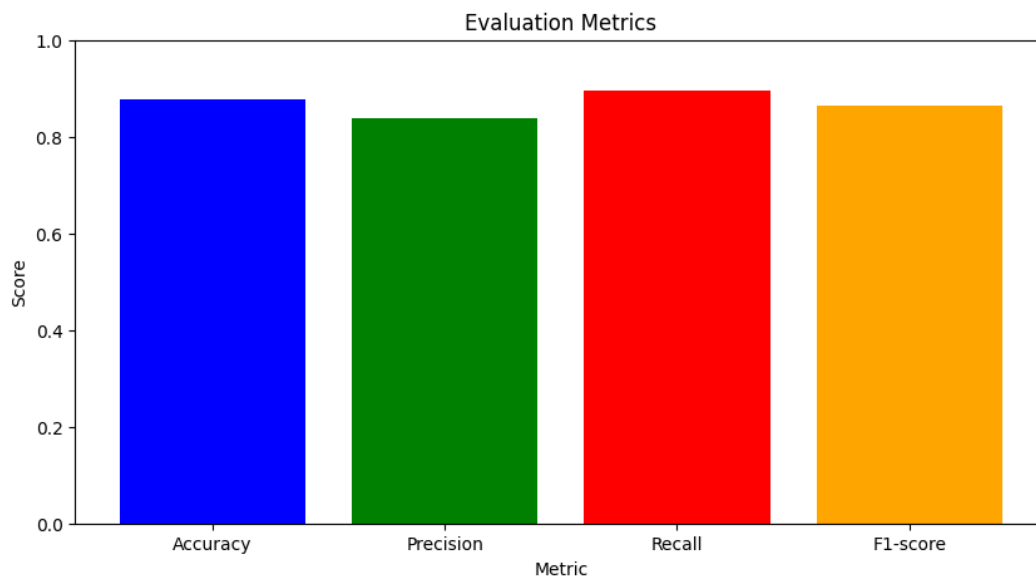
# Plot the evaluation metrics
labels = ['Accuracy', 'Precision', 'Recall', 'F1-score']
values = [lr_accuracy, lr_precision, lr_recall, lr_f1]

plt.figure(figsize=(10, 5))
plt.bar(labels, values, color=['blue', 'green', 'red', 'orange'])
plt.title('Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.ylim(0, 1) # Set y-axis limit to 0-1 for scores
plt.show()

```

Output

Accuracy: 0.8771186440677966
 Precision: 0.8378378378378378
 Recall: 0.8942307692307693
 F1-score: 0.8651162790697674



(ii) Decision Tree Classifier

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Create a Decision Tree model
model = DecisionTreeClassifier()

# Train the model using the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
dt_accuracy = accuracy_score(y_test, y_pred)
dt_precision = precision_score(y_test, y_pred)
dt_recall = recall_score(y_test, y_pred)
dt_f1 = f1_score(y_test, y_pred)

# Print the metric values
print("Accuracy:", dt_accuracy)
print("Precision:", dt_precision)
print("Recall:", dt_recall)
print("F1-score:", dt_f1)

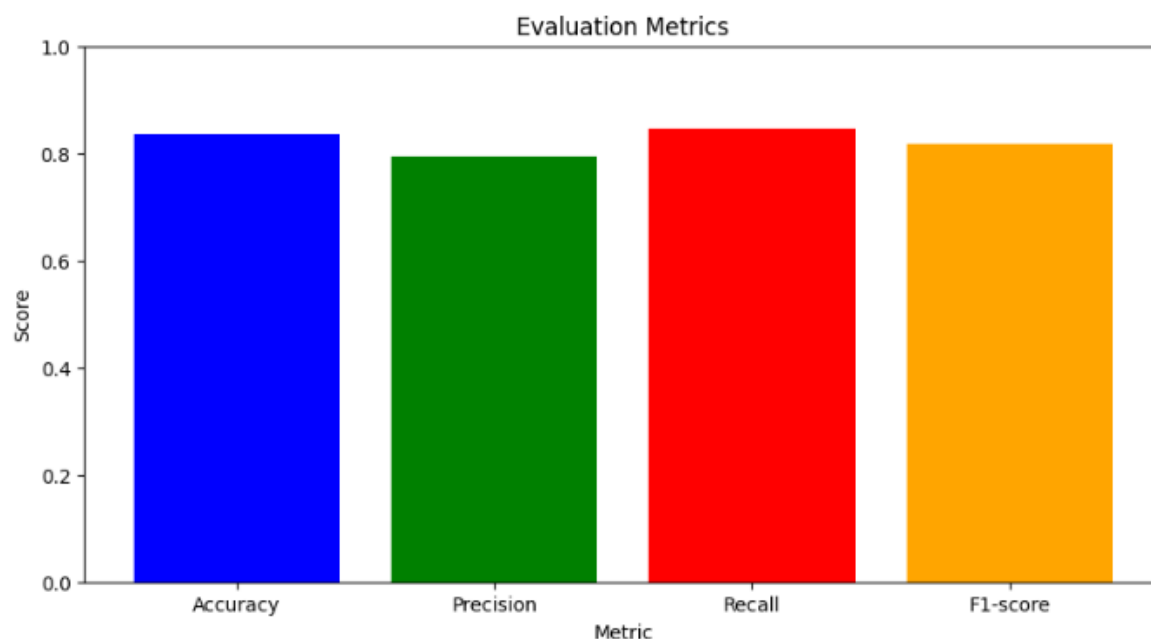
# Plot the evaluation metrics
labels = ['Accuracy', 'Precision', 'Recall', 'F1-score']
values = [dt_accuracy, dt_precision, dt_recall, dt_f1]

plt.figure(figsize=(10, 5))
plt.bar(labels, values, color=['blue', 'green', 'red', 'orange'])
plt.title('Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.ylim(0, 1) # Set y-axis limit to 0-1 for scores
plt.show()

```

Output

Accuracy: 0.8347457627118644
 Precision: 0.7927927927927928
 Recall: 0.8461538461538461
 F1-score: 0.8186046511627907



(iii) Random Forest Classifier

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Create a Random Forest model
model = RandomForestClassifier()

# Train the model using the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
rf_accuracy = accuracy_score(y_test, y_pred)
rf_precision = precision_score(y_test, y_pred)
rf_recall = recall_score(y_test, y_pred)
rf_f1 = f1_score(y_test, y_pred)

# Print the metric values
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1-score:", rf_f1)

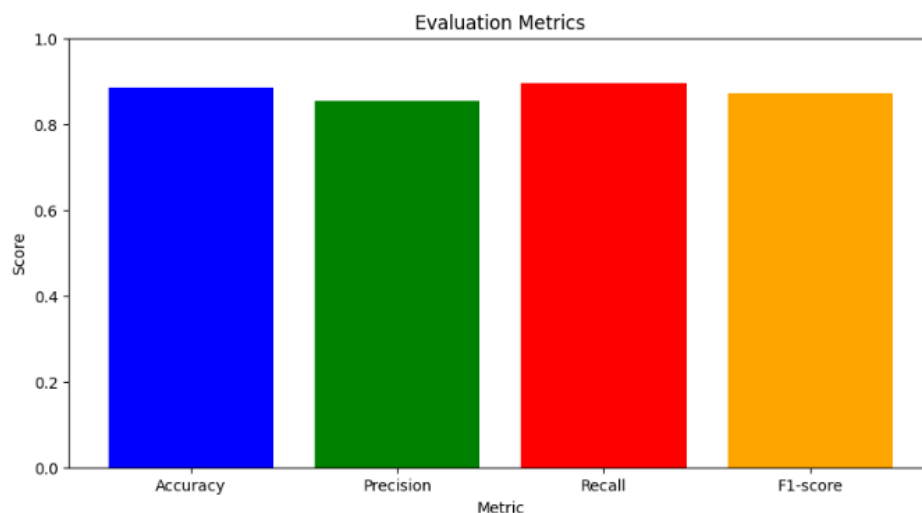
# Plot the evaluation metrics
labels = ['Accuracy', 'Precision', 'Recall', 'F1-score']
values = [rf_accuracy, rf_precision, rf_recall, rf_f1]

plt.figure(figsize=(10, 5))
plt.bar(labels, values, color=['blue', 'green', 'red', 'orange'])
plt.title('Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.ylim(0, 1) # Set y-axis limit to 0-1 for scores
plt.show()

```

Output

Accuracy: 0.885593220338983
 Precision: 0.8532110091743119
 Recall: 0.8942307692307693
 F1-score: 0.8732394366197184



(iv) K Nearest Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Create a KNN classifier model
knn_model = KNeighborsClassifier(n_neighbors=5) # Specify the number of neighbors (you can adjust this parameter)

# Train the model using the training data
knn_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
knn_accuracy = accuracy_score(y_test, y_pred)
knn_precision = precision_score(y_test, y_pred)
knn_recall = recall_score(y_test, y_pred)
knn_f1 = f1_score(y_test, y_pred)

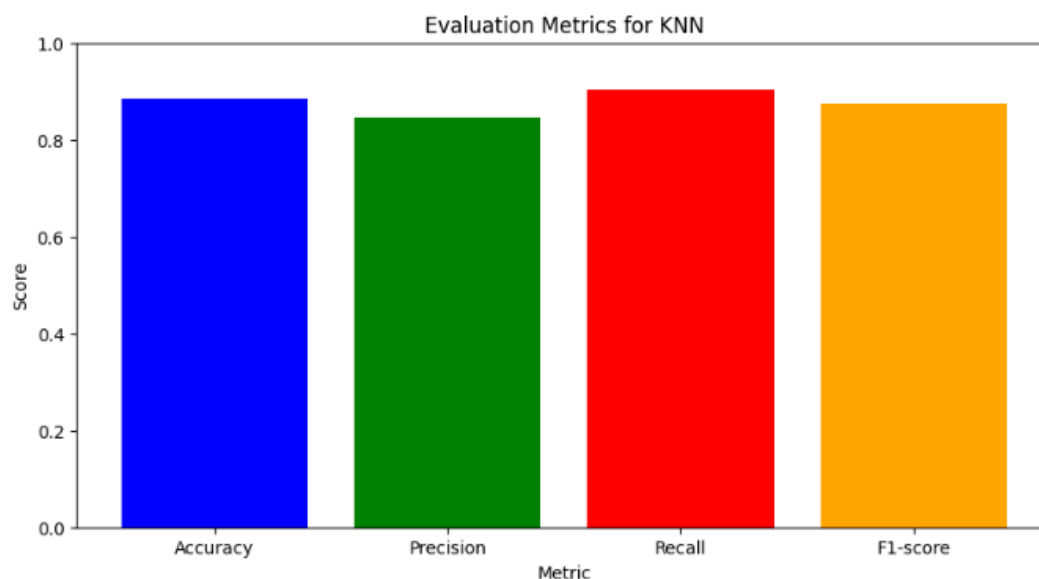
# Print the metric values
print("Accuracy:", knn_accuracy)
print("Precision:", knn_precision)
print("Recall:", knn_recall)
print("F1-score:", knn_f1)

# Plot the evaluation metrics
labels = ['Accuracy', 'Precision', 'Recall', 'F1-score']
values = [knn_accuracy, knn_precision, knn_recall, knn_f1]

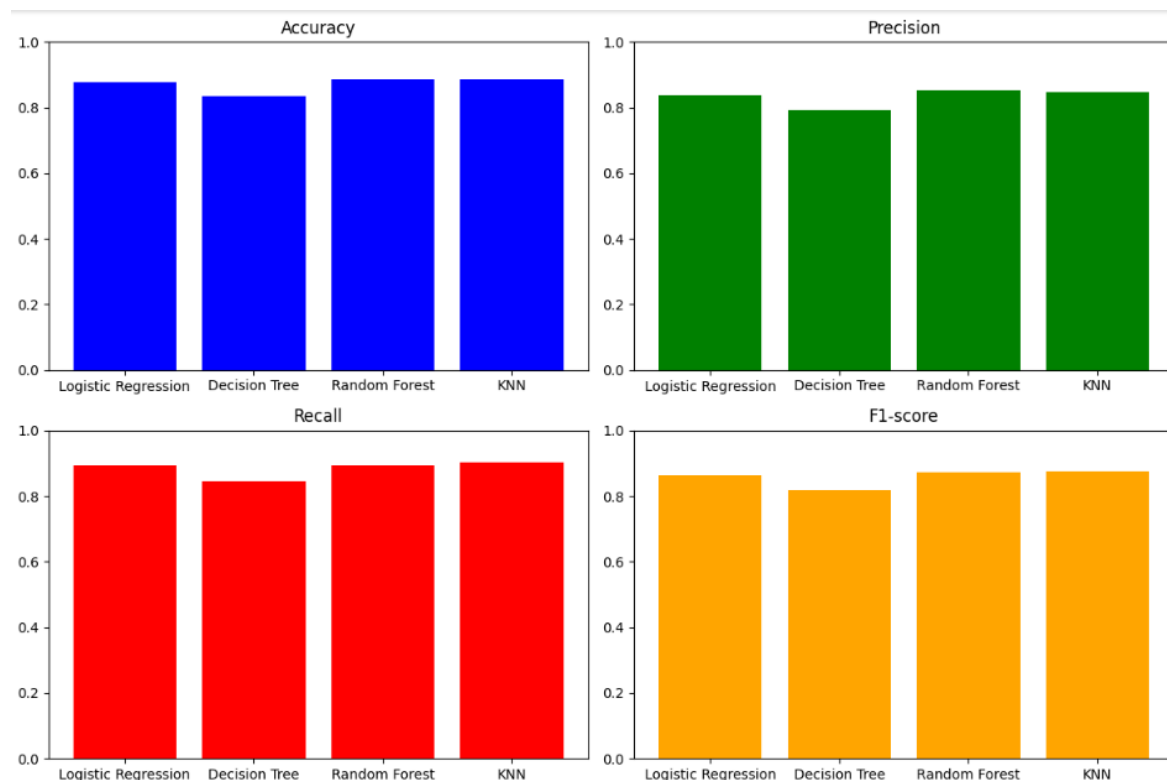
plt.figure(figsize=(10, 5))
plt.bar(labels, values, color=['blue', 'green', 'red', 'orange'])
plt.title('Evaluation Metrics for KNN')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.ylim(0, 1) # Set y-axis limit to 0-1 for scores
plt.show()
```

Output

Accuracy: 0.885593220338983
Precision: 0.8468468468468469
Recall: 0.9038461538461539
F1-score: 0.8744186046511627



Comparison:



(v) Stacking Classifier

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define the base models
base_models = [
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier()),
    ('rf', RandomForestClassifier()),
    ('knn', KNeighborsClassifier(n_neighbors=5)) # You can adjust the number of neighbors
]

# Define the meta-classifier
meta_classifier = LogisticRegression()

# Create the Stacking Classifier
stacking_clf = StackingClassifier(
    estimators=base_models,
    final_estimator=meta_classifier
)

# Train the Stacking Classifier using the training data
stacking_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = stacking_clf.predict(X_test)

# Calculate evaluation metrics for the stacking model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the metric values
print("Stacking Model Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
```

Output

```
Stacking Model Metrics:  
Accuracy: 0.8771186440677966  
Precision: 0.8440366972477065  
Recall: 0.8846153846153846  
F1-score: 0.863849765258216
```

(vi) Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score  
  
# Create the Gradient Boosting model  
gb_model = GradientBoostingClassifier()  
  
# Train the model using the training data  
gb_model.fit(X_train, y_train)  
  
# Make predictions on the test set  
y_pred = gb_model.predict(X_test)  
  
# Calculate evaluation metrics for the Gradient Boosting model  
accuracy = accuracy_score(y_test, y_pred)  
precision = precision_score(y_test, y_pred)  
recall = recall_score(y_test, y_pred)  
f1 = f1_score(y_test, y_pred)  
  
# Print the metric values  
print("Gradient Boosting Model Metrics:")  
print("Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1-score:", f1)
```

Output

```
Gradient Boosting Model Metrics:  
Accuracy: 0.8898305084745762  
Precision: 0.8679245283018868  
Recall: 0.8846153846153846  
F1-score: 0.8761904761904762
```

```
''' Out of all the Trained Models - KNN is chosen for my Classification task since it provides  
better evaluation metrics '''  
''' Now the KNN model is imported in Flask to make predictions on real time (new data) in VS Code '''
```


Now, KNN is chosen for building Predictive Maintenance Model and the trained model is loaded on Pickle for connecting with Flask for the purpose of :

- Real time monitoring using Test Data
- Alert System

➤ Maintaining Equipment Health

5. User Interface:

(i) Home Page

 **EquiPred**

Equipment Failure Prediction

Predict the Probability of Equipment Failure

FootFall Enter FootFall Value	Ambient Temperature Enter Ambient Temperature	Self LR Enter selfLR
ClinLR Enter ClinLR	DoleLR Enter DoleLR	PID Enter PID
OutPressure Enter outpressure	InPressure Enter inpressure	Equipment Temperature Enter Temperature

(ii) Prediction Page

Equipment Failure Prediction

Predict the Probability of Equipment Failure

FootFall 0	Ambient Temperature 7	Self LR 7
ClinLR 1	DoleLR 6	PID 6
OutPressure 36	InPressure 4	Equipment Temperature 1

PREDICT PROBABILITY

VIEW DASHBOARD

(iii)Alert System

127.0.0.1:5000/predict

☆

Equipment Failure Prediction

Predict the Probability of Equipment Failure

FootFall

Enter FootFall Value

Ambient Temperature

Enter Ambient Temperature

Self LR

Enter selfLR

ClinLR

Enter ClinLR

DoleLR

Enter DoleLR

PID

Enter PID

OutPressure

Enter outpressure

InPressure

Enter inpressure

Equipment Temperature

Enter Temperature

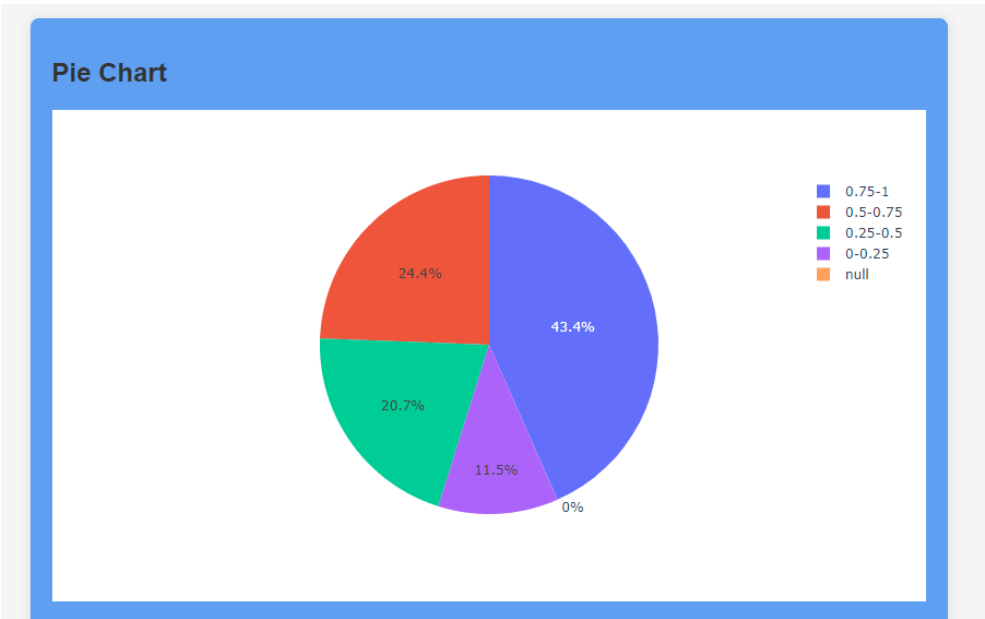
PREDICT PROBABILITY

VIEW DASHBOARD

Your Equipment is Prone to Failure. Probability of Equipment Failure is 1.00

(iv)Dashboard for displaying Predict Health

Piechart



Equipments that requires immediate Maintenance

Equipments that Require Immediate Maintenance

Probability	Equipment ID
1.0	0
1.0	23
0.8	51
0.8	56
0.8	58
0.8	67
1.0	69
0.8	84
0.8	114
0.8	132
0.8	140
0.8	171
1.0	173
0.8	181

5. Performance Metrics:

Accuracy

- Accuracy measures the overall correctness of the model.
- It is calculated as the ratio of correctly predicted instances to the total instances.

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Model Accuracy : 0.912

Precision

- Precision measures the proportion of true positive predictions out of all positive predictions made by the model.
- It is a measure of the model's exactness

$$precision = \frac{TP}{TP + FP}$$

Model Precision : 0.903

Recall

- Recall measures the proportion of actual positives that were correctly predicted by the model.
- It is a measure of the model's completeness.

$$recall = \frac{TP}{TP + FN}$$

Model Recall : 0.905

F1 Score

- F1 Score is the harmonic mean of Precision and Recall. It provides a balance between Precision and Recall.
- It is especially useful when you have imbalanced classes.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

Model F1 Score : 0.891

LINKS

Github Project Link: <https://github.com/SivaJegadeesh/EquiPred>

Submitted by
Siva Jegadeesh C B
2021503559