# Picture1

**College of computing**

**Department of software engineering**

**Machine Learning Project Documentation**

**Name : Eden Getachew**

**Id :1401182**

**ATM Maintenance Prediction System**

This documentation outlines the steps, technologies, and methods involved in building and deploying the ATM Maintenance Prediction system using machine learning and API deployment techniques. The system aims to predict the maintenance status of an ATM based on sensor data.

**Table of Contents**

**Contents Page No-**

1. [Overview](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "overview)…………………………………………………… 1

2. [Data Preprocessing](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "data-preprocessing)……………………………………. 1

3. [Model Training](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "model-training)………………………………………….. 2

4. [FastAPI Deployment](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "fastapi-deployment)…………………………………. 3

5. [Streamlit Interface](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "streamlit-interface)……………………………………… 4

6. [Deployment on Render](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "deployment-on-render)……………………………… 5

7. [API Interaction](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "api-interaction)…………………………………………… 5

8. [Troubleshooting](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "troubleshooting)………………………………………… 6

9. [Conclusion](https://chatgpt.com/share/67a6889c-52cc-8003-957d-26d47ee19098" \l "conclusion)…………………………………………………. 6

* **Overview**

The ATM Maintenance Prediction System is designed to predict whether an ATM will require maintenance based on various sensor data. The project leverages machine learning algorithms to build a predictive model, which is then exposed via a FastAPI backend for real-time predictions. Additionally, a Streamlit frontend was built for an interactive user interface.

* **Core Components:**
* **Machine Learning Model**: Trained using historical ATM data.
* **API Server**: Exposes the model via FastAPI for easy access.
* **Streamlit**: Provides a simple UI for users to interact with the model.
* **Deployment**: The model and API are deployed on Render, a cloud platform.
* **Data Preprocessing**

### **Dataset**

The dataset used for training the model consists of various sensor readings, such as:

* Air temperature [K]
* Process temperature [K]
* Rotational speed [rpm]
* Torque [Nm]
* Tool wear [min]
* Machine Type (Type\_M, Type\_L)

The data is preprocessed to handle missing values, normalize features, and perform one-hot encoding for categorical variables (such as Type\_M and Type\_L).

### Key Preprocessing Steps:

* **Handling Missing Data**: Fill or remove missing values.
* **Feature Scaling**: Normalize numerical features like temperature and torque.
* **One-Hot Encoding**: Convert categorical variables (Type\_M and Type\_L) into binary columns.
* **Model Training**
* **Model Choice:**

We used a **Random Forest Classifier** to predict ATM maintenance failures. Random Forest is chosen due to its high performance in handling large datasets and its ability to model complex relationships between features.

### **Training Process**:

* **Train-Test Split**: The dataset is split into training and testing sets.
* **Model Evaluation**: The model's performance is evaluated using metrics such as accuracy and confusion matrix.

**Final Model:**

The final trained model is saved using **joblib** for future use in the FastAPI backend.

import joblib

from sklearn.ensemble import RandomForestClassifier

from data\_preprocessing import load\_and\_preprocess\_data

X\_train, X\_test, y\_train, y\_test = load\_and\_preprocess\_data('../data/ai4i2020.csv')

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

joblib.dump(model, '../models/atm\_rf\_model.pkl')  # Save model

print("Model training complete. Model saved as atm\_rf\_model.pkl.")

* **FastAPI Deployment**

**FastAPI Setup**

The FastAPI backend serves the trained model and exposes an endpoint for predictions.

import joblib

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

import pandas as pd

app = FastAPI()

# Load the model

try:

    model = joblib.load('models/atm\_rf\_model.pkl')  # Ensure the path is correct relative to the location of api.py

    print("Model loaded successfully.")

    print("Model features:", model.feature\_names\_in\_)  # Inspect model features

except Exception as e:

    print(f"Error loading model: {e}")

    model = None  # Set model to None in case of an error

class InputData(BaseModel):

    Air\_temperature\_K: float

    Process\_temperature\_K: float

    Rotational\_speed\_rpm: float

    Torque\_Nm: float

    Tool\_wear\_min: float

    Type\_M: int

    Type\_L: int

@app.post("/predict")

def predict(data: InputData):

    if model is None:

        raise HTTPException(status\_code=500, detail="Model loading failed.")

    try:

        # Convert input data to DataFrame

        input\_data = pd.DataFrame([data.dict()])

        # Rename columns to match the feature names in the model

        input\_data.rename(columns={

            'Air\_temperature\_K': 'Air temperature [K]',

            'Process\_temperature\_K': 'Process temperature [K]',

            'Rotational\_speed\_rpm': 'Rotational speed [rpm]',

            'Torque\_Nm': 'Torque [Nm]',

            'Tool\_wear\_min': 'Tool wear [min]'

        }, inplace=True)

        # Reorder columns to match the order during model training

        expected\_order = model.feature\_names\_in\_  # Get the expected feature order

        input\_data = input\_data[expected\_order]  # Reorder input data based on model's expected order

        print(f"Input Data (renamed and reordered): {input\_data.head()}")  # Debugging input data

        # Make prediction using the loaded model

        prediction = model.predict(input\_data)

        print(f"Prediction: {prediction}")  # Debugging prediction result

        # Map the prediction to readable format

        prediction\_label = "Failure" if prediction[0] == 1 else "No Failure"

        return {"prediction": prediction\_label}

    except Exception as e:

        print(f"Error during prediction: {e}")

        raise HTTPException(status\_code=500, detail="Internal Server Error")

**Running the FastAPI Server:**

**To run the FastAPI server locally, use the following command:**

uvicorn api:app --reload

* **Streamlit Interface**

The Streamlit interface provides a user-friendly front end where users can input sensor data and receive predictions.

import streamlit as st

import requests

import pandas as pd

import json

# Define the API URL (Make sure to change this to the correct URL if deployed)

api\_url = "https://atm-maintenance-prediction-using-random.onrender.com/predict"  # Adjust as per your deployment

# Streamlit interface for inputs

st.title("ATM Maintenance Prediction")

# Input fields

air\_temperature = st.number\_input('Air temperature [K]', min\_value=0.0, value=300.0)

process\_temperature = st.number\_input('Process temperature [K]', min\_value=0.0, value=290.0)

rotational\_speed = st.number\_input('Rotational speed [rpm]', min\_value=0, value=3500)

torque = st.number\_input('Torque [Nm]', min\_value=0.0, value=55.0)

tool\_wear = st.number\_input('Tool wear [min]', min\_value=0.0, value=15.0)

# Type values (ensure they are integers)

type\_m = st.selectbox("Type\_M", options=[0, 1], index=1)

type\_l = st.selectbox("Type\_L", options=[0, 1], index=0)

# Collect the data as a dictionary

input\_data = {

    "Air\_temperature\_K": air\_temperature,

    "Process\_temperature\_K": process\_temperature,

    "Rotational\_speed\_rpm": rotational\_speed,

    "Torque\_Nm": torque,

    "Tool\_wear\_min": tool\_wear,

    "Type\_M": type\_m,

    "Type\_L": type\_l

}

# Convert the input data to JSON format

input\_json = json.dumps(input\_data)

# Make a POST request to the FastAPI prediction endpoint

if st.button('Predict'):

    try:

        response = requests.post(api\_url, data=input\_json, headers={'Content-Type': 'application/json'})

        if response.status\_code == 200:

            result = response.json()

            st.write(f"Prediction Result: {result['prediction']}")

        else:

            st.error(f"Error {response.status\_code}: {response.text}")

    except Exception as e:

        st.error(f"An error occurred: {e}")

* **Deployment on Render**
* **Deployment Steps:**

1. **Create a Render Account**: Sign up at [Render](https://render.com/).
2. **Create a Web Service**:
   * Choose "Web Service" and connect your GitHub repository or upload the project files.
   * Set the environment to **Python 3.x** and install necessary dependencies (**fastapi, uvicorn,** etc.).
3. **Set Up Environment**:
   * Add necessary environment variables, including the API URL if needed.
4. **Start the Service**: Deploy the FastAPI backend and check for successful deployment.

* **API Interaction**

The deployed API accepts data in the following format:

# Sample test data (representing a single instance for prediction)

sample\_test\_data = {

    'Air temperature [K]': [300.0],

    'Process temperature [K]': [290.0],

    'Rotational speed [rpm]': [3500],

    'Torque [Nm]': [55.0],

    'Tool wear [min]': [15.0],

    'Type\_L': [0],  # Assuming 0 for Type\_L (you may set accordingly based on your encoding)

    'Type\_M': [1]   # Assuming 1 for Type\_M

}

# Convert sample data to DataFrame for prediction

sample\_data\_df = pd.DataFrame(sample\_test\_data)

# Ensure the columns are in the same order as the training data

sample\_data\_df = sample\_data\_df[['Air temperature [K]', 'Process temperature [K]',

                                  'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]',

                                  'Type\_L', 'Type\_M']]  # Adjust the column order if necessary

# Check the structure of the sample data

print(sample\_data\_df)

# Make prediction

prediction = model.predict(sample\_data\_df)

# Show prediction

prediction\_label = "Failure" if prediction[0] == 1 else "No Failure"

print(f"Prediction: {prediction\_label}")

**Response Example:**

{

"**prediction": "Failure**"}

* **Troubleshooting**

1. **Missing Feature Names**: Ensure that the feature names in the input data match the names used when the model was trained.
2. **Model Not Found**: Ensure that the model file (atm\_rf\_model.pkl) is available in the correct directory.
3. **API Errors**: Check the FastAPI logs for more detailed error messages when encountering issues.

* **Conclusion**

The ATM Maintenance Prediction system allows real-time maintenance predictions based on sensor data using a trained machine learning model. The system is accessible via an API and has a simple, interactive frontend built with Streamlit. The model is deployed on Render for scalability and ease of access.

This document should serve as a comprehensive guide to understand the components of the project, how they interact, and how to deploy and interact with the system. Feel free to expand upon or adjust the deployment and code setup as per your specific requirements.