# Low Level Documentation

***Document Name:*** **Low Level Documentation for AnomaData application project**

***Author:*** **Saikat Talukdar**

| Version | Date | Summary of changes | Reference ID |
|---|---|---|---|
| 0.0 | 5/7/2024 | Created | |
| | | | |
| | | | |
| | | | |

## Application Overview

### AnomaData (Automated Anomaly Detection for Predictive Maintenance)

**Problem Statement:**

Many different industries need predictive maintenance solutions to reduce risks and gain actionable insights through processing data from their equipment.

Although system failure is a very general issue that can occur in any machine, predicting the failure and taking steps to prevent such failure is most important for any machine or software application.

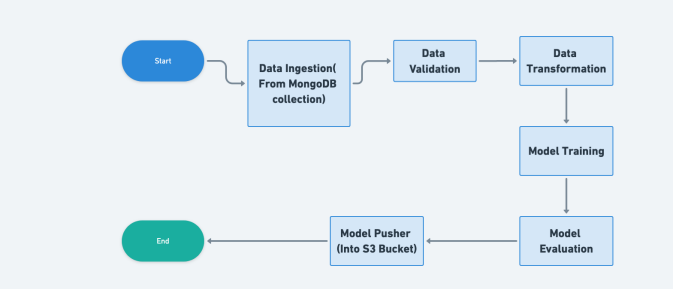
Predictive maintenance evaluates the condition of equipment by performing online monitoring. The goal is to perform maintenance before the equipment degrades or breaks down.

This project is aimed at predicting the machine breakdown by identifying the anomalies in the data.

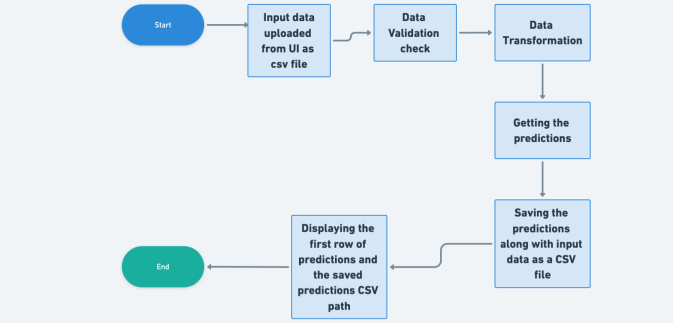
The data we have contains about 18000+ rows collected over few days. The column 'y' contains the binary labels, with 1 denoting there is an anomaly. The rest of the columns are predictors.

# Low Level Documentation

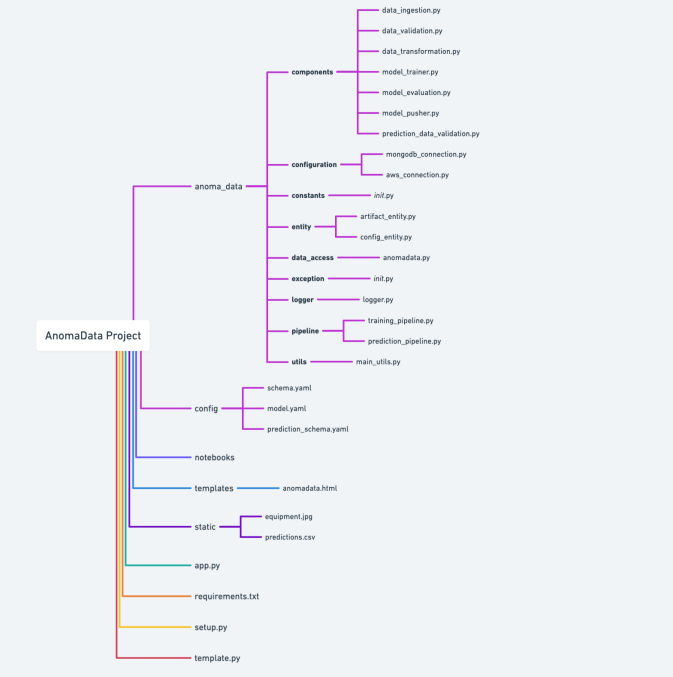
## Architecture




Stages of training pipeline




Stages of prediction pipeline




Complete Project structure

## Detailed Overview

As we can see from the above architecture diagram, we have two pipelines for the implementation of the project:

1. Training Pipeline: This is where the training data is fed to the model to train it in order to accurately predict on the test data.
2. Prediction Pipeline: This is where the input test data as a CSV file is fed to the model from the application UI to get predictions.

We can also see the detailed folder and file structure from above.

# Training Pipeline:

Our training pipeline has these stages:

1. Data Ingestion stage
2. Data Validation Stage
3. Data Transformation Stage
4. Model Training Stage
5. Model Evaluation Stage
6. Model Pusher Stage

## Data Ingestion

*In this stage, the data which we have stored in the MongoDB collection is fetched. The fetched data is stored as a .csv file inside the artifact folder in the 'feature_store' folder. This data is also split into train.csv and test.csv and saved inside the artifact folder in the 'ingested' folder.*

*Everytime we train the model, a new artifact folder is created with the timestamp.*

## Data Validation

*In this stage, we perform different sets of validation on the ingested files train.csv and test.csv.*

1. *Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, schema.yaml"*
2. *If all the columns are present - All the columns in the train.csv and test.csv should be the same as that given in the schema file.*

*If validation status from this stage is True, then the data is sent to the data transformation stage.*

## Data Transformation

*1) Both train data and test data is divided into input feature and target feature data frame by dropping the target column and using it respectively.*

*2) Unnecessary columns are dropped as per the 'drop_cols' column list in the schema file.*

*3) Power Transform (yeo-johnson method) applied to the skewed columns and standard scaler applied to the feature columns.*

*4) SMOTEEN applied to address dataset imbalance.*

## Model Training

*1) Two algorithms used: KNN Classifier and Random Forest.*

*2) The corresponding hyper parameters are stored in model.yaml*

*3) Using neuro-mf package, the best model giving the best scores with the given parameters are chosen and the model is stored as in the artifact folder as model.pkl*

## Model Evaluation

*The score of the model present in the S3 bucket is matched against the model trained currently. If the change in score is above the threshold mentioned in schema file, i.e, if the current model has better score than the model in S3 bucket and the difference is above the threshold, then evaluation status is set to True. The current model will be now pushed to the S3 bucket in the next stage.*

*If there is no model in the S3, then the current model will be pushed into it, provided it is above the trained model threshold score.*

## Model Pusher

*Depending on the evaluation status, the model is pushed into the S3 bucket.*

*With this, our Training pipeline is complete.*

## Prediction Pipeline

*1) Input data CSV file is chosen by the user from the UI.*

*2) This data is validated to check if the number of columns are same as the prediction_schema.yaml file. Also checked if all the required columns present. If validation status True, then it passes to the next step.*

*3) Irrelevant columns are dropped as per the 'drop_cols' list in the prediction schema.*

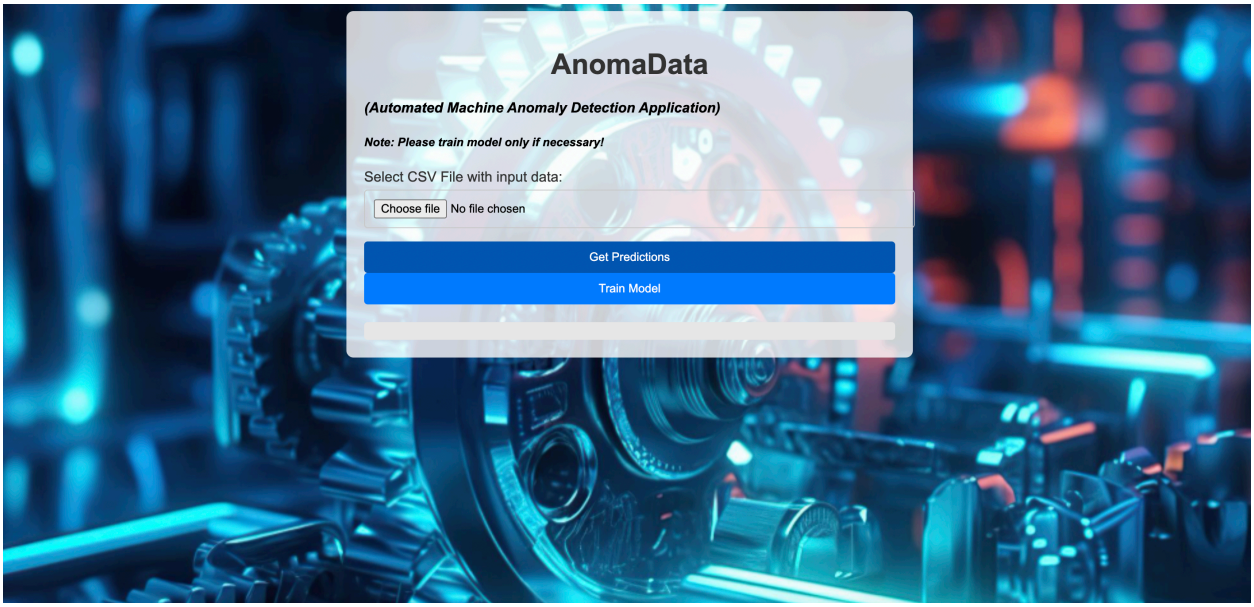
*4) The processed data is fed to the model in the S3 bucket to get the predictions.*

*5) The predictions along with the input data is stored in predictions.csv file. Also the first row of prediction is displayed to the user.*

*With this, our Prediction pipeline is complete and our application is ready to be deployed.*

## Application User Interface

*This is a FastAPI application. The UI looks like this:*

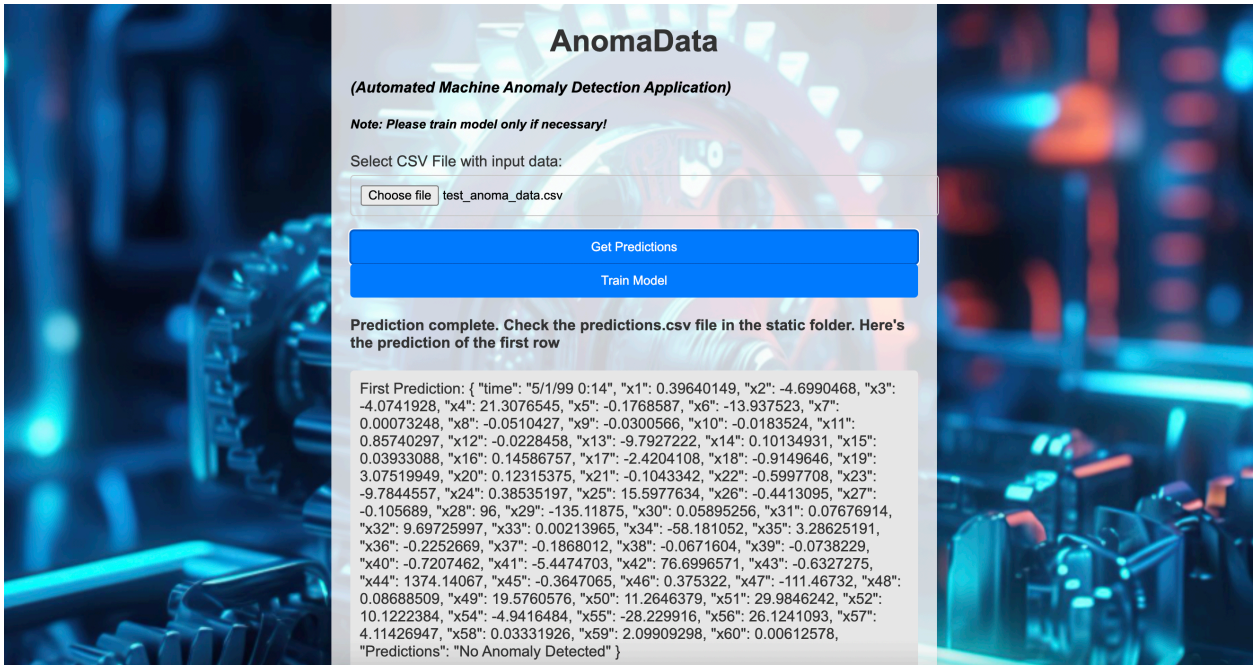



*There are these three buttons on the UI:*

*Choose File: This button is there for the user to upload the input CSV file.*

*Get Predictions: This button is to get the predictions on the input data*

*Train Model: This button is to train the model again.*




*This is how the UI looks like once we get the predictions using Get Predictions button.*

## Deployment using CI/CD pipeline

*This application was test-deployed using GitHub Actions in EC2 machine in AWS.*

## Future improvement scope

1. Handling other file types besides csv for test data.

2. Training data can be divided into clusters and then different models can be used for different clusters of data instead of a single model.

## Files descriptions

***requirements.txt*** *file consists of all the packages that you need to deploy the app in the cloud.*

***app.py*** *is the entry point of our application.*

***setup.py*** *file to set up the packages.*

***template.py*** *file to set up the folder structure.*

***mongodb_connection.py*** *file to set up the MongoDB connection.*

***aws_connection.py*** *file to set up connection with AWS.*

***init.py*** *file in* ***constants*** *to store the values of constants.*

***artifact_entity.py*** *file to store the definitions of artifact entity classes.*

***config_entity.py*** *file to store file path of configuration entity classes.*

***init.py*** *(****exception****) file to handle exceptions.*

***logger.py*** *file to log the execution details.*

***schema.yaml*** *file to store the valid data schema information for training data.*

***model.yaml*** *file to store the model types and hyper parameters for passing to ModelFactory in neuro-mf package to determine best model.*

***prediction_schema.yaml*** *file to store the valid data schema information for prediction data.*

***data_ingestion.py, data_validation.py, data_transformation.py, model_trainer.py, model_evaluation.py, model_pusher.py*** *files for training data pipeline steps.*

***prediction_data_validation.py*** *file for prediction data validation.*

***training_pipeline.py*** *file for running the training pipeline.*

***prediction_pipeline.py*** *file for running the prediction pipeline.*