**Anomaly detection in an IoT setting (spotlight: Stream processing)**

**Problem Statement**

A cutting-edge modern factory is producing machine components for wind turbines. The factory is equipped with numerous sensors which track the production cycle. The factory managers can oversee the numeric values in a BI dashboard. In addition to this, they would like to be informed by a decision support system when patterns emerge in the sensor data, implying that something has gone wrong. As the data scientist for this project, you're assigned to develop and implement an anomaly detection system that integrates with the productive system. You can rely on the expertise of the shop floor employees who have been working at the production site for a long time and have gathered valuable domain knowledge. In close consultancy with the shop floor employees, it has become clear that temperature, humidity, and sound volume are good indicators for an anomaly in the production cycle and a faulty produced item. You build a simple machine learning model in Python which uses these features to predict an anomaly score for each made item. This model should not be very elaborate, but the focus of your task is to design a model so that it is easily implemented in the productive system. Your system must be able to process data streams, as the sensors in the factory measure data continuously. Your implemented model should take these measurements over a standardized API and respond with a prediction score for an anomaly. Your system must be easily monitorable, maintainable, scalable, and adaptable to new data.

1. Design the conceptual architecture of your system. By doing so, consider data ingestion, processing, and handling requests to the prediction model as a service. Draft a visual overview of your architectural design, showing which data and processes are handed over by which application to the next. This will also guide you through the next steps of your project.

2. Choose an open data source that can serve as sample data for your project. A good starting point for your research are open data science competition websites, such as Kaggle. Alternatively, you can also produce your own fictional sample data. As you have to simulate continuous streams of data, you basically have three options: 1) Run an application on your local machine which produces a continuous stream of simulated sensor data, 2) find a continuous open stream of sample data online, or 3) use a static opendata set and simulate its continuous nature by manually updating the data and monitoring the changes in your system

3. Build a simple anomaly detection model using Python. The model should be trained by the simulated sensor data you acquired in the second step. Do not put too much effort into building this model, and just make sure that it performs the task at hand and check basic statistical measures.

4. Package your model in a way that it can take (simulated) sensor data over a standardized RESTful API and respond with a prediction for an anomaly. You might want to use Python libraries for this, such as Flask or mlflow. A good starting point for this step is the documentation of the respective Python libraries. This will be the most work-intensive step of your project.

5. It is unnecessary to implement your system in the cloud to obtain the highest grade for your project.

Implementing to the cloud is a little more elaborate, though you might want to challenge yourself and gain some extra points for this effort. For your final oral project report, also consider these questions from the module description:

• What were the challenges of integrating a predictive model into an application or service?

• What are the constraints of implementing a predictive model as a service?

• Which requirements for data acquisition, storage, and processing had to be met, and how did you achieve this?

• What are monitoring components required for reliable execution of the predictive model?

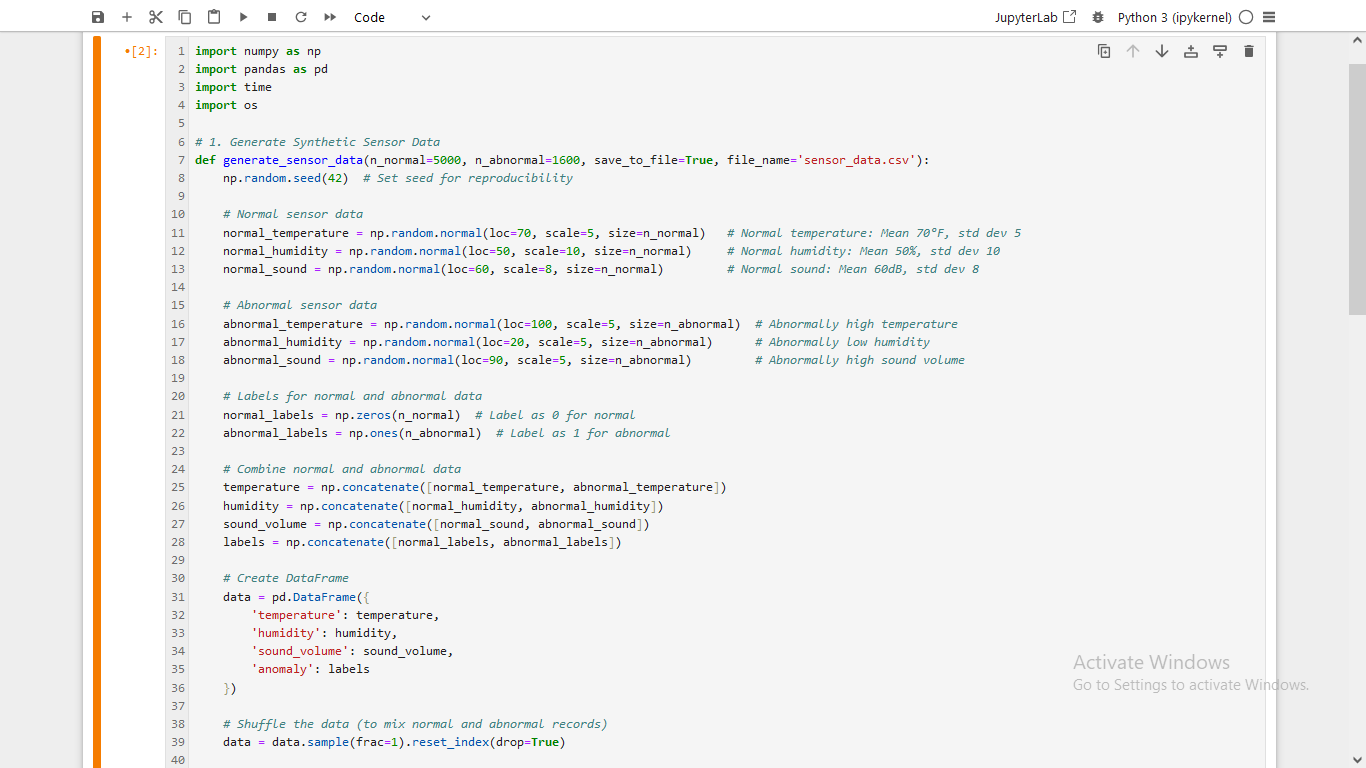
• What is the design of your system? Present a visual draft of your system regarding storing, accessing, and serving the predictive model.

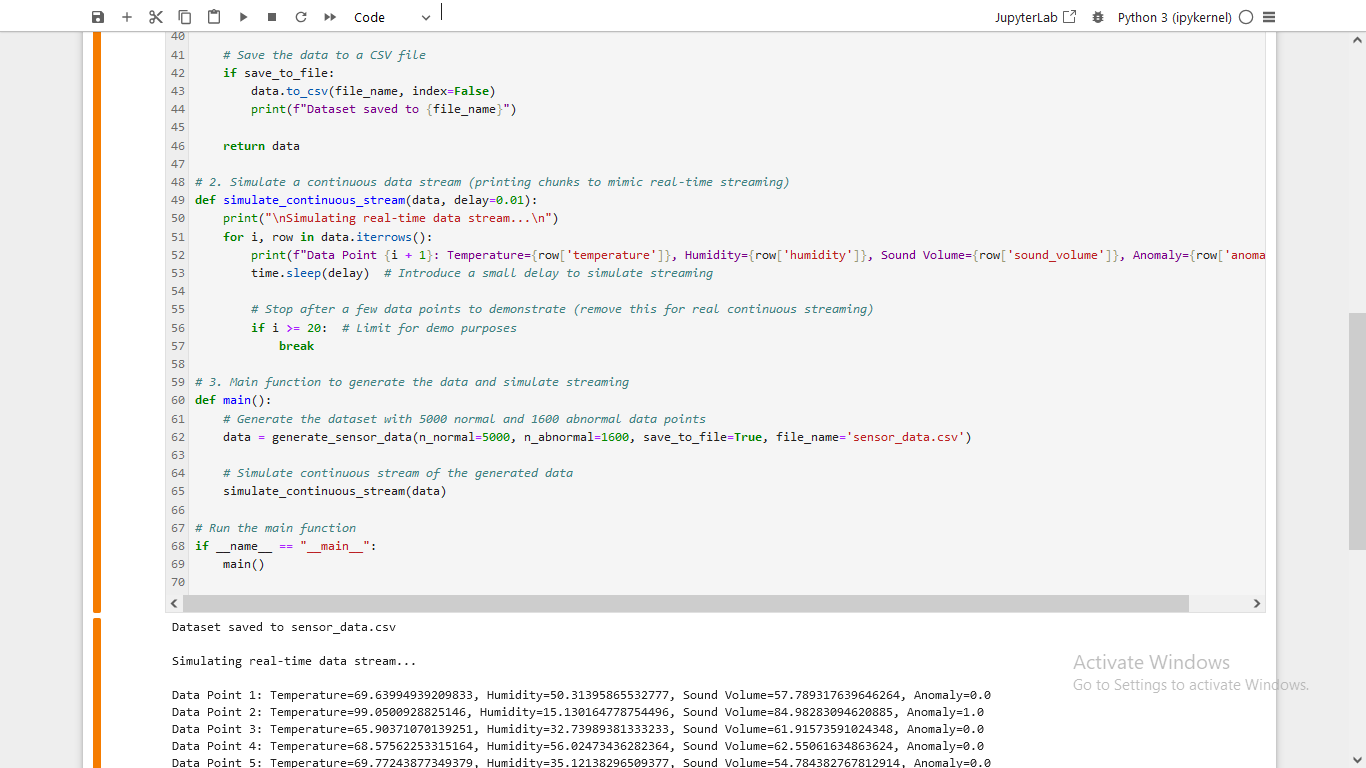
• In your oral presentation, provide a link for your audience to follow and reproduce your code

IMPLEMENTATION

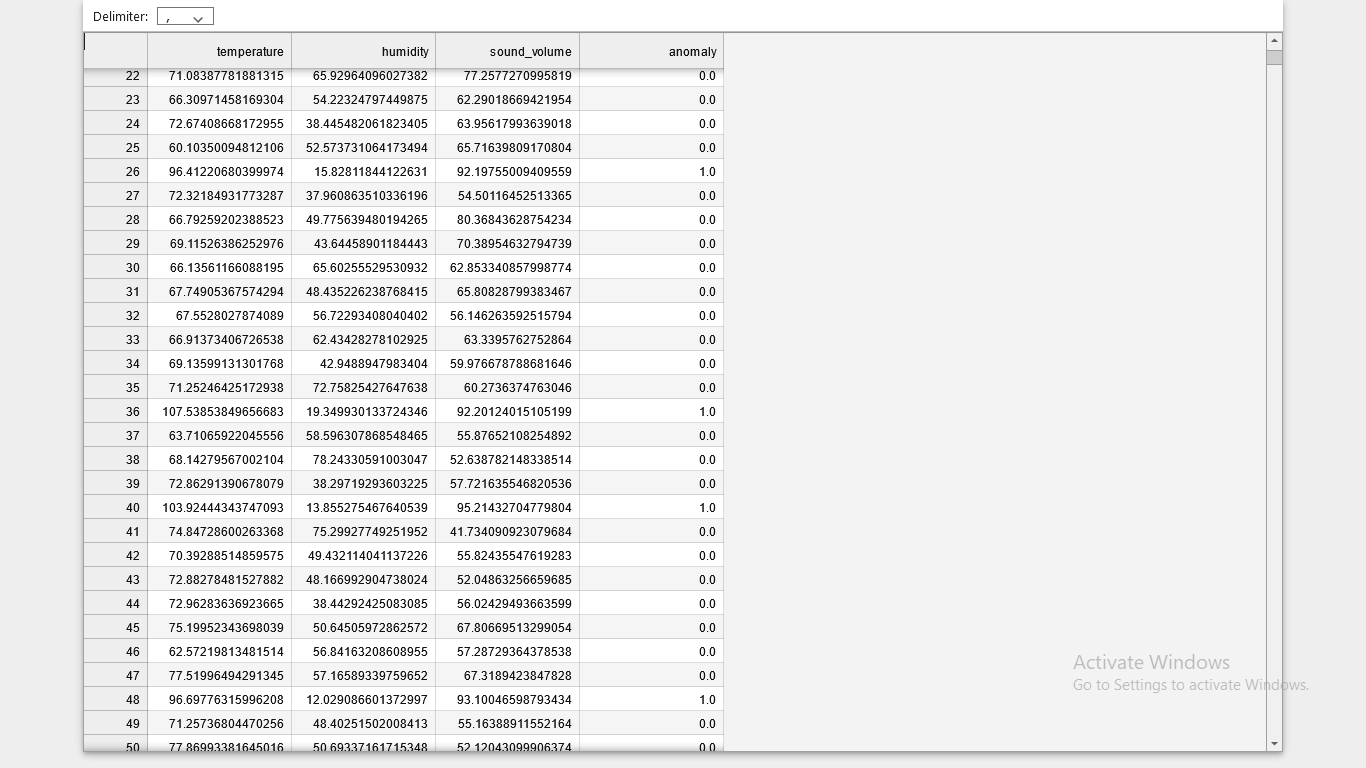
I started off the implementation by simulating real-time data stream which has as its features Temperature, Humidity and Sound volume. The stream of data is then organized and save in csv file that will form the dataset that will be use to train the model to detect anomaly .

During the generation of the synthetic sensor data, I specified that we should have 5000 normal data points or instances or rows and 1600 anomaly data points or instances or rows. After the imports of some essential python packages like numpy,pandas,time and os ; we were ready to synthesize the dataset using some properties that the real dataset has and that is why we use the Gaussian Distribution or Normal Distribution.





**Generated Real-Tme Data**



After we generate the real-time data and saved it to the csv file, we are now ready to develop the model that we will feed in the synthesized dataset, I mean we want to train the model with data we extract from the csv file. Since we generated the dataset using Gaussian distribution features , we will not be needing much of the process associated with cleaning the data like taking care of null values, missing values etc but we will scale the values to put the numbers on the same pedestrian or scale to avoid tilting the machine learning computation to an odd direction. So we start by importing all the necessary packages to help us do the different aspects of the machine learning task. We bring in pandas, numpy, train\_test\_split from sklearn.model\_selection , StandardScaler from sklearn.preprocessing, LogisticRegression from sklearn.linear\_model, and from sklearn.metrics we imported classification\_report, accuracy\_score, confusion\_matrix and finally we brought in joblib.

Let me state in brief what the various imports are doing in the whole process:

We use the pandas to load the dataset from the csv file and organize the data in such a way that the model can accept the data for further processing.

We use the train\_test\_split package to split the dataset into training and testing portions. We use 30% of the dataset for testing and 70% for training, to ensure reproducibility of the results we got we use a random\_state value of 42.

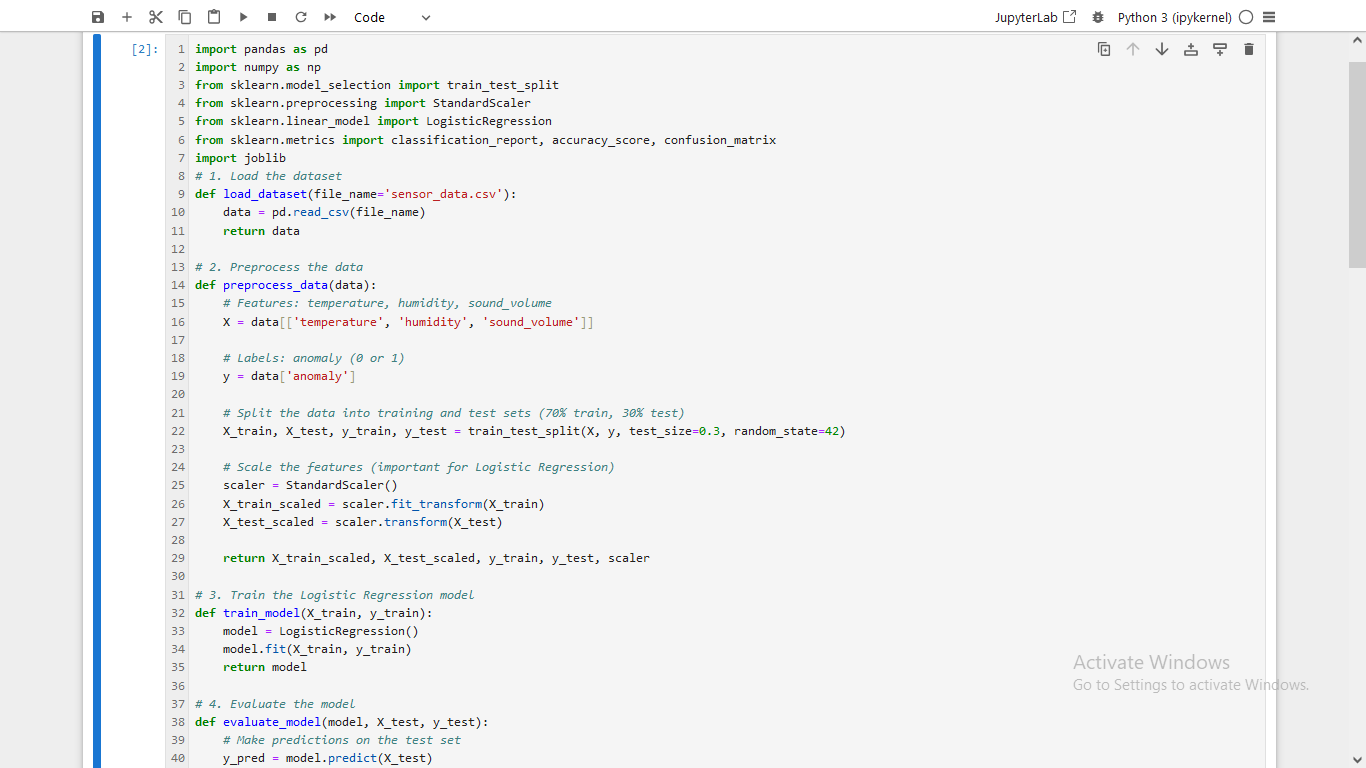
We use the StandardScaler to scale all our numeric values so it will all be uniform and be on the same scale.

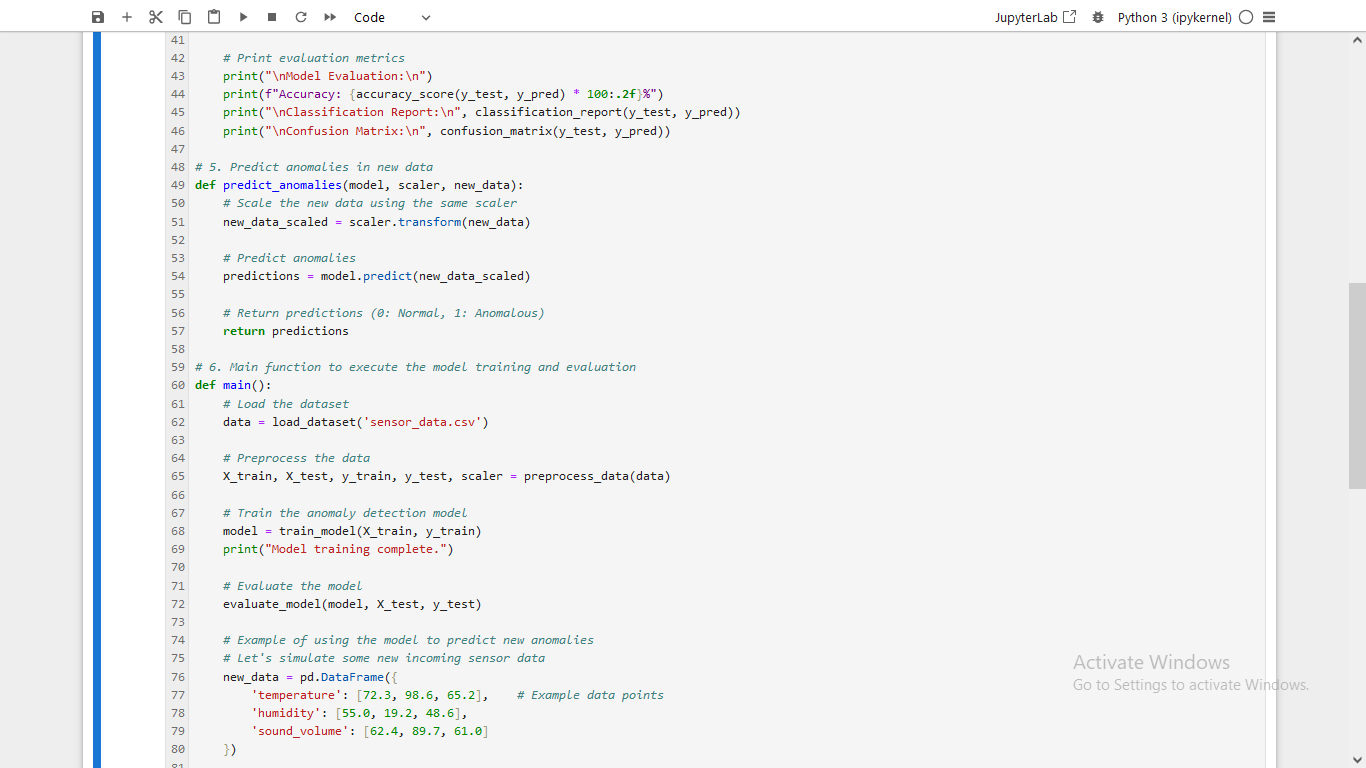
We use the LogisticRegression has the machine learning model that does the training of the model , the is the package that allows the model to learn useful patterns from the dataset.

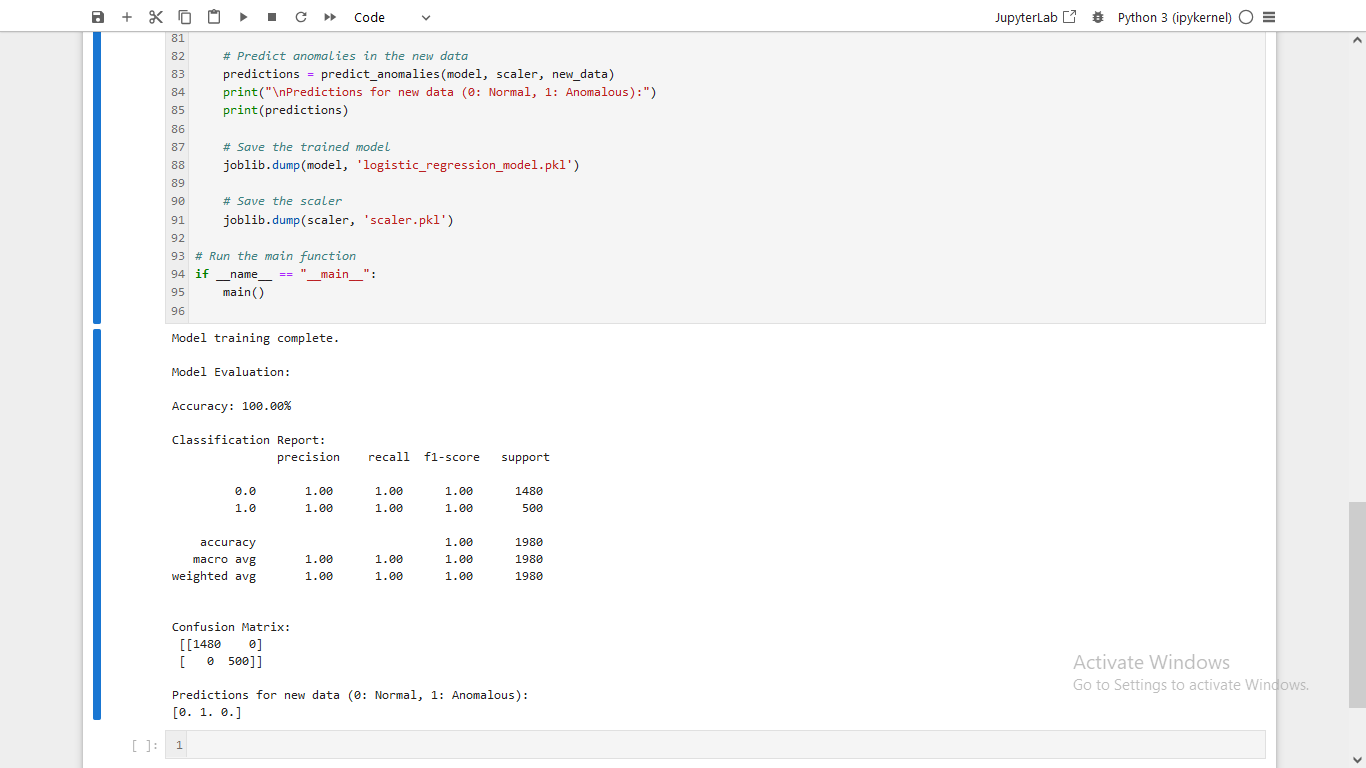
All the packages from the sklearn.metrics module like Classification\_report ,accuracy\_score, confusion\_matrix gives us a good idea how the model is doing i.e. how good the model has learnt the inherent patterns in the dataset and how prepared it is to predict outcomes for unseen data stream.

Finally when we are done with the model, we need to package the model and its scaler in such a way that it can be deployed to other working environments where its services can be use in various ways i.e. like predicting anamoly for a new stream of data via a RESTful API using http request, this is where the joblib comes in. The joblib helps us to package our anomaly detection model and its associated scaler so that we can deploy them easily to the server of our choice.

The model implementation code is given below and its associated metric reports:





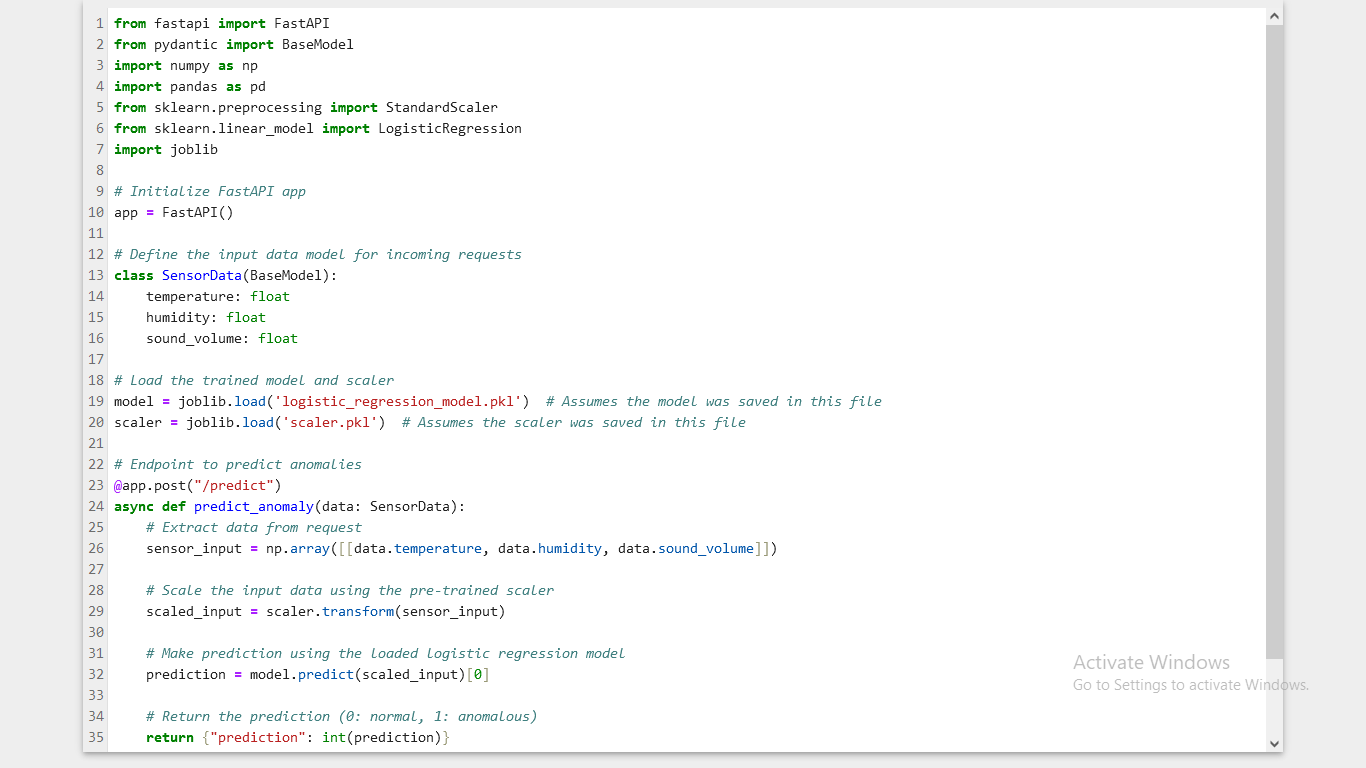


After we are done with the model is time to develop the back end that we will deploy the packaged model to, and this backend will receive HTTP request, pass it on to the model, the model will scale the input , then it will check for pattern the input corresponds with then we will get a feedback.

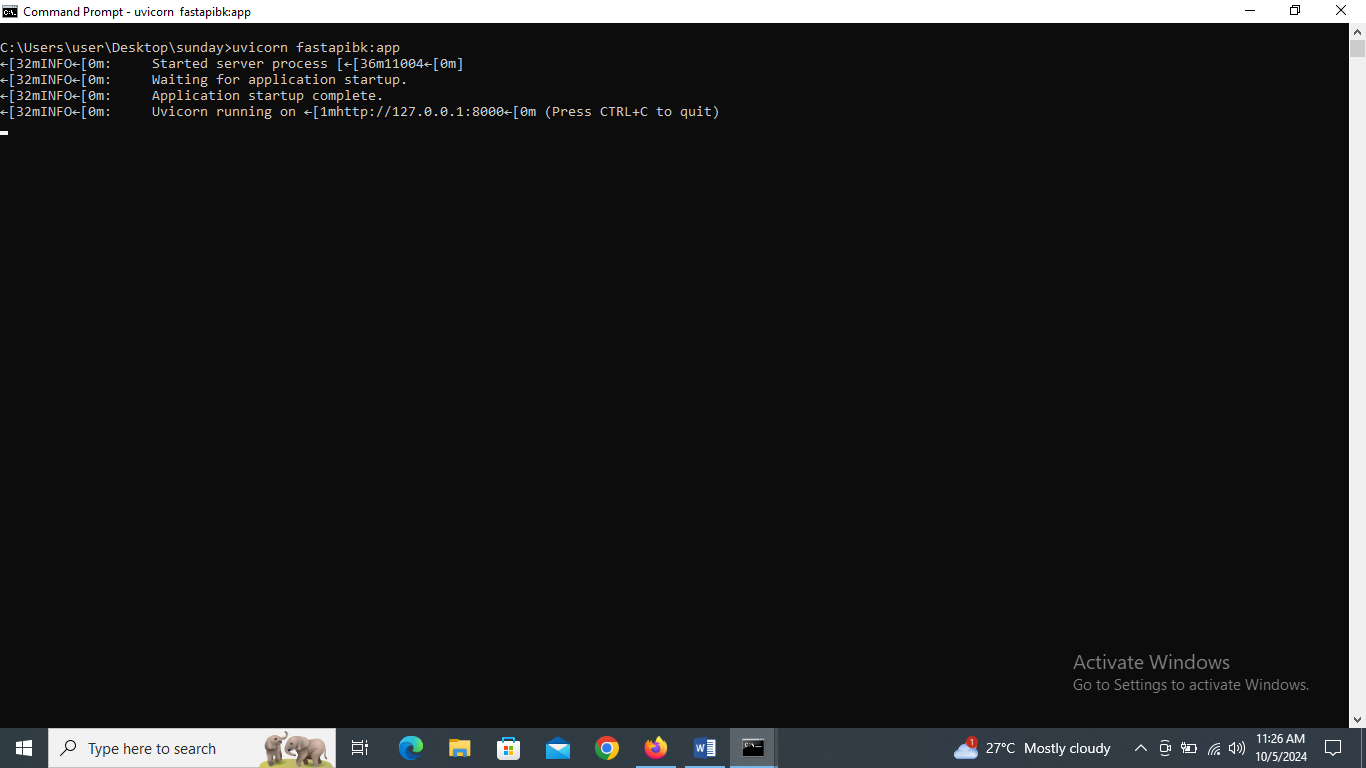
For easy communication with clients, we resolve to use an Application Programming Interface (API) methodology , that enables client a client to use a simple interface to request for data and the server or backend to respond to the client via simple formats like json, xml ,csv etc.

To implement API we use a python package called FastAPI that enable us create the different endpoints we will be needing for seamless communication between client and server.This implementation will load the model and scaler via the joblib and we can interact with it. For this project we just created a one endpoint called ‘/predict’, from which client will request anomaly report by passing in data that encompasses temperature, humidity and sound volume, the server will give a feedback base on the data it receives.

We further deploy this endpoint app to an asynchronous server gate interface (ASGI) called uvicorn to help in receiving and responding to request from the client . As at this point our backend can respond to a client using HTTP RESTful API mechanism.

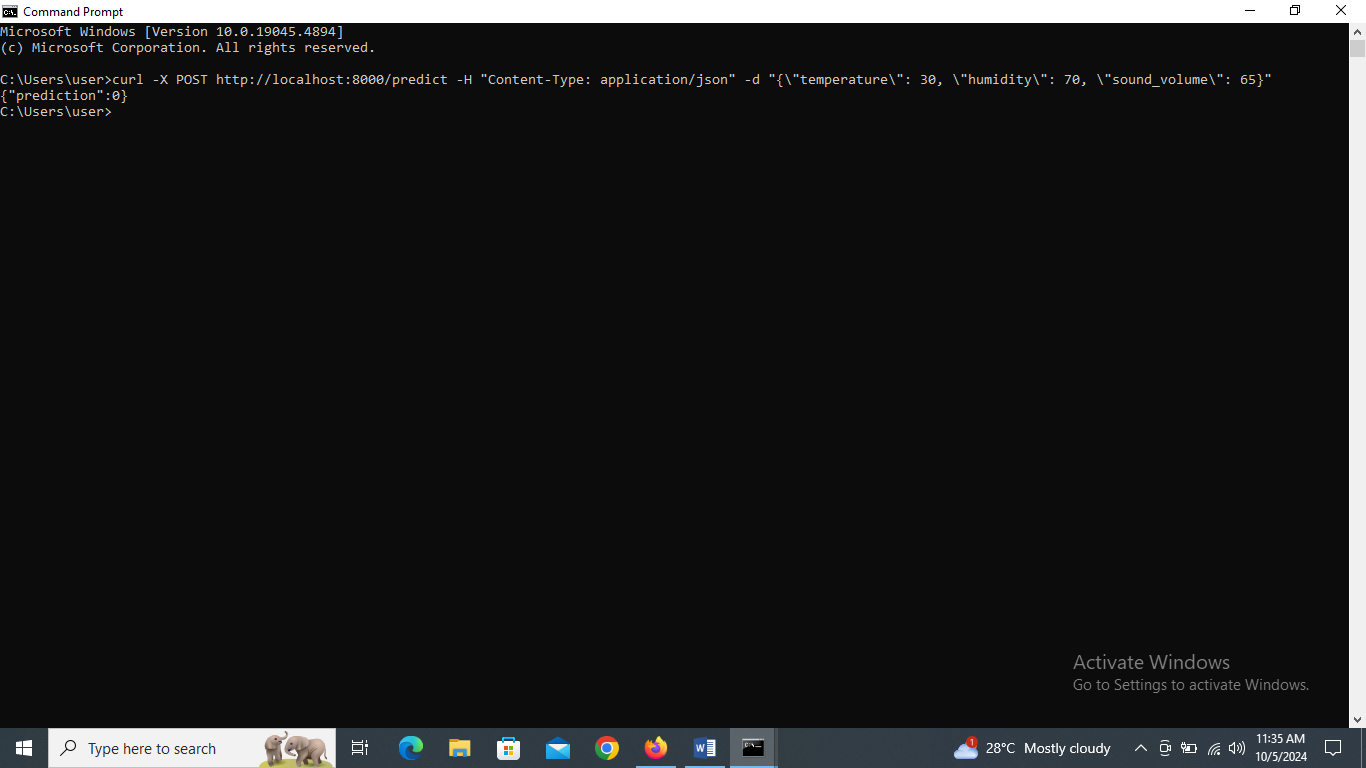


So we can start the backend by typing uvicorn fastapi:app then enter in the command prompt to wait for request from the client , the server has started and waiting for interaction

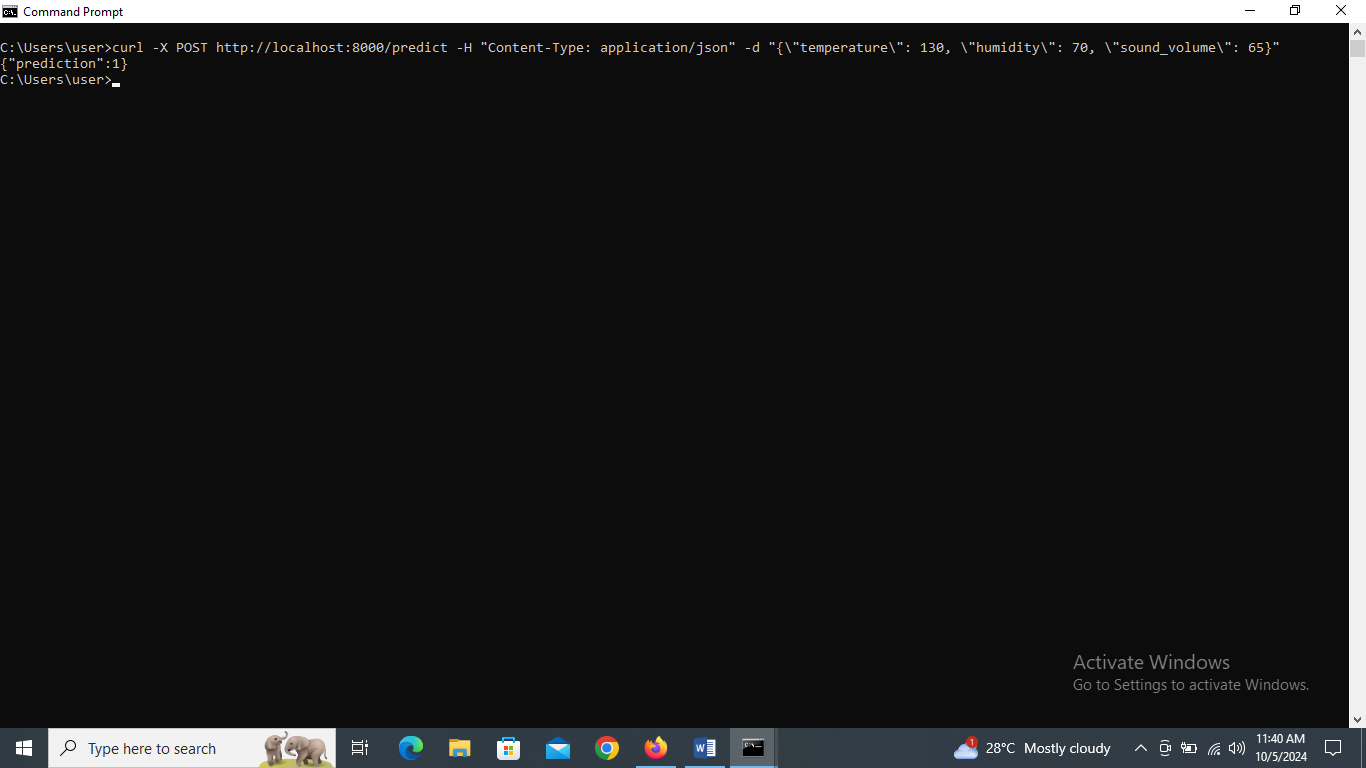


So interact with such a service we need an http request, which we can do by either building a frontend user interface, or we can use postman or we can use the popular command line tool called curl. Any of our choice is fine, but here as an illustration I will use curl and pass in temperature equals 30, humidity equals 70 and sound volume equals 65, so we can see the response.

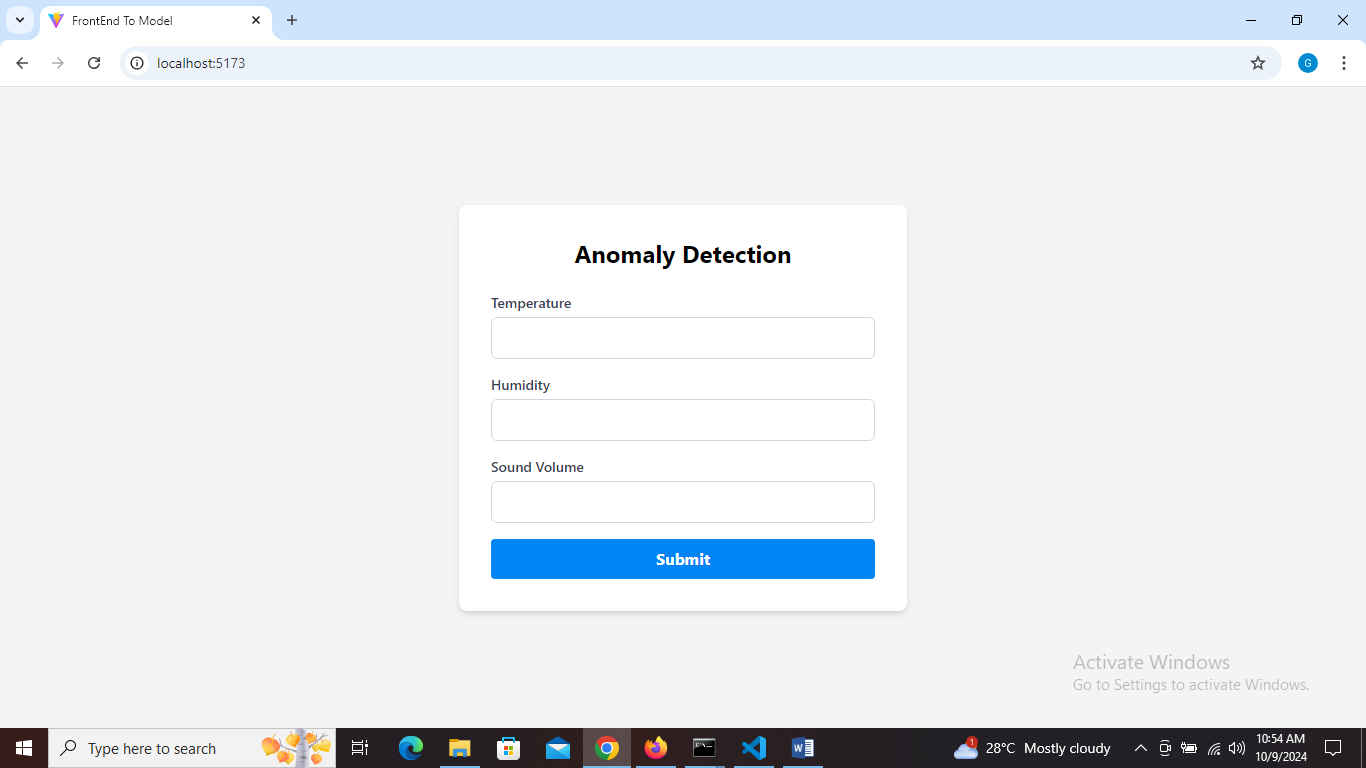
So open a new command line window and paste the following --🡪 curl -X POST http://localhost:8000/predict -H "Content-Type: application/json" -d "{\"temperature\": 30, \"humidity\": 70, \"sound\_volume\": 65}"



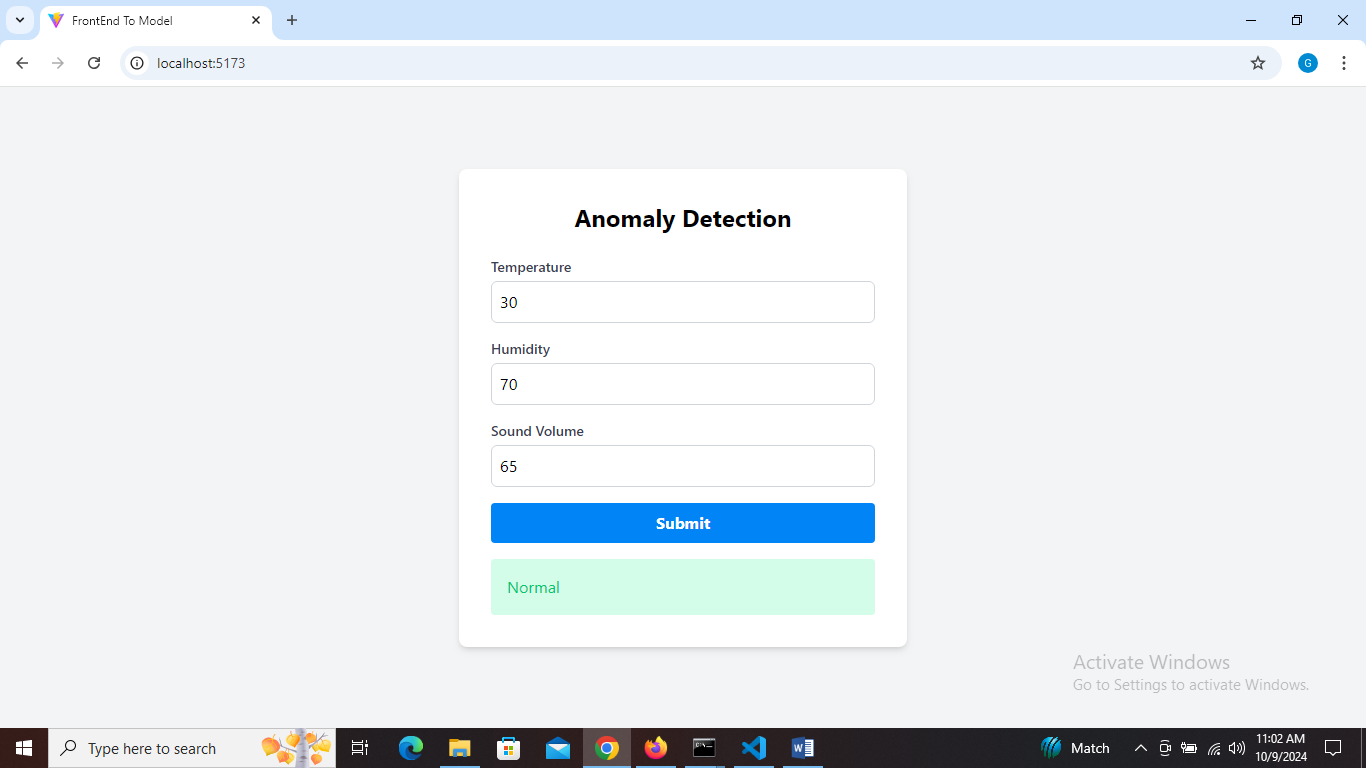
Notice I get a json response {“prediction”:0} , which mean is normal . Let me now use temperature equals 130. You will notice is an anomaly {“prediction”:1}



So let’s see the usage of the service using a frontend that can help us easily pass values of temperature, humidity and sound volume. We use Vite to set up the frontend project, then we use the javascript framework call ReactJs and a css framework called TailWindCSS to build and style the simple user interface that will act as a client to our backend.



Entering temperature equals 30, humidity equals 70 and sound volume equals 65, we get a response of normal from the back end, shown below:



Entering temperature equals 130, humidity equals 70 and sound volume equals 65, we get a response of Anomalous from the back end, shown below:

