POWER SYSTEM FAULT DETECTION AND CLASSIFICATION

A IBM PROJECT REPORT

Submitted by

THIVYA PRIYA G (24213120132)

in partial fulfillment of the award of the degree of

MASTER OF BUSINESS ADMINISTRATION

in

GENERAL MANAGEMENT

SASTRA DEEMED UNIVERSITY,

TANJORE–613401.

JULY 2025

DECLARATION

I hereby declare that the work entitled "Power System Fault Detection and Classification **USING MACHINE LEARNING"** is submitted in partial fulfillment of the requirement for the reward of the degree in MBA, Sastra Deemed University, Tanjore is a record of our own work carried out by me during the academic year 2024-26 under the supervision and guidance of Mr. NARENDRA ELURI from IBM (EDUNET FOUNDATION)**.** The extent and source of information have derived from the existing literature and have been indicated through the dissertation at the appropriate places. The matter embodied in this work is original and has not been submitted for the award of any degree or diploma, either in this or any other University.

G. THIVYA PRIYA (24213120132)

I certify that the declaration made by above candidate is true.

ABSTRACT

Power system fault detection and classification are critical for ensuring the stability, reliability, and safety of electrical power networks. Traditional methods, while effective, often suffer from limitations in speed, accuracy, and adaptability to complex grid conditions. This project proposes a machine learning (ML)-based approach for automatic fault detection and classification in power systems. By leveraging historical data and features such as voltage, current, and frequency variations, various ML models—such as Support Vector Machines (SVM), Random Forests, and Artificial Neural Networks (ANN)—are trained to recognize patterns associated with different types of faults, including single line-to-ground (SLG), line-to-line (LL), double line-to-ground (DLG), and three-phase faults. The proposed system is capable of real-time analysis and delivers high accuracy in identifying fault types and locations, significantly reducing response time and aiding in faster fault clearance. Experimental results demonstrate the superiority of the ML-based approach over conventional techniques, highlighting its potential for intelligent power system monitoring and automation.

CHAPTER 1

INTRODUCTION

In this project, I am implementing a **Power System Fault Detection and Classification** solution using **IBM Cloud**. The primary objective is to detect and classify different types of faults in a power distribution system to ensure reliable and safe operation of electrical networks.

For this purpose, I am utilising **IBM watsonx.ai**, which provides an integrated platform for AI model development, data analysis, and deployment. The electrical measurement data, including voltage and current phasors, is securely stored in **IBM Cloud Object Storage**, enabling efficient data retrieval and management throughout the project.

Using the **watsonx.ai runtime environment**, I performed data preprocessing, feature extraction, and machine learning model development to distinguish between normal operating conditions and various fault conditions such as line-to-ground faults, line-to-line faults, and three-phase faults.

By leveraging **IBM Cloud's scalable infrastructure**, **watsonx.ai's AI capabilities**, and **cloud storage services**, this project ensures:

- **Accurate and real-time fault detection and classification**
- **Efficient utilisation of cloud resources for model training and testing**
- **Enhanced operational reliability** of power distribution systems through automated fault analysis

This approach demonstrates the potential of cloud-based AI solutions in improving the monitoring and protection mechanisms of modern electrical power systems.

PROBLEM STATEMENT

This project focuses on designing a **machine learning model for fault detection and classification in power distribution systems**. By utilising electrical measurement data such as voltage and current phasors, the model distinguishes between normal operating conditions and various fault types, including **line-to-ground, line-to-line, and three-phase faults**. The primary objective is to achieve **rapid and accurate fault identification**, ensuring the **stability, reliability, and safety** of the power grid.

PROPOSED SOLUTION

The proposed system utilises a **machine learning-based approach** integrated with **IBM Cloud services** for efficient fault detection and classification in power distribution systems. The solution involves the following components:

1. **Data Collection and Storage**
    - Electrical measurement data (voltage and current phasors) is collected from the power system sensors.
    - The data is securely stored in **IBM Cloud Object Storage** for scalable access and management.
2. **Data Preprocessing**
    - Data cleaning to remove missing or erroneous values
    - Feature extraction to derive relevant parameters for fault detection
3. **Model Development using watsonx.ai**

- Implementation of classification algorithms such as **Decision Tree, Random Forest, or SVM** within **watsonx.ai's runtime environment**
- Training the model to differentiate between normal operating conditions and fault types (line-to-ground, line-to-line, and three-phase faults)

4. **Model Evaluation**

    - Performance metrics such as accuracy, precision, recall, and confusion matrix analysis are used to validate model effectiveness.

5. **Deployment and Integration**
    - The trained model is deployed through **IBM Cloud** to enable real-time fault detection.
    - Integration with monitoring systems for automated fault alerts and grid stability maintenance.

SYSTEM DEVELOPMENT APPROACH

The development of this system involves the following structured approach:

1. **Problem Definition and Objective Setting**
    - Clearly define the problem: detecting and classifying faults in power distribution systems.
    - Set objectives to achieve **rapid, accurate, and reliable fault identification** for grid stability.

2. **Data Acquisition and Storage**
    - Collect electrical measurement data such as voltage and current phasors.
    - Store the data securely in **IBM Cloud Object Storage** for easy access, scalability, and integration with AI tools.

3. **Data Preprocessing**
    - Clean the dataset by handling missing values and outliers.

- Perform feature extraction and selection to identify relevant attributes for fault classification.

4. **Exploratory Data Analysis (EDA)**
    - Analyse data patterns, visualise class distributions, and understand fault characteristics using statistical and graphical techniques within **watsonx.ai notebooks**.

5. **Model Design and Development**
    - Select suitable classification algorithms (e.g., **Decision Tree, Random Forest, SVM**) to build the fault detection model.
    - Train the model using the preprocessed data within **watsonx.ai's runtime environment**.

6. **Model Evaluation and Validation**
    - Evaluate model performance using metrics such as **accuracy, precision, recall, and confusion matrix**.
    - Validate results to ensure robust fault classification across all fault types.

7. **Deployment**
    - Deploy the trained model on **IBM Cloud** for real-time prediction and integration with monitoring systems.
    - Enable automated alerts for detected faults to aid timely maintenance actions.

8. **Documentation and Reporting**
    - Document each development phase systematically.
    - Prepare reports, visuals, and summaries for academic and practical presentation.

ALGORITHM & DEPLOYMENT

Algorithm

For this project, the **Random Forest Classifier** is proposed due to its high accuracy, robustness to noise, and effectiveness in multi-class fault classification. The steps are as follows:

1. **Input:** Electrical measurement data (voltage and current phasors)

2. **Data Preprocessing:**
   - Handle missing values
   - Extract relevant features for classification
   - Normalize or standardize data if required

3. **Model Training:**
   - Split the data into training and testing sets
   - Train the **Random Forest Classifier** on the training set
   - Tune hyperparameters (number of trees, max depth) to improve performance

4. **Prediction:**
   - Use the trained model to predict fault types on the test data

5. **Output:** Classified fault type (e.g., normal, line-to-ground, line-to-line, three-phase fault)

Deployment

The deployment process using **IBM Cloud and watsonx.ai** involves the following steps:

1. **Model Export:**
   - Save the trained model as a pickle (`.pkl`) file within watsonx.ai.

2. **IBM Cloud Object Storage:**
   - Upload the model file and dataset to **IBM Cloud Object Storage** for secured and scalable storage.

3. **Create Deployment Space in watsonx.ai:**
    - Navigate to **watsonx.ai**
    - Create a new **deployment space** for the project

4. **Model Deployment:**
    - Deploy the saved model as a **web service/API** within watsonx.ai
    - Enable **REST API endpoints** for real-time fault detection and classification

5. **Integration with Applications:**
    - Use the API in monitoring applications to send real-time data and receive fault classification outputs.

6. **Testing and Monitoring:**
    - Test the deployment endpoint with new data samples to validate deployment success.
    - Monitor API usage and model performance over time for continuous improvement.

IMPLEMENTATION

Experiment Notebook - AutoAI Notebook v2.1.7

This notebook contains the steps and code to demonstrate support of AutoAI experiments in the watsonx.ai Runtime. It introduces Python API commands for data retrieval, training experiments, persisting pipelines, testing pipelines, refining pipelines, and scoring the resulting model.

Note: Notebook code generated using AutoAI will execute successfully. If code is modified or reordered, there is no guarantee it will successfully execute. For details, see: Saving an Auto AI experiment as a notebook

Some familiarity with Python is helpful. This notebook uses Python 3.11 and the `ibm-watsonx-ai` package.

Notebook goals

The learning goals of this notebook are:

- Defining an AutoAI experiment
- Training AutoAI models
- Comparing trained models
- Deploying the model as a web service
- Scoring the model to generate predictions

Contents

This notebook contains the following parts:

Setup

Package installation

watsonx.ai connection

Experiment configuration

Experiment metadata

Working with completed AutoAI experiment

Get fitted AutoAI optimizer

Pipelines comparison

Setup

Package installation

Before you use the sample code in this notebook, install the following packages:

- ibm-watsonx-ai,
- autoai-libs,
- lale,
- scikit-learn,
- xgboost,
- lightgbm,
- snapml

```
!pip install ibm-watsonx-ai | tail -n 1

!pip install autoai-libs~=2.0 | tail -n 1

!pip install -U 'lale~=0.8.3' | tail -n 1

!pip install scikit-learn==1.3.* | tail -n 1

!pip install xgboost==2.0.* | tail -n 1

!pip install lightgbm==4.2.* | tail -n 1

!pip install snapml==1.14.* | tail -n 1
```

Experiment configuration

Experiment metadata

This cell defines the metadata for the experiment, including: training_data_references, training_result_reference, experiment_metadata.

```
from ibm_watsonx_ai.helpers import DataConnection

from ibm_watsonx_ai.helpers import ContainerLocation

training_data_references = [

    DataConnection(

        data_asset_id='7acfce2e-aeeb-4b03-8436-d7db0d94520c'

    ),

]

training_result_reference = DataConnection(

    location=ContainerLocation(
path='auto_ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/wml_data/8c223ead-c41f-422b-9679-b20423a5ad28/data/automl',
model_location='auto_ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/wml_data/8c223ead-c41f-422b-9679-b20423a5ad28/data/automl/model.zip',
training_status='auto_ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/wml_data/8c223ead-c41f-422b-9679-b20423a5ad28/training-status.json'

    )

)

experiment_metadata = dict(

    prediction_type='multiclass',

    prediction_column='Fault Type',

    holdout_size=0.1,

    scoring='accuracy',

    csv_separator=',',
```

```
    random_state=33,
    max_number_of_estimators=2,
    training_data_references=training_data_references,
    training_result_reference=training_result_reference,
    deployment_url='https://eu-gb.ml.cloud.ibm.com',
    project_id='b01a4ae2-308f-4baa-8e0f-7ecd5799c138',
    drop_duplicates=True,
    include_batched_ensemble_estimators=[],
    feature_selector_mode='auto'
)
```

watsonx.ai connection

This cell defines the credentials required to work with the watsonx.ai Runtime.

Action: Provide the IBM Cloud apikey, For details, see documentation.

```
import getpass
api_key = getpass.getpass("Please enter your api key (press enter): ")
from ibm_watsonx_ai import Credentials
credentials = Credentials(
    api_key=api_key,
    url=experiment_metadata['deployment_url']
)
```

Working with completed AutoAI experiment

This cell imports the pipelines generated for the experiment. The best pipeline will be saved as a model.

Get fitted AutoAI optimizer

```
from ibm_watsonx_ai.experiment import AutoAI

pipeline_optimizer = AutoAI(credentials,
project_id=experiment_metadata['project_id']).runs.get_optimizer(metadata=experiment_metad
ata)
```

Use `get_params()` to retrieve configuration parameters.

```
pipeline_optimizer.get_params()
```

Pipelines comparison

Use the `summary()` method to list trained pipelines and evaluation metrics information in the form of a Pandas DataFrame. You can use the DataFrame to compare all discovered pipelines and select the one you like for further testing.

```
summary = pipeline_optimizer.summary()

best_pipeline_name = list(summary.index)[0]

Summary
```

Get pipeline as a scikit-learn pipeline model

After you compare the pipelines, download and save a scikit-learn pipeline model object from the AutoAI training job.

Tip: To get a specific pipeline, pass the pipeline name in:

```
pipeline_optimizer.get_pipeline(pipeline_name=pipeline_name)

pipeline_model = pipeline_optimizer.get_pipeline()
```

Next, check the importance of features for selected pipeline.

```
pipeline_optimizer.get_pipeline_details()['features_importance']
```

Tip: If you want to check all the details of the model evaluation metrics, use:

```
pipeline_optimizer.get_pipeline_details()
```

Score the fitted pipeline with the generated scorer using the holdout dataset.

1. **Get sklearn pipeline_model**

```
sklearn_pipeline_model =
pipeline_optimizer.get_pipeline(astype=AutoAI.PipelineTypes.SKLEARN)
```

2. **Get training and testing data**

```
from ibm_watsonx_ai import APIClient

client = APIClient(credentials=credentials)

if 'space_id' in experiment_metadata:

    client.set.default_space(experiment_metadata['space_id'])

else:

    client.set.default_project(experiment_metadata['project_id'])

training_data_references[0].set_client(client)

_, X_test, _, y_test =
training_data_references[0].read(experiment_metadata=experiment_metadata,
with_holdout_split=True, use_flight=True)
```

3. **Define scorer, score the fitted pipeline with the generated scorer using the holdout dataset.**

```
from sklearn.metrics import get_scorer

scorer = get_scorer(experiment_metadata['scoring'])

score = scorer(sklearn_pipeline_model, X_test.values, y_test.values)

print(score)
```

Inspect pipeline

Visualize pipeline model

Preview pipeline model stages as a graph. Each node's name links to a detailed description of the stage.

pipeline_model.visualize()

Preview pipeline model as a Python code

In the next cell, you can preview the saved pipeline model as a Python code.
You can review the exact steps used to create the model.

Note: If you want to get sklearn representation, add the following parameter to the `pretty_print` call: `astype='sklearn'`.

pipeline_model.pretty_print(combinators=**False**, ipython_display=**True**)

Calling the `predict` method

If you want to get a prediction by using the pipeline model object, call `pipeline_model.predict()`.

Note: If you want to work with a pure sklearn model:

- add the following parameter to the `get_pipeline` call: `astype='sklearn'`,
- or `scikit_learn_pipeline = pipeline_model.export_to_sklearn_pipeline()`

Deploy and Score

In this section you will learn how to deploy and score the model as a web service.

You can use the commands below to promote the model to space and create online deployment (web service).

Working with spaces

In this section you will specify a deployment space for organizing the assets for deploying and scoring the model. If you do not have an existing space, you can use Deployment Spaces Dashboard to create a new space, following these steps:

- Click **New Deployment Space**.
- Create an empty space.
- Select Cloud Object Storage.
- Select watsonx.ai Runtime and press **Create**.
- Copy `space_id` and paste it below.

Tip: You can also use the API to prepare the space for your work. Learn more here.

Info: Below cells are `raw` type - in order to run them, change their type to `code` and run them (no need to restart the notebook). You may need to add some additional info (see the **action** below).

Action:

Assign or update space ID below.

Deployment creation

```
target_space_id = input("Enter your space ID here (press enter): ")

from ibm_watsonx_ai.deployment import WebService

service = WebService(
    source_instance_credentials=credentials,
    target_instance_credentials=credentials,
    source_project_id=experiment_metadata['project_id'],
    target_space_id=target_space_id
)

service.create(
    model=best_pipeline_name,
    metadata=experiment_metadata,
```

```
    deployment_name='Best_pipeline_webservice'
)
```

Use the `print` method for the deployment object to show basic information about the service:

```
print(service)
```

To show all available information about the deployment, use the `.get_params()` method.

```
service.get_params()
```

Scoring of webservice

You can make a scoring request by calling `score()` on the deployed pipeline.

If you want to work with the web service in an external Python application, follow these steps to retrieve the service object:

- Initialize the service by `service = WebService(target_instance_credentials=credentials,target_space_id=experiment_metadata['space_id'])`
- Get deployment_id: `service.list()`
- Get webservice object: `service.get('deployment_id')`

After that you can call `service.score(score_records_df)` method. The `score()` method accepts `pandas.DataFrame` objects.

Deleting deployment

You can delete the existing deployment by calling the `service.delete()` command. To list the existing web services, use the `service.list()` method.

Running the AutoAI experiment with Python API

Info: Below cells are raw type - in order to run them, change their type to code and run them (no need to restart the notebook). You may need to add some additional info.

If you want to run the AutoAI experiment using the Python API, follow these steps. The experiment settings were generated basing on parameters set in the AutoAI UI.

```
from ibm_watsonx_ai.experiment import AutoAI

experiment = AutoAI(credentials, project_id=experiment_metadata['project_id'])

OPTIMIZER_NAME = 'custom_name'

from ibm_watsonx_ai.helpers import DataConnection

from ibm_watsonx_ai.helpers import ContainerLocation

training_data_references = [

    DataConnection(

        data_asset_id='7acfce2e-aeeb-4b03-8436-d7db0d94520c'

    ),

]

training_result_reference = DataConnection(

    location=ContainerLocation(
path='auto_ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/wml_data/8c223ead-c41f-422b-9679-b20423a5ad28/data/automl',
model_location='auto_ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/wml_data/8c223ead-c41f-422b-9679-b20423a5ad28/data/automl/model.zip',


training_status='auto_ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/wml_data/8c223ead-c41f-422b-9679-b20423a5ad28/training-status.json'

    )

)
```

The new pipeline optimizer will be created and training will be triggered.

```
pipeline_optimizer = experiment.optimizer(
    name=OPTIMIZER_NAME,
    prediction_type=experiment_metadata['prediction_type'],
    prediction_column=experiment_metadata['prediction_column'],
    scoring=experiment_metadata['scoring'],
    holdout_size=experiment_metadata['holdout_size'],
    csv_separator=experiment_metadata['csv_separator'],
    drop_duplicates=experiment_metadata['drop_duplicates'],
include_batched_ensemble_estimators=experiment_metadata['include_batched_ensemble_esti
mators'],
    incremental_learning=False,
    feature_selector_mode=experiment_metadata['feature_selector_mode'],
)
pipeline_optimizer.fit(
    training_data_references=training_data_references,
    training_results_reference=training_result_reference,
)
```

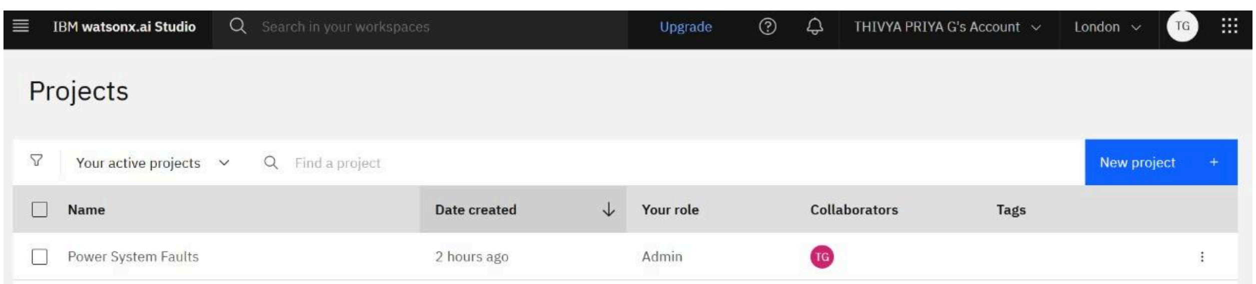
Next steps

You successfully completed this notebook! You learned how to use ibm-watsonx-ai to run and explore AutoAI experiments. Check out the official AutoAI site for more samples, tutorials, documentation, how-tos, and blog posts.

STEPS WITH SCREEN SHOT

NORMAL STEPS TO BEGIN PROJECT

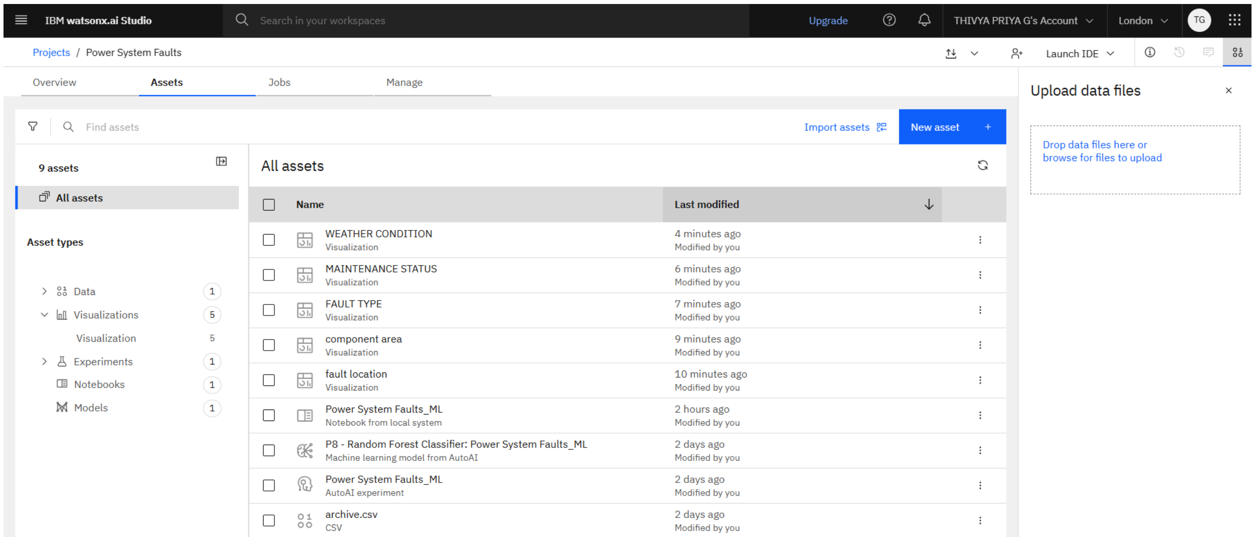
- Create a IBM cloud account using https://cloud.ibm.com/login
- Give the details what they ask & enter given code from https://academic.ibm.com/a2mt/email-auth#/
- Now text in search as watsonx.ai and click it and follow the steps like it asked
- So, in my project iam using with watsonx.ai, watsonx.runtime.ai, cloud object storage
- Then begins with new project "Power System Fault Detection and Classification" using machine learning



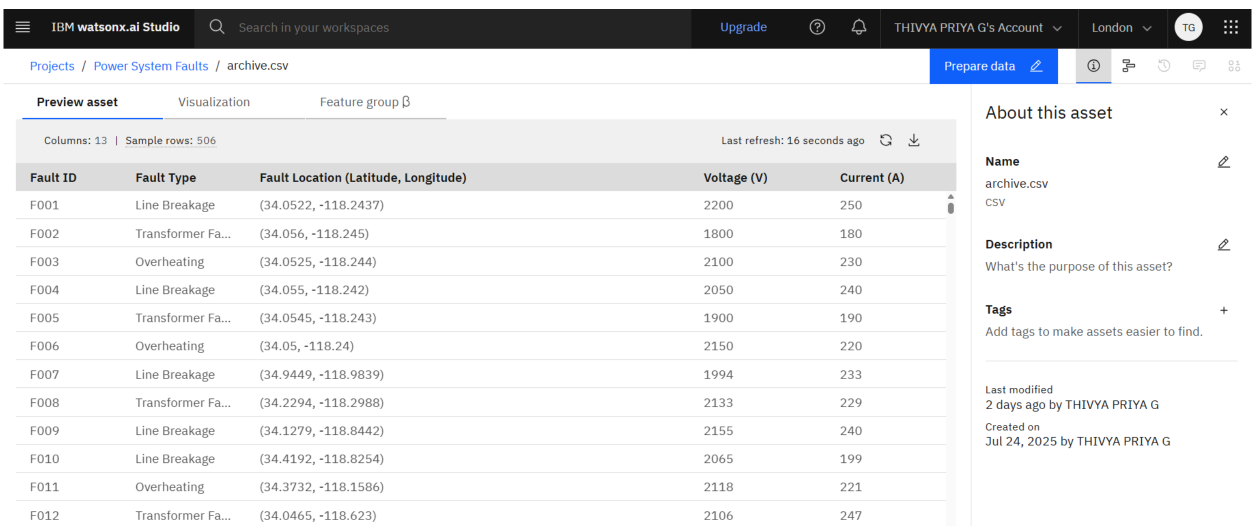
ASSETS

DATASETS

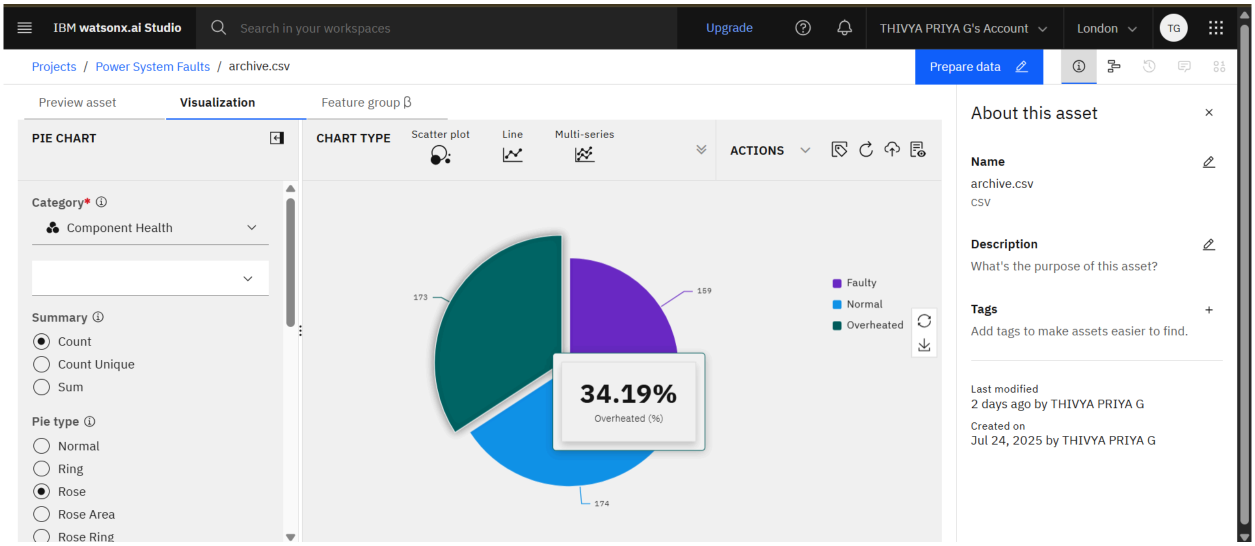
- I have downloaded my datasets through kaggle account which is Power System Faults Dataset and upload in assets and associate with it.

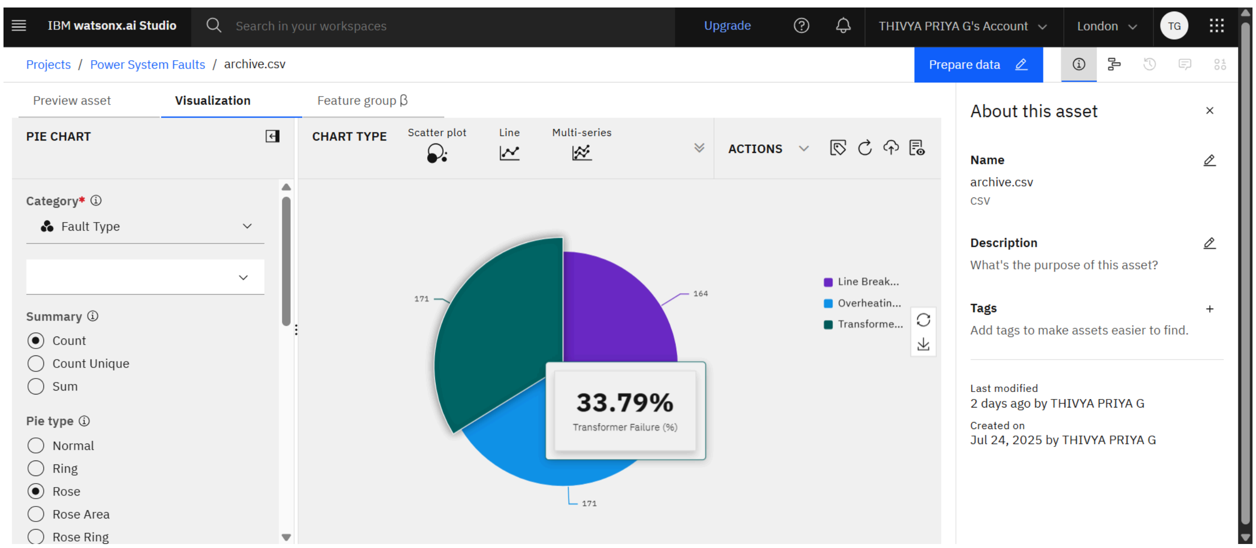


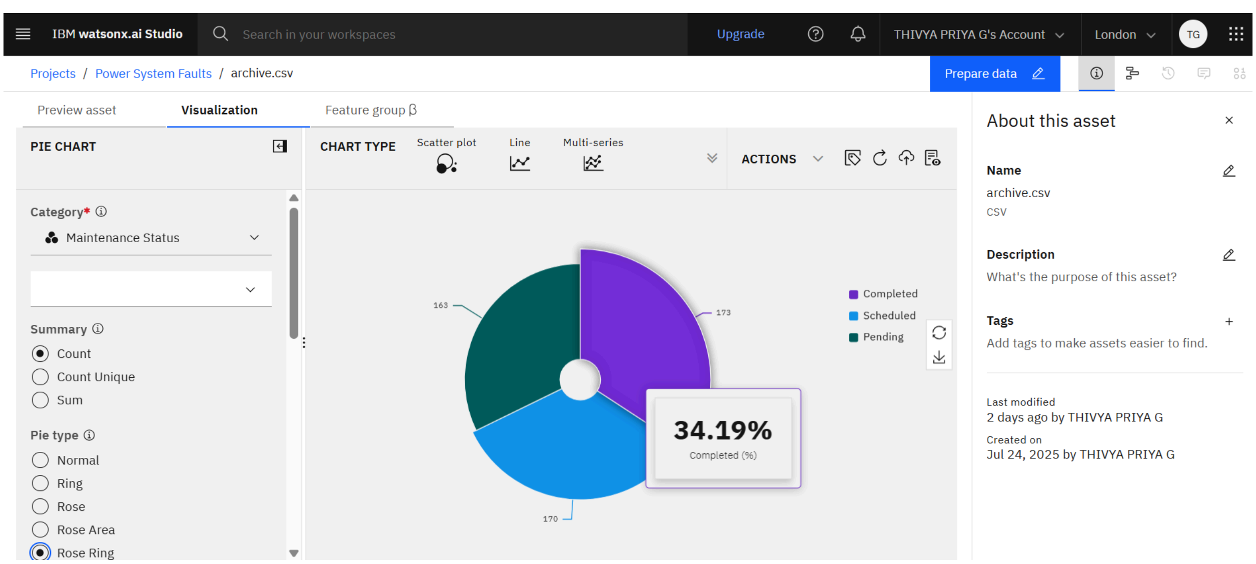
PREVIEW ASSET

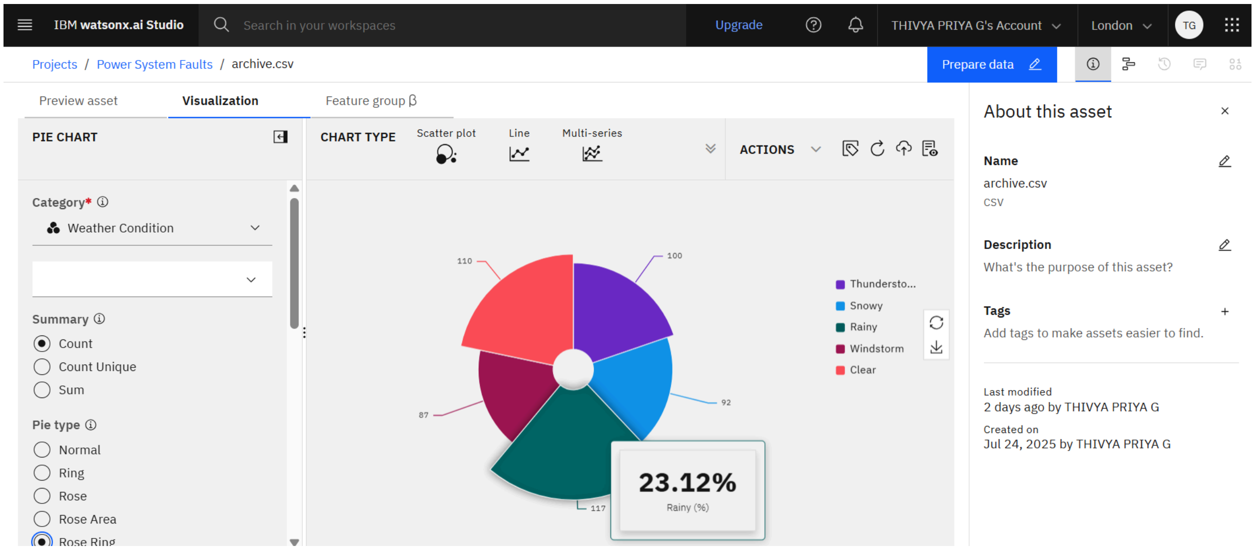


VISUALIZATION


IBM watsonx.ai Studio
Search in your workspaces
Upgrade
THIVYA PRIYA G's Account
London
TG
Projects / Power System Faults / archive.csv
Prepare data
Preview asset
Visualization
Feature group β
PIE CHART
CHART TYPE
Scatter plot
Line
Multi-series
ACTIONS
Category*
Component Health
Summary
Count
Count Unique
Sum
Pie type
Normal
Ring
Rose
Rose Area
Rose Ring
173
159
174
34.19%
Overheated (%)
Faulty
Normal
Overheated
About this asset
Name
archive.csv
CSV
Description
What's the purpose of this asset?
Tags
Add tags to make assets easier to find.
Last modified
2 days ago by THIVYA PRIYA G
Created on
Jul 24, 2025 by THIVYA PRIYA G


IBM watsonx.ai Studio
Search in your workspaces
Upgrade
THIVYA PRIYA G's Account
London
TG
Projects / Power System Faults / archive.csv
Prepare data
Preview asset
Visualization
Feature group β
PIE CHART
CHART TYPE
Scatter plot
Line
Multi-series
ACTIONS
Category*
Fault Type
Summary
Count
Count Unique
Sum
Pie type
Normal
Ring
Rose
Rose Area
Rose Ring
171
164
171
33.79%
Transformer Failure (%)
Line Break...
Overheatin...
Transforme...
About this asset
Name
archive.csv
CSV
Description
What's the purpose of this asset?
Tags
Add tags to make assets easier to find.
Last modified
2 days ago by THIVYA PRIYA G
Created on
Jul 24, 2025 by THIVYA PRIYA G


IBM watsonx.ai Studio
Search in your workspaces
Upgrade
THIVYA PRIYA G's Account
London
TG
Projects / Power System Faults / archive.csv
Prepare data
Preview asset
Visualization
Feature group β
PIE CHART
CHART TYPE
Scatter plot
Line
Multi-series
ACTIONS
Category*
Maintenance Status
Summary
Count
Count Unique
Sum
Pie type
Normal
Ring
Rose
Rose Area
Rose Ring
163
173
170
34.19%
Completed (%)
Completed
Scheduled
Pending
About this asset
Name
archive.csv
CSV
Description
What's the purpose of this asset?
Tags
Add tags to make assets easier to find.
Last modified
2 days ago by THIVYA PRIYA G
Created on
Jul 24, 2025 by THIVYA PRIYA G



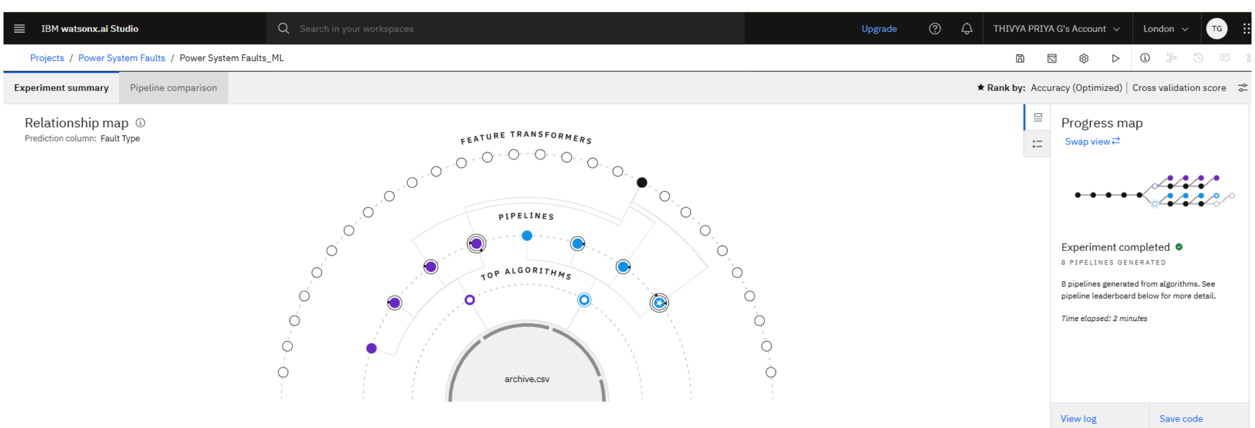
Followed by,

EXPERIMENTS

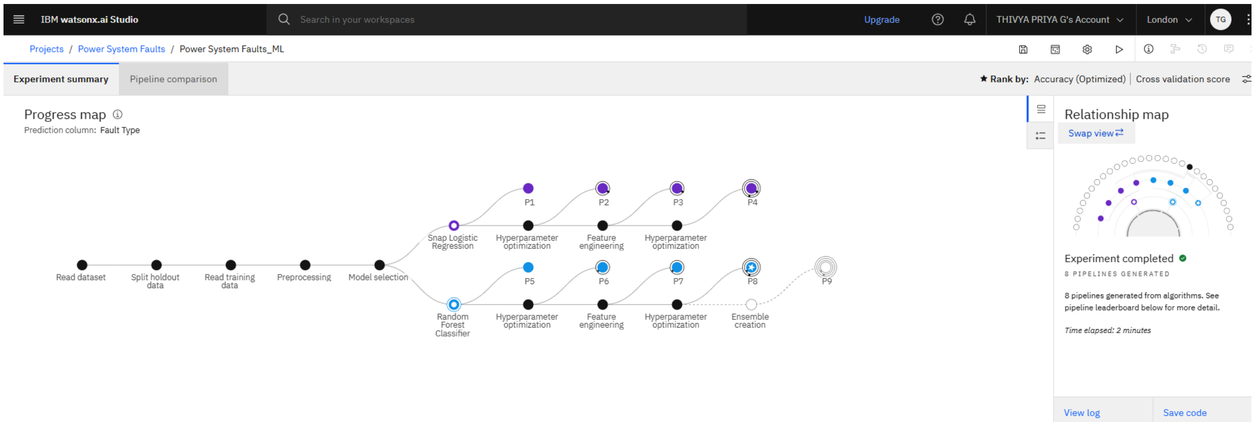
1) EXPERIMENT SUMMARY

https://eu-gb.dataplatform.cloud.ibm.com/ml/auto-ml/e9aec363-24bb-4881-8f8e-b6368dd6e8d4/train?projectid=b01a4ae2-308f-4baa-8e0f-7ecd5799c138&context=cpdaas

RELATIONSHIP MAP



PROGRESS MAP



CROSS VALIDATION SCORE FOR THE GIVEN METRICS:

RANKED BY: ACCURACY (OPTIMIZED)

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	HPO-1 FE	00:00:32
	5	Pipeline 6	Random Forest Classifier		0.369	HPO-1	00:00:06
	6	Pipeline 2	Snap Logistic Regression		0.367	HPO-1	00:00:04
	7	Pipeline 5	Random Forest Classifier		0.360	*None*	00:00:01
	8	Pipeline 1	Snap Logistic Regression		0.358	*None*	00:00:01

RANKED BY: F1 (MACRO)

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	HPO-1 FE	00:00:32

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	Enhancements	Build time
	5	Pipeline 6	Random Forest Classifier		0.369	0.368	HPO-1	00:00:06
	6	Pipeline 2	Snap Logistic Regression		0.367	0.365	HPO-1	00:00:04
	7	Pipeline 5	Random Forest Classifier		0.360	0.358	*None*	00:00:01
	8	Pipeline 1	Snap Logistic Regression		0.358	0.357	*None*	00:00:01

RANKED BY: F1 (MICRO)

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	HPO-1 FE	00:00:32

Pipeline leaderboard

Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	Enhancements	Build time
5	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	HPO-1	00:00:06
6	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	HPO-1	00:00:04
7	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	*None*	00:00:01
8	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	*None*	00:00:01

RANKED BY: F1 (WEIGHTED)

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	F_1 weighted Cross Validation	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	0.406	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	0.389	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	0.389	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	0.372	HPO-1 FE	00:00:32

Pipeline leaderboard

Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	F_1 weighted Cross Validation	Enhancements	Build time
5	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	0.368	HPO-1	00:00:06
6	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	0.366	HPO-1	00:00:04
7	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	0.358	*None*	00:00:01
8	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	0.357	*None*	00:00:01

RANKED BY: LOG LOSS

Cross Validation Holdout

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	F_1 weighted Cross Validation	Log loss Cross Validation	Enhancements	Build time
★	1	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	HPO-1 FE HPO-2	00:00:26
	2	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	HPO-1 FE	00:00:22
	3	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	0.366	1.095	HPO-1	00:00:04
	4	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	0.357	1.098	*None*	00:00:01

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	F_1 weighted Cross Validation	Log loss Cross Validation	Enhancements	Build time
	5	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	0.406	1.102	HPO-1 FE HPO-2	00:00:44
	6	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	0.368	1.106	HPO-1	00:00:06
	7	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	0.372	1.114	HPO-1 FE	00:00:32
	8	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	0.358	1.912	*None*	00:00:01

RANKED BY: PRECISION (MACRO)

Pipeline leaderboard ▽

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	F_1 weighted Cross Validation	Log loss Cross Validation	Precision macro Cross Validation	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	0.406	1.102	0.407	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	0.372	1.114	0.375	HPO-1 FE	00:00:32
	5	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	0.368	1.106	0.368	HPO-1	00:00:06
	6	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	0.366	1.095	0.367	HPO-1	00:00:04
	7	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	0.357	1.098	0.359	*None*	00:00:01
	8	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	0.358	1.912	0.358	*None*	00:00:01

RANKED BY: PRECISION (MICRO)

Pipeline leaderboard ▽

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F_1 macro Cross Validation	F_1 micro Cross Validation	F_1 weighted Cross Validation	Log loss Cross Validation	Precision macro Cross Validation	Precision micro Cross Validation	Enhancements
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	0.406	1.102	0.407	0.409	HPO-1 FE HPO-2
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	0.393	HPO-1 FE HPO-2
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	0.393	HPO-1 FE
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	0.372	1.114	0.375	0.376	HPO-1 FE
	5	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	0.368	1.106	0.368	0.369	HPO-1
	6	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	0.366	1.095	0.367	0.367	HPO-1
	7	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	0.358	1.912	0.358	0.360	*None*
	8	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	0.357	1.098	0.359	0.358	*None*

RANKED BY: PRECISION (WEIGHTED)

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F1 macro Cross Validation	F1 micro Cross Validation	F1 weighted Cross Validation	Log loss Cross Validation	Precision macro Cross Validation	Precision micro Cross Validation	Precision weighted Cross Validation
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	0.406	1.102	0.407	0.409	0.408
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	0.393	0.394
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	0.393	0.394
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	0.372	1.114	0.375	0.376	0.376

Pipeline leaderboard

Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F1 macro Cross Validation	F1 micro Cross Validation	F1 weighted Cross Validation	Log loss Cross Validation	Precision macro Cross Validation	Precision micro Cross Validation	Precision weighted Cross Validation
5	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	0.368	1.106	0.368	0.369	0.369
6	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	0.366	1.095	0.367	0.367	0.368
7	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	0.357	1.098	0.359	0.358	0.359
8	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	0.358	1.912	0.358	0.360	0.358

RANKED BY: RECALL (MACRO, MICRO, WEIGHTED)

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	F1 macro Cross Validation	F1 micro Cross Validation	F1 weighted Cross Validation	Log loss Cross Validation	Precision macro Cross Validation	Precision micro Cross Validation	Precision weighted Cross Validation	Recall macro Cross Validation	Recall micro Cross Validation	Recall weighted Cross Validation	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	0.405	0.409	0.406	1.102	0.407	0.409	0.408	0.408	0.409	0.409	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	0.393	0.394	0.392	0.393	0.393	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	0.388	0.393	0.389	1.088	0.393	0.393	0.394	0.392	0.393	0.393	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	0.371	0.376	0.372	1.114	0.375	0.376	0.376	0.375	0.376	0.376	HPO-1 FE	00:00:32
	5	Pipeline 6	Random Forest Classifier		0.369	0.368	0.369	0.368	1.106	0.368	0.369	0.369	0.369	0.369	0.369	HPO-1	00:00:06
	6	Pipeline 2	Snap Logistic Regression		0.367	0.365	0.367	0.366	1.095	0.367	0.367	0.368	0.367	0.367	0.367	HPO-1	00:00:04
	7	Pipeline 5	Random Forest Classifier		0.360	0.358	0.360	0.358	1.912	0.358	0.360	0.358	0.360	0.360	0.360	None	00:00:01
	8	Pipeline 1	Snap Logistic Regression		0.358	0.357	0.358	0.357	1.098	0.359	0.358	0.359	0.358	0.358	0.358	None	00:00:01

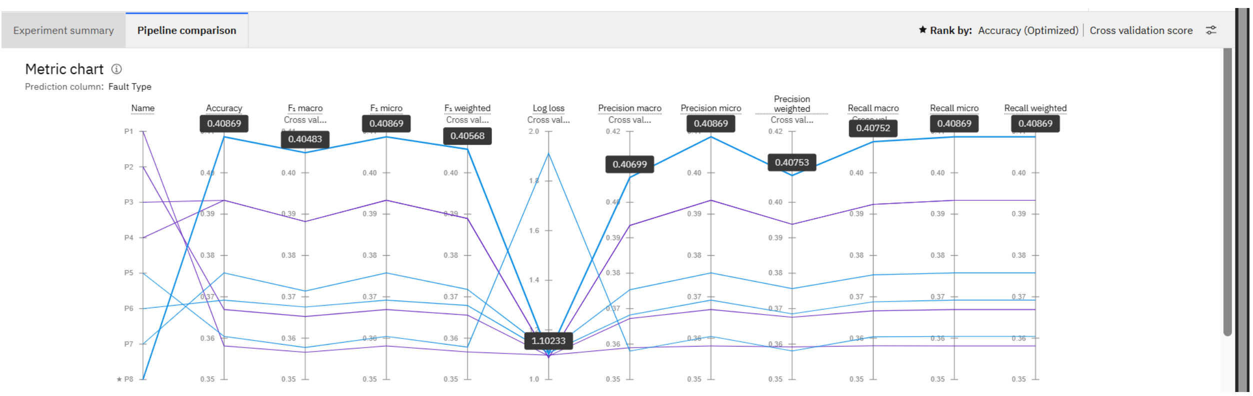
2)PIPELINE COMPARISON

HOLDOUT SCORE FOR THE GIVEN METRICS:

Pipeline leaderboard

	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	Accuracy (Optimized) Holdout	F1 macro Cross Validation	F1 micro Cross Validation	F1 weighted Cross Validation	Log loss Cross Validation	Precision macro Cross Validation	Precision micro Cross Validation	Precision weighted Cross Validation	Recall macro Cross Validation	Recall macro Holdout	Recall micro Cross Validation	Recall weighted Cross Validation	Enhancements	Build time
★	1	Pipeline 6	Random Forest Classifier		0.369	0.451	0.368	0.369	0.368	1.106	0.368	0.369	0.369	0.369	0.451	0.369	0.369	HPO-1	00:00:06
	2	Pipeline 1	Snap Logistic Regression		0.358	0.392	0.357	0.358	0.357	1.098	0.359	0.358	0.359	0.358	0.392	0.358	0.358	None	00:00:01
	3	Pipeline 2	Snap Logistic Regression		0.367	0.373	0.365	0.367	0.366	1.095	0.367	0.367	0.368	0.367	0.373	0.367	0.367	HPO-1	00:00:04
	4	Pipeline 4	Snap Logistic Regression		0.393	0.333	0.388	0.393	0.389	1.088	0.393	0.393	0.394	0.392	0.333	0.393	0.393	HPO-1 FE HPO-2	00:00:26
	5	Pipeline 3	Snap Logistic Regression		0.393	0.333	0.388	0.393	0.389	1.088	0.393	0.393	0.394	0.392	0.333	0.393	0.393	HPO-1 FE	00:00:22
	6	Pipeline 8	Random Forest Classifier		0.409	0.314	0.405	0.409	0.406	1.102	0.407	0.409	0.408	0.408	0.314	0.409	0.409	HPO-1 FE HPO-2	00:00:44
	7	Pipeline 7	Random Forest Classifier		0.376	0.314	0.371	0.376	0.372	1.114	0.375	0.376	0.376	0.375	0.314	0.376	0.376	HPO-1 FE	00:00:32
	8	Pipeline 5	Random Forest Classifier		0.360	0.314	0.358	0.360	0.358	1.912	0.358	0.360	0.358	0.360	0.314	0.360	0.360	None	00:00:01

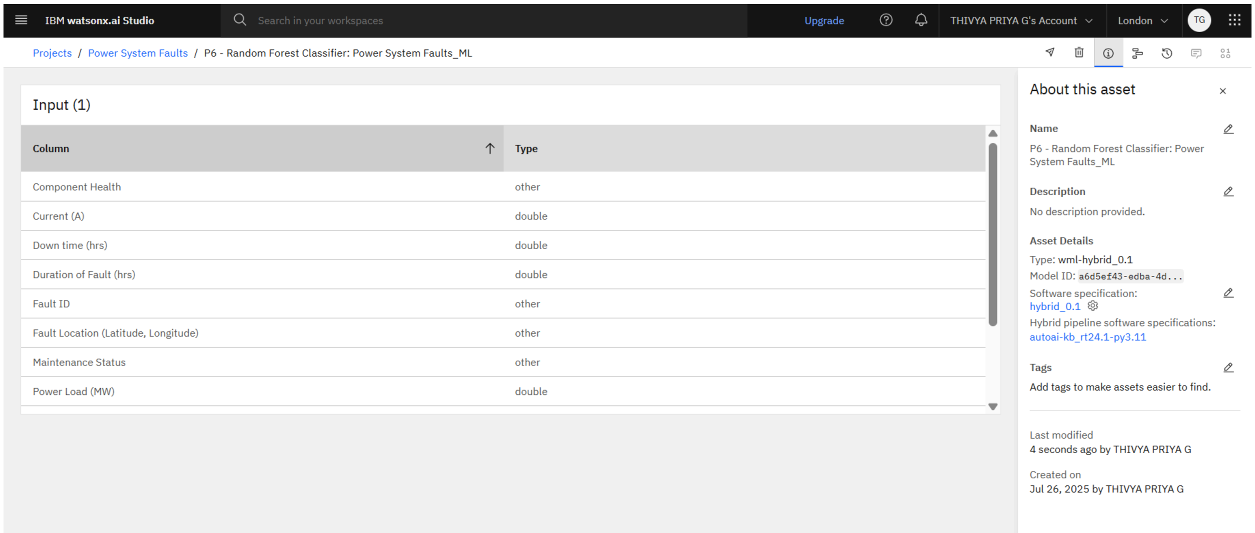
CROSS VALIDATION SCORE FOR THE GIVEN METRICS:



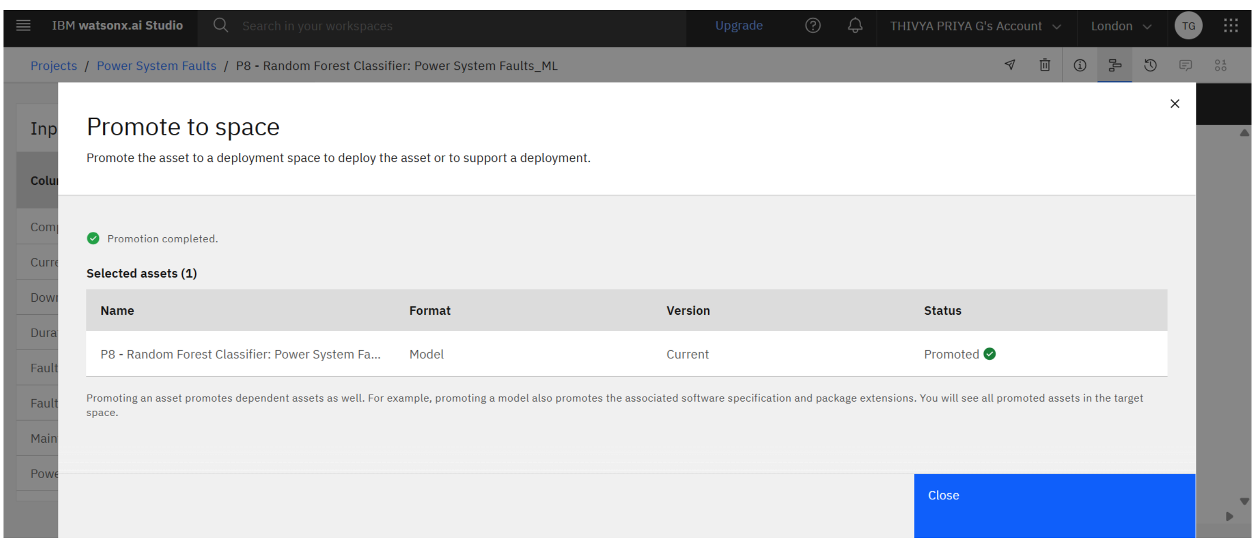
Pipeline leaderboard

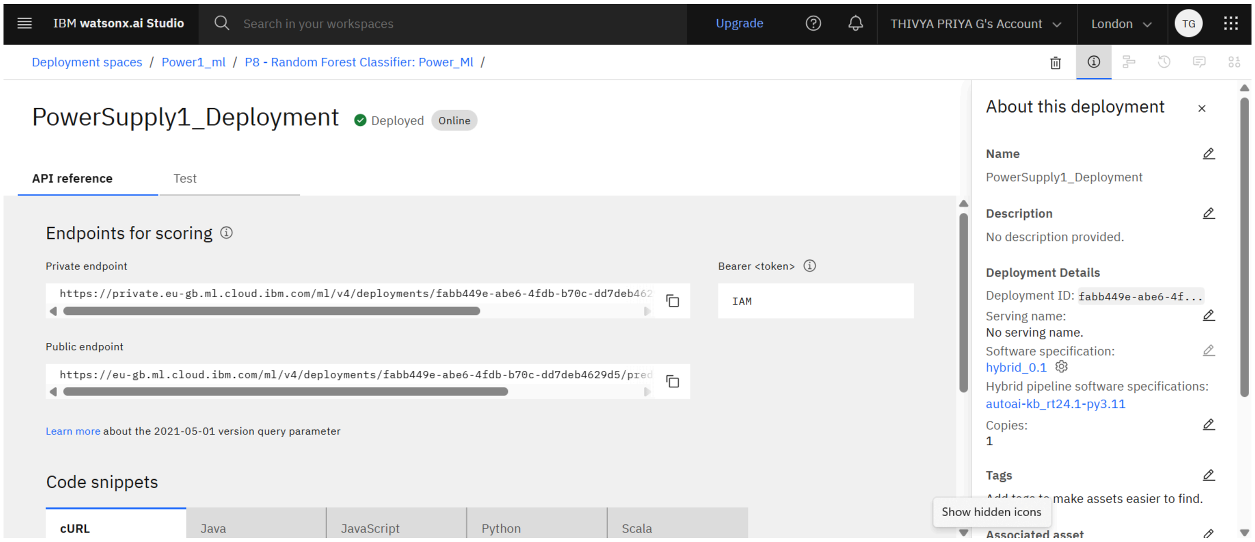
	Rank	Name	Algorithm	Specialization	Accuracy (Optimized) Cross Validation	Accuracy (Optimized) Holdout	F1 macro Cross Validation	F1 macro Holdout	F1 micro Cross Validation	F1 weighted Cross Validation	Log loss Cross Validation	Log loss Holdout	Precision macro Cross Validation	Precision micro Cross Validation	Precision weighted Cross Validation	Recall macro Cross Validation	Recall macro Holdout	Recall micro Cross Validation	Recall weighted Cross Validation	Recall weighted Holdout	Enhancements	Build time
★	1	Pipeline 8	Random Forest Classifier		0.409	0.314	0.405	0.301	0.409	0.406	1.102	1.128	0.407	0.409	0.408	0.408	0.314	0.409	0.409	0.314	HPO-1 FE HPO-2	00:00:44
	2	Pipeline 4	Snap Logistic Regression		0.393	0.333	0.388	0.318	0.393	0.389	1.088	1.108	0.393	0.393	0.394	0.392	0.333	0.393	0.393	0.333	HPO-1 FE HPO-2	00:00:26
	3	Pipeline 3	Snap Logistic Regression		0.393	0.333	0.388	0.318	0.393	0.389	1.088	1.108	0.393	0.393	0.394	0.392	0.333	0.393	0.393	0.333	HPO-1 FE	00:00:22
	4	Pipeline 7	Random Forest Classifier		0.376	0.314	0.371	0.286	0.376	0.372	1.114	1.126	0.375	0.376	0.376	0.375	0.314	0.376	0.376	0.314	HPO-1 FE	00:00:32
	5	Pipeline 6	Random Forest Classifier		0.369	0.451	0.368	0.442	0.369	0.368	1.106	1.077	0.368	0.369	0.369	0.369	0.451	0.369	0.369	0.451	HPO-1	00:00:06
	6	Pipeline 2	Snap Logistic Regression		0.367	0.373	0.365	0.361	0.367	0.366	1.095	1.086	0.367	0.367	0.368	0.367	0.373	0.367	0.367	0.373	HPO-1	00:00:04
	7	Pipeline 5	Random Forest Classifier		0.360	0.314	0.358	0.315	0.360	0.358	1.912	1.222	0.358	0.360	0.358	0.360	0.314	0.360	0.360	0.314	None	00:00:01
	8	Pipeline 1	Snap Logistic Regression		0.358	0.392	0.357	0.371	0.358	0.357	1.098	1.076	0.359	0.358	0.359	0.358	0.392	0.358	0.358	0.392	None	00:00:01

MODEL



PROMOTE TO SPACE (TO CREATE DEPLOYMENT)





API REFERENCE:

PRIVATE END POINTS

https://private.eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7deb4629d5/predictions?version=2021-05-01

PUBLIC END POINTS

https://eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7deb4629d5/predictions?version=2021-05-01

CODE SNIPPETS

cURL

NOTE: you must set $API_KEY below using information retrieved from your IBM Cloud account (https://eu-gb.dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-authentication.html?context=cpdaas)

export API_KEY=<your API key>

export IAM_TOKEN=$(curl --insecure -X POST --location "https://iam.cloud.ibm.com/identity/token" \

```
--header "Content-Type: application/x-www-form-urlencoded" \

--header "Accept: application/json" \

--data-urlencode "grant_type=urn:ibm:params:oauth:grant-type:apikey" \

--data-urlencode "apikey=$API_KEY" | jq -r '.access_token')

# TODO:  manually define and pass values to be scored below

curl --location
"https://private.eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7deb46
29d5/predictions?version=2021-05-01" \

--header "Content-Type: application/json" \

--header "Accept: application/json" \

--header "Authorization: Bearer $IAM_TOKEN" \

--data "{

   \"input_data\": [

       {

           \"fields\": [$ARRAY_OF_INPUT_FIELDS],

           \"values\": [[$ARRAY_OF_VALUES_TO_BE_SCORED],
[$ANOTHER_ARRAY_OF_VALUES_TO_BE_SCORED]]

       }

   ]

}"
```

JAVA

```
import java.io.*;

import java.net.MalformedURLException;

import java.util.Base64;

import java.util.HashMap;
```

```
import java.util.Map;

import java.net.HttpURLConnection;

import java.net.URL;

import java.nio.charset.StandardCharsets;

public class HttpClientTest {

	public static void main(String[] args) throws IOException {

		// NOTE: you must manually set API_KEY below using information retrieved from
your IBM Cloud account.
(https://eu-gb.dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-authentication.html
?context=cpdaas)

		String API_KEY = "<your API key>";

		HttpURLConnection tokenConnection = null;

		HttpURLConnection scoringConnection = null;

		BufferedReader tokenBuffer = null;

		BufferedReader scoringBuffer = null;

		try {

			// Getting IAM token

			URL tokenUrl = new
URL("https://iam.cloud.ibm.com/identity/token?grant_type=urn:ibm:params:oauth:grant-type:api
key&apikey=" + API_KEY);

			tokenConnection = (HttpURLConnection) tokenUrl.openConnection();

			tokenConnection.setDoInput(true);

			tokenConnection.setDoOutput(true);

			tokenConnection.setRequestMethod("POST");

			tokenConnection.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");

			tokenConnection.setRequestProperty("Accept", "application/json");
```

```
                    if (tokenConnection.getResponseCode() == 200) { // Successful response

                            tokenBuffer = new BufferedReader(new
InputStreamReader(tokenConnection.getInputStream()));

                    } else { // Error response

                            tokenBuffer = new BufferedReader(new
InputStreamReader(tokenConnection.getErrorStream()));

                    }

        String line;

                    StringBuffer jsonString = new StringBuffer();

        while ((line = tokenBuffer.readLine()) != null) {

           jsonString.append(line);

        }

        System.out.println("Token response body:\n" + jsonString);

                    // Scoring request

                    URL scoringUrl = new
URL("https://private.eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7d
eb4629d5/predictions?version=2021-05-01");

                    String iam_token = "Bearer " +
jsonString.toString().split(":")[1].split("\"")[1];

                    scoringConnection = (HttpURLConnection) scoringUrl.openConnection();

                    scoringConnection.setDoInput(true);

                    scoringConnection.setDoOutput(true);

                    scoringConnection.setRequestMethod("POST");

                    scoringConnection.setRequestProperty("Accept", "application/json");

                    scoringConnection.setRequestProperty("Authorization", iam_token);

                    scoringConnection.setRequestProperty("Content-Type", "application/json;
charset=UTF-8");
```

```
			OutputStreamWriter writer = new
OutputStreamWriter(scoringConnection.getOutputStream(), "UTF-8");

			// NOTE:  manually define and pass the array(s) of values to be scored in
the next line

			String payload = """

			{\"input_data\": [

				{

					\"fields\": [array_of_input_fields],

					\"values\": [array_of_values_to_be_scored,
another_array_of_values_to_be_scored]

				}

			]}""";

			writer.write(payload);

			writer.close();

			if (scoringConnection.getResponseCode() == 200) { // Successful
response

				scoringBuffer = new BufferedReader(new
InputStreamReader(scoringConnection.getInputStream()));

			} else { // Error response

				scoringBuffer = new BufferedReader(new
InputStreamReader(scoringConnection.getErrorStream()));

			}

	String lineScoring;

			StringBuffer jsonStringScoring = new StringBuffer();

	while ((lineScoring = scoringBuffer.readLine()) != null) {

	    jsonStringScoring.append(lineScoring);

	}
```

```
        System.out.println("Scoring response body:\n" + jsonStringScoring);
            } catch (IOException e) {
                System.out.println("The request was not valid.");
                System.out.println(e.getMessage());
            }
            finally {
                if (tokenConnection != null) {
                    tokenConnection.disconnect();
                }
                if (tokenBuffer != null) {
                    tokenBuffer.close();
                }
                if (scoringConnection != null) {
                    scoringConnection.disconnect();
                }
                if (scoringBuffer != null) {
                    scoringBuffer.close();
                }
            }
        }
}
```

JAVASCRIPT

```
const XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
```

```
// NOTE: you must manually enter your API_KEY below using information retrieved from your
IBM Cloud account
(https://eu-gb.dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-authentication.html
?context=cpdaas)

const API_KEY = "<your API key>";

function getToken(errorCallback, loadCallback) {

	const req = new XMLHttpRequest();

	req.addEventListener("load", loadCallback);

	req.addEventListener("error", errorCallback);

	req.open("POST", "https://iam.cloud.ibm.com/identity/token");

	req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

	req.setRequestHeader("Accept", "application/json");

	req.send("grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey=" + API_KEY);

}

function apiPost(scoring_url, token, payload, loadCallback, errorCallback){

	const oReq = new XMLHttpRequest();

	oReq.addEventListener("load", loadCallback);

	oReq.addEventListener("error", errorCallback);

	oReq.open("POST", scoring_url);

	oReq.setRequestHeader("Accept", "application/json");

	oReq.setRequestHeader("Authorization", "Bearer " + token);

	oReq.setRequestHeader("Content-Type", "application/json;charset=UTF-8");

	oReq.send(payload);

}

getToken((err) => console.log("An error occurred submitting the request."), () => {

	let tokenResponse;
```

```
try {
    tokenResponse = JSON.parse(this.responseText);
} catch(ex) {
    // TODO: handle parsing exception
}
// NOTE:  manually define and pass the array(s) of values to be scored in the next line
const payload = `{"input_data": [
    {
        "fields": [array_of_input_fields],
        "values": [array_of_values_to_be_scored,
another_array_of_values_to_be_scored]
    }
]}`;
const scoring_url =
"https://private.eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7deb46
29d5/predictions?version=2021-05-01";
apiPost(scoring_url, tokenResponse.access_token, payload, function (resp) {
    let parsedPostResponse;
    try {
        parsedPostResponse = JSON.parse(this.responseText);
    } catch (ex) {
        // TODO: handle parsing exception
    }
    console.log("Scoring response");
    console.log(parsedPostResponse);
}, function (error) {
```

```
                    console.log(error);

          });

});
```

PYTHON

```
import requests

# NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account (https://eu-gb.dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-authentication.html?context=cpdaas)

API_KEY = "<your API key>"

token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey": API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})

mltoken = token_response.json()["access_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

# NOTE:  manually define and pass the array(s) of values to be scored in the next line

payload_scoring = {"input_data": [

          {

                    "fields": [array_of_input_fields],

                    "values": [array_of_values_to_be_scored, another_array_of_values_to_be_scored]

          }

]}

response_scoring = requests.post('https://private.eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7deb4629d5/predictions?version=2021-05-01', json=payload_scoring,

 headers={'Authorization': 'Bearer ' + mltoken})

print("Scoring response")
```

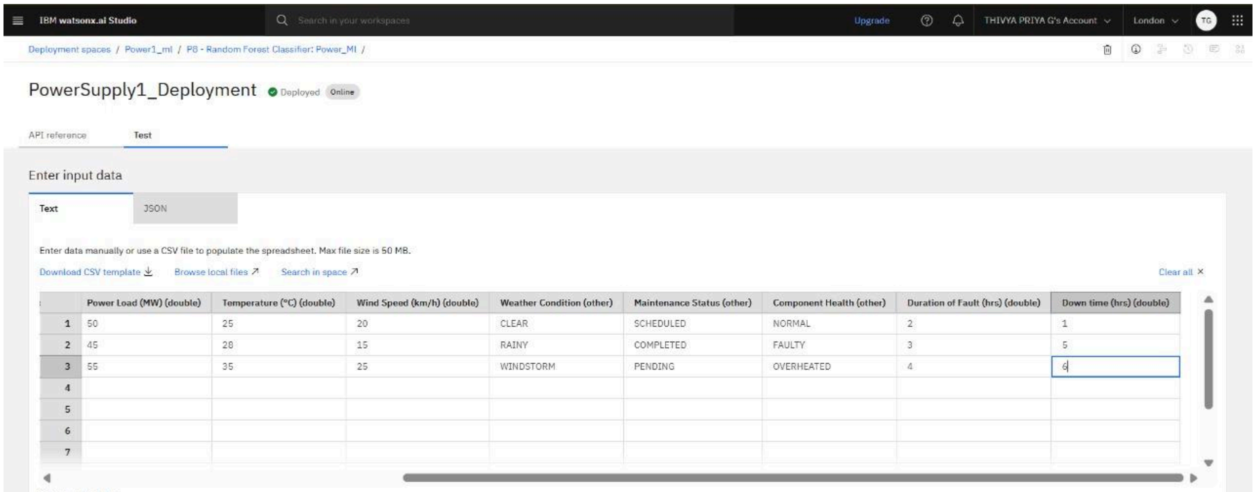
```
try:
    print(response_scoring.json())
except ValueError:
    print(response_scoring.text)
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

SCALA

```
import scalaj.http.{Http, HttpOptions}
import scala.util.{Success, Failure}
import java.util.Base64
import java.nio.charset.StandardCharsets
import play.api.libs.json._
// NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account (https://eu-gb.dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-authentication.html?context=cpdaas)
val API_KEY = "<your API key>"
// Get IAM service token
val iam_url = "https://iam.cloud.ibm.com/identity/token"
val iam_response = Http(iam_url).header("Content-Type", "application/x-www-form-urlencoded").header("Accept",
 "application/json").postForm(Seq("grant_type" -> "urn:ibm:params:oauth:grant-type:apikey",
  "apikey" -> API_KEY)).asString
val iamtoken_json: JsValue = Json.parse(iam_response.body)
val iamtoken = (iamtoken_json \ "access_token").asOpt[String] match {
```

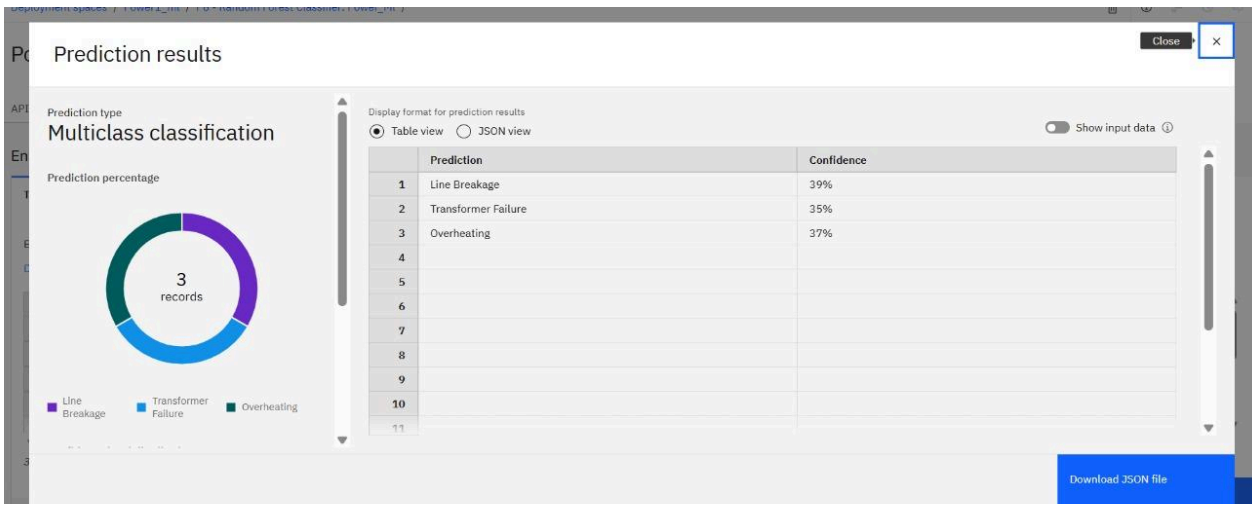
```
            case Some(x) => x

            case None => ""

}

// TODO:  manually define and pass list of values to be scored

val payload_scoring = Json.stringify(Json.toJson(Map("input_data" -> List(Map(

            "fields" -> Json.toJson(List(list_of_input_fields)),

            "values" -> Json.toJson(list_of_values_to_be_scored)

)))))

val scoring_url =
"https://private.eu-gb.ml.cloud.ibm.com/ml/v4/deployments/fabb449e-abe6-4fdb-b70c-dd7deb46
29d5/predictions?version=2021-05-01"

val response_scoring = Http(scoring_url).postData(payload_scoring).header("Content-Type",

 "application/json").header("Authorization", "Bearer " + iamtoken).option(HttpOptions.

method("POST")).option(HttpOptions.connTimeout(10000)).option(HttpOptions.readTimeout(50
000)).asString

println("scoring response")

println(response_scoring)
```

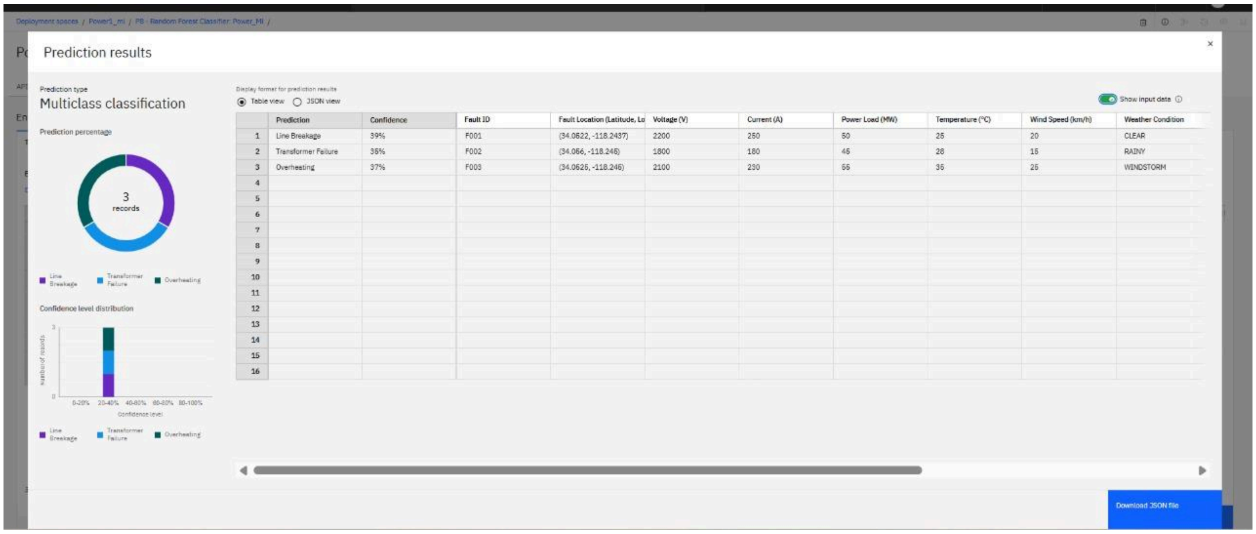
TEST:



CLICK PREDICT, AND GET THIS AS AN OUTPUT

TABLE VIEW





JSON VIEW


```
[
    {
        "fields": [
            "prediction",
            "probability"
        ],
        "values": [
            [
                "Line Breakage",
                [
                    0.3903001601394518,
                    0.2418251292774404,
                    0.36787471058310767
                ]
            ],
```

[
"Transformer Failure",
[
0.305630902371415,
0.3409365463246582,
0.3534325513039267
]
],
[
"Overheating",
[
0.29108219796175333,
0.37123284986298116,

RESULT

This algorithm and deployment strategy ensure an **accurate, scalable, and real-time fault detection system** integrated seamlessly with IBM Cloud's infrastructure for **power system reliability and operational efficiency**.

CONCLUSION

In this project, a **machine learning-based approach** was successfully designed and implemented for detecting and classifying various types of faults in power distribution systems. Using electrical measurement data such as voltage and current phasors, the developed model accurately distinguishes between **normal operating conditions and faults like line-to-ground, line-to-line, and three-phase faults**.

By leveraging **IBM Cloud services**, including **watsonx.ai for model development and runtime execution** and **IBM Cloud Object Storage for secure data handling**, the solution ensures **efficient, scalable, and reliable fault detection**. This enhances the operational stability and safety of power systems, enabling **rapid response and maintenance actions to prevent extended outages or equipment damage**.

Overall, the project demonstrates the potential of **cloud-integrated AI solutions** in modernising power system monitoring and protection strategies.

FUTURE SCOPE

- Integrate with **IoT and smart grids** for real-time monitoring
- Use **deep learning models** for improved accuracy
- Expand with **larger, diverse datasets** for robustness
- Deploy as **Edge AI solutions** for low-latency detection
- Develop models for **fault location identification**
- Integrate with **automated protection systems** for rapid isolation

REFERENCES

[1] B. Gou, D. Lubkeman, and R. Jones, "Automated fault location on distribution feeders using synchronized voltage phasors," *IEEE Transactions on Power Delivery*, vol. 21, no. 1, pp. 348–353, Jan. 2006.

[2] S. Ten, S. M. Brahma, and R. Agrawal, "Fault classification and section identification in power distribution systems using smart sensors," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 108–117, Mar. 2011.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009.

[4] IBM Cloud Documentation, "Using watsonx.ai for Machine Learning Model Development." [Online]. Available: https://cloud.ibm.com/docs/watsonx-ai. [Accessed: Jul. 24, 2025].

[5] J. J. Grainger and W. D. Stevenson, *Power System Analysis*. New York, NY, USA: McGraw Hill, 1994.

[6] Scikit-learn Developers, "Scikit-learn: Machine Learning in Python." [Online]. Available: https://scikit-learn.org/stable/. [Accessed: Jul. 24, 2025].

IBM CERTIFICATION

IBM **SkillsBuild** Completion Certificate

This certificate is presented to

Thivya Priya G

for the completion of

Lab: Retrieval Augmented Generation with LangChain

(ALM-COURSE_3824998)

According to the Adobe Learning Manager system of record

Completion date: 23 Jul 2025 (GMT)

Learning hours: 20 mins

In recognition of the commitment to achieve professional excellence

THIVYA PRIYA G

Has successfully satisfied the requirements for:

Getting Started with Artificial Intelligence

Issued on: Jul 15, 2025
Issued by: IBM SkillsBuild

Verify: https://www.credly.com/badges/c7bf7ed4-9106-401e-8bdf-903b861fae14


In recognition of the commitment to achieve professional excellence

Journey to Cloud: Envisioning Your Solution

IBM SkillsBuild

IBM

THIVYA PRIYA G

Has successfully satisfied the requirements for:

Journey to Cloud: Envisioning Your Solution

Issued on: Jul 16, 2025

Issued by: IBM SkillsBuild

Verify: https://www.credly.com/badges/7d6acb83-b5f1-41f4-b984-41e54ce136d2

IBM

OTHER REFERENCES:

www.linkedin.com/in/thivya-priya-g-970360239