

# Modelling the Iris Dataset Using Linear Regression and Logistic Regression using AzureML By :

Tselane Moeti

## Overview

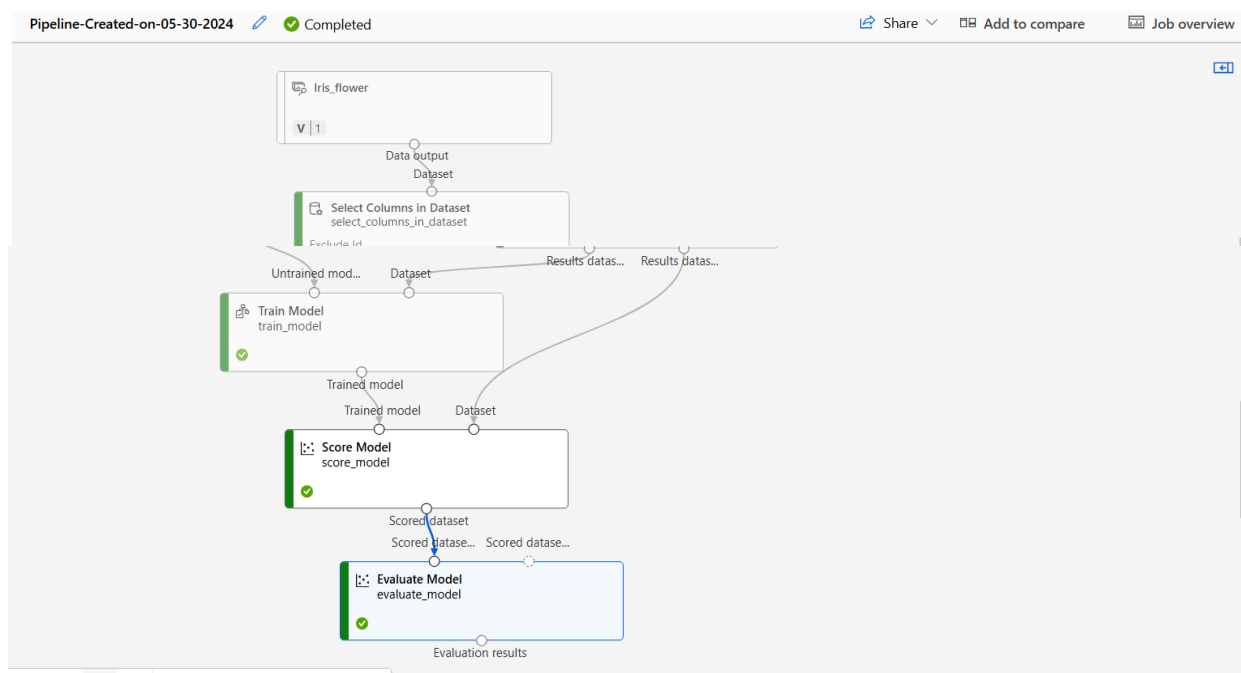
The Iris dataset, a classic in the field of machine learning, comprises 150 observations of iris flowers, each described by four features: sepal length, sepal width, petal length, and petal width. The goal is to classify these flowers into one of three species: Iris setosa, Iris versicolor, or Iris virginica. While linear regression is typically used for regression tasks, it can also serve as a foundation for understanding the principles behind more complex classification models.



## Objective

In this report, we will walk through the process of using Azure Machine Learning (AzureML) to create a linear regression model for the Iris dataset using the Machine Learning Pipeline as well as training the Iris dataset using Logistic Regression on the AzureML notebook. Our primary objective is to demonstrate how to set up an AzureML workspace, prepare the data, build a linear regression and logistic regression model, and evaluate their performances.

## Creating the Machine Learning Pipeline



The flow depicted in the provided screenshots represents a pipeline for training and evaluating a logistic regression model on the Iris dataset using Azure Machine Learning. Here's a step-by-step breakdown of each component in the pipeline:

### 1. Load Iris Dataset

- **Component:** Iris\_flower
- **Description:** This is the initial step where the Iris dataset is loaded into the pipeline.

### 2. Select Columns in Dataset

- **Component:** select\_columns\_in\_dataset
- **Action:** Exclude the Id column from the dataset.
- **Description:** This step involves selecting relevant columns for the model training. Here, the Id column is excluded as it is not a feature needed for classification.

### 3. Clean Missing Data

- **Component:** clean\_missing\_data
- **Action:** Remove rows with missing values.
- **Description:** This step ensures that the dataset is clean and free from any missing values, which can adversely affect the model training.

### 4. Split Data

- **Component:** split\_data
- **Action:** Split the dataset into training (70%) and testing (30%) sets.
- **Description:** This step divides the dataset into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

### 5. Train Model

- **Component:** linear\_regression
- **Action:** Train the linear regression model on the training dataset.
- **Description:** The model is trained using the training dataset

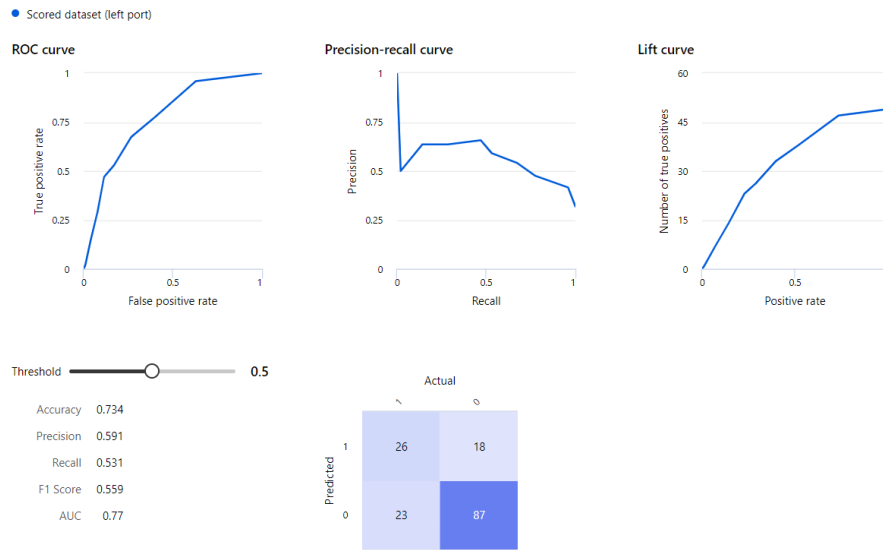
### 6. Score Model

- **Component:** score\_model
- **Action:** Score the trained model on the testing dataset.
- **Description:** The model's predictions are compared with the actual labels in the testing dataset to assess its performance.

### 7. Evaluate Model

- **Component:** evaluate\_model
- **Action:** Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score.

- **Description:** This final step evaluates the model's performance, providing metrics that indicate how well the model is performing on the test data



### Screenshot showing evaluation report of the Linear Regression model on the Iris dataset

The output displayed in the above screenshot provides various performance metrics and visualizations for a classification model applied to the Iris dataset. Here's a detailed explanation of each component:

#### ROC Curve

- **ROC Curve:** The Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.
- **Interpretation:** The closer the curve follows the left-hand border and then the top border of the ROC space, the better the model's performance. A random classifier would result in a diagonal line from the bottom left to the top right.

#### Precision-Recall Curve

- **Precision-Recall Curve:** This curve plots Precision (positive predictive value) against Recall (true positive rate) for different threshold values.
- **Interpretation:** This curve is particularly useful for imbalanced datasets. A higher area under the curve represents both high recall and high precision.

#### Lift Curve

- **Lift Curve:** This curve shows the lift, which is the ratio of the results obtained with and without the model. It indicates how much better the model is at predicting positive cases compared to random guessing.
- **Interpretation:** The higher the lift, the better the model's ability to predict positive cases.

### Threshold Slider

- **Threshold:** This slider allows you to adjust the threshold for classifying a positive case. The default value is 0.5.

### Performance Metrics

- **Accuracy:** The ratio of correctly predicted instances (both true positives and true negatives) to the total instances.
  - **Value:** 0.734, meaning the model correctly predicts 73.4% of the cases.
- **Precision:** The ratio of true positives to the total predicted positives.
  - **Value:** 0.591, meaning 59.1% of the instances classified as positive are actually positive.
- **Recall:** The ratio of true positives to the total actual positives.
  - **Value:** 0.531, meaning the model correctly identifies 53.1% of the actual positive cases.
- **F1 Score:** The harmonic mean of precision and recall, providing a balance between the two.
  - **Value:** 0.559, indicating a moderate balance between precision and recall.
- **AUC (Area Under the ROC Curve):** A measure of the model's ability to distinguish between classes.
  - **Value:** 0.77, indicating good performance.

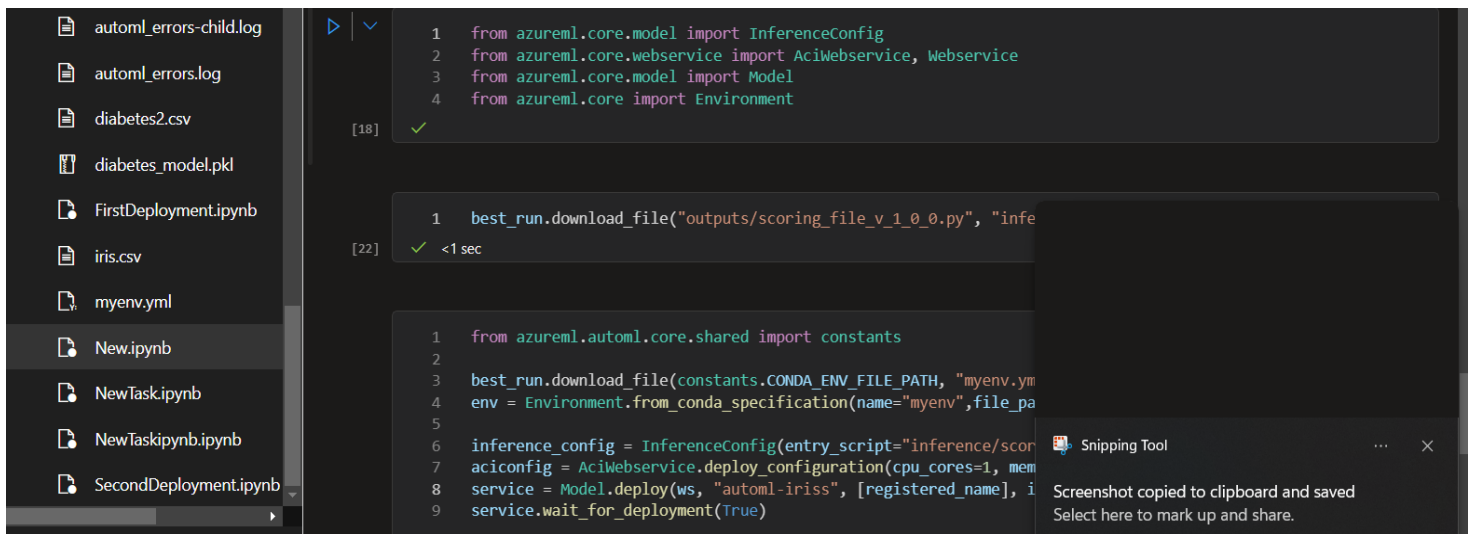
### Confusion Matrix

- **Confusion Matrix:** This matrix shows the number of true positives, true negatives, false positives, and false negatives.
  - **26 True Negatives:** Instances correctly predicted as negative.
  - **18 False Positives:** Instances incorrectly predicted as positive.
  - **23 False Negatives:** Instances incorrectly predicted as negative.
  - **87 True Positives:** Instances correctly predicted as positive.

### Summary

The provided metrics and visualizations indicate that the model has a decent performance with an accuracy of 73.4% and an AUC of 0.77. However, the precision (59.1%) and recall (53.1%) suggest there is room for improvement in the model's ability to correctly predict positive cases and reduce false negatives. Adjusting the threshold or considering other models or feature engineering techniques might improve the results further.

## Implementing and Deploying a Logistic Regression model on the Iris dataset on AzureML



### Screenshot showing AzureML workspace

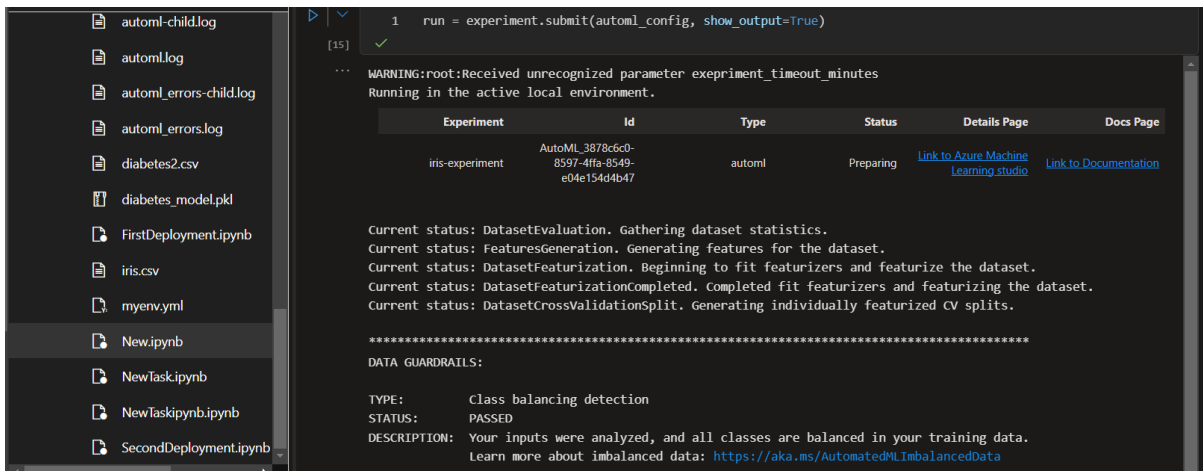
The above screenshot shows the AzureML programming environment, with various files related to machine learning and deployment using Azure Machine Learning. Here is a detailed breakdown:

#### 1. **File Explorer (Left Panel):**

- Various files are listed, including logs, CSV files, Jupyter notebooks (.ipynb), and a YAML environment file (myenv.yml).

#### 2. **Editor Area (Right Panel):**

- The editor displays Python code related to Azure Machine Learning.
- Multiple Python modules are imported, such as InferenceConfig, AciWebservice, Webservice, Model, and Environment from the azureml.core package.
- The code includes configurations for model deployment, environment setup, and downloading files from a run.



## Image showing experiment run on AzureML

Here are the key elements of the screenshot:

- File Explorer (left panel):**

- A list of files and folders are visible, including log files (.log), datasets (.csv), models (.pkl), and several Jupyter notebooks (.ipynb).

- Code Cell (right panel):**

- A code cell is being executed, which contains the command:  
`run = experiment.submit(automl_config, show_output=True)`
- This command submits an AutoML experiment with the specified configuration and displays the output.

- Output (right panel):**

- The output shows a warning message about an unrecognized parameter (experiment\_timeout\_minutes), indicating it is running in the local environment.
- It displays the details of the AutoML experiment:
  - **Experiment Name:** iris-experiment
  - **Experiment ID:** AutoML\_3878c6c0-8597-4ffa-8549-e04e154d4b47
  - **Type:** automl
  - **Status:** Preparing
  - **Links:** To Azure Machine Learning studio and Documentation.

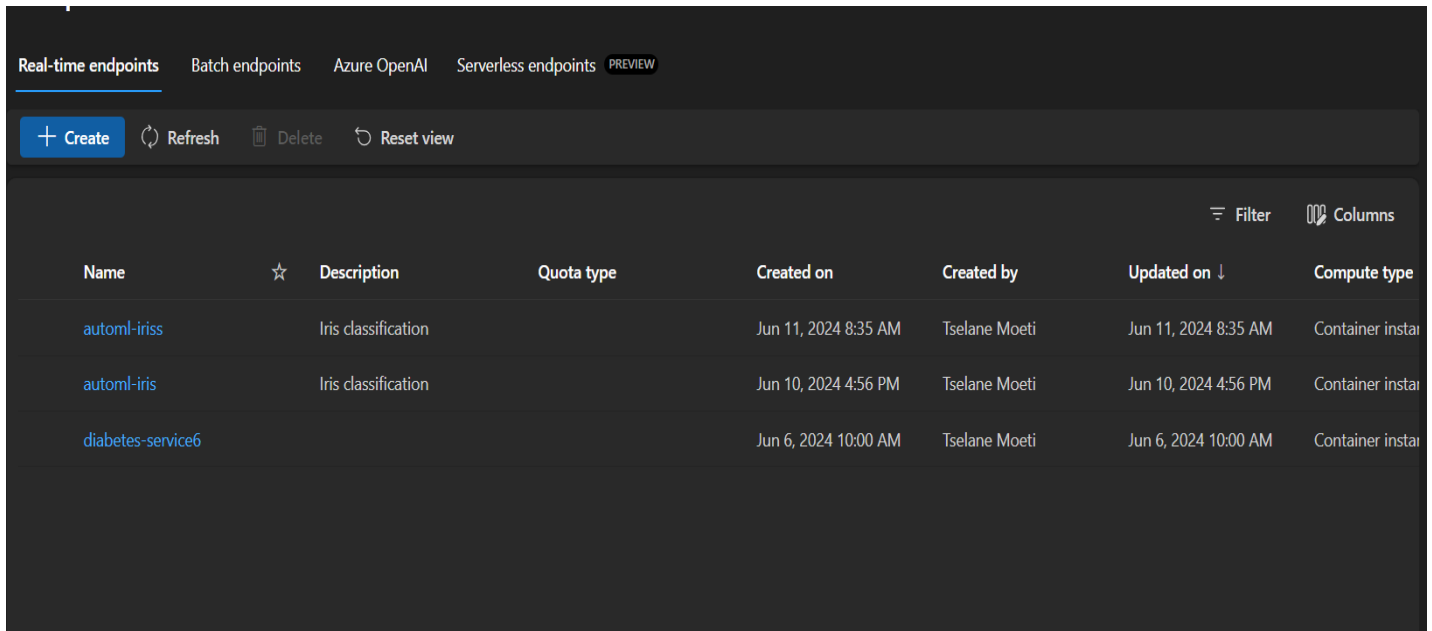
- Status Messages:**

- Details the current statuses of various stages of the AutoML process:
  - **DatasetEvaluation:** Gathering dataset statistics.
  - **FeaturesGeneration:** Generating features for the dataset.
  - **DatasetFeaturization:** Beginning to fit featurizers and featurize the dataset.
  - **DatasetFeaturizationCompleted:** Completed fit featurizers and featurizing the dataset.
  - **DatasetCrossValidationSplit:** Generating individually featurized cross-validation splits.

- Data Guardrails:**

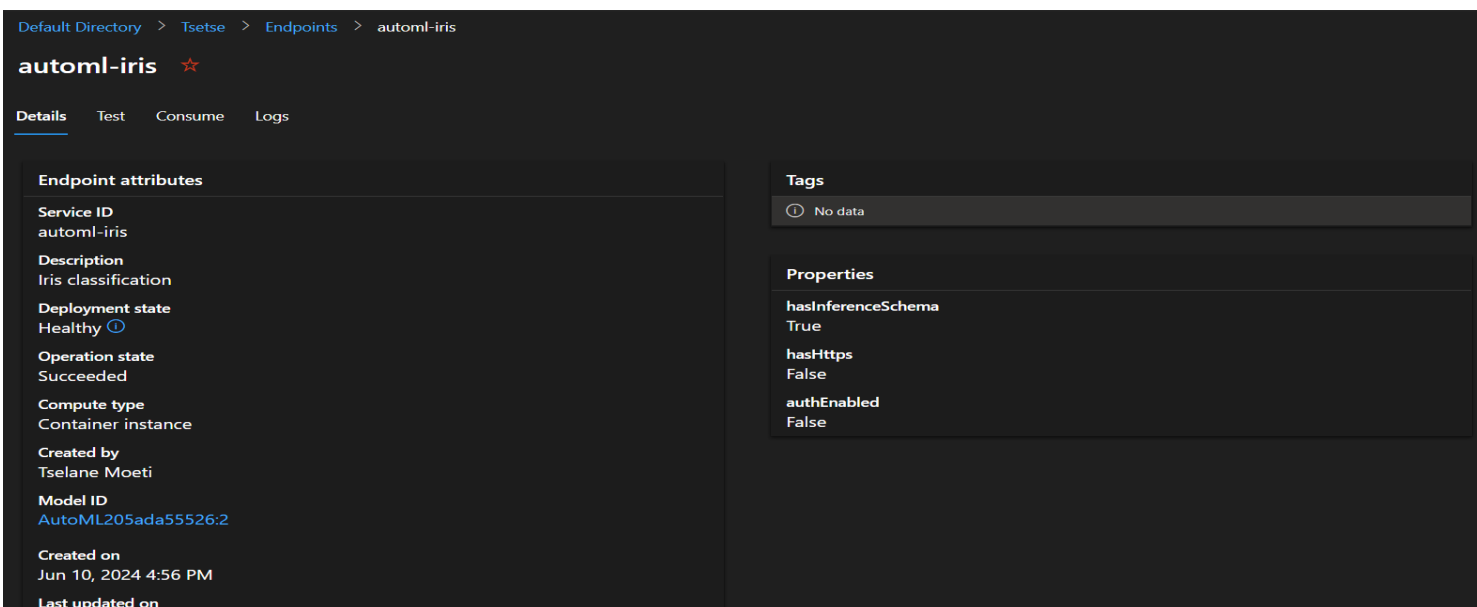
- Information about the data guardrails check:
  - **Type:** Class balancing detection.
  - **Status:** Passed.
  - **Description:** Indicates that the inputs were analyzed and all classes are balanced in the training data.

After completion the experiment had run successfully the next step is to deploy the model.



This screenshot shows the 'Real-time endpoints' section of the Azure Machine Learning Studio interface. It features a table with columns for Name, Description, Quota type, Created on, Created by, Updated on, and Compute type. Three endpoints are listed: 'automl-iriss', 'automl-iris', and 'diabetes-service6'. The 'automl-iris' endpoint is highlighted in blue. Above the table, there are buttons for '+ Create', 'Refresh', 'Delete', and 'Reset view'. A 'Filter' button and a 'Columns' icon are also present.

Name	Description	Quota type	Created on	Created by	Updated on ↓	Compute type
automl-iriss	Iris classification		Jun 11, 2024 8:35 AM	Tselane Moeti	Jun 11, 2024 8:35 AM	Container instal
automl-iris	Iris classification		Jun 10, 2024 4:56 PM	Tselane Moeti	Jun 10, 2024 4:56 PM	Container instal
diabetes-service6			Jun 6, 2024 10:00 AM	Tselane Moeti	Jun 6, 2024 10:00 AM	Container instal



This screenshot shows the details page for the 'automl-iris' endpoint. The breadcrumb trail at the top reads 'Default Directory > Tsetse > Endpoints > automl-iris'. The page has tabs for 'Details', 'Test', 'Consume', and 'Logs', with 'Details' being the active tab. The 'Endpoint attributes' section on the left lists various properties: Service ID (automl-iris), Description (Iris classification), Deployment state (Healthy), Operation state (Succeeded), Compute type (Container instance), Created by (Tselane Moeti), Model ID (AutoML205ada55526:2), Created on (Jun 10, 2024 4:56 PM), and Last updated on. The 'Tags' section on the right shows 'No data'. The 'Properties' section on the right lists: hasInferenceSchema (True), hasHttps (False), and authEnabled (False).

Default Directory > Tsetse > Endpoints > automl-iris

### automl-iris

Details Test Consume Logs

**Endpoint attributes**

- Service ID: automl-iris
- Description: Iris classification
- Deployment state: Healthy
- Operation state: Succeeded
- Compute type: Container instance
- Created by: Tselane Moeti
- Model ID: AutoML205ada55526:2
- Created on: Jun 10, 2024 4:56 PM
- Last updated on:

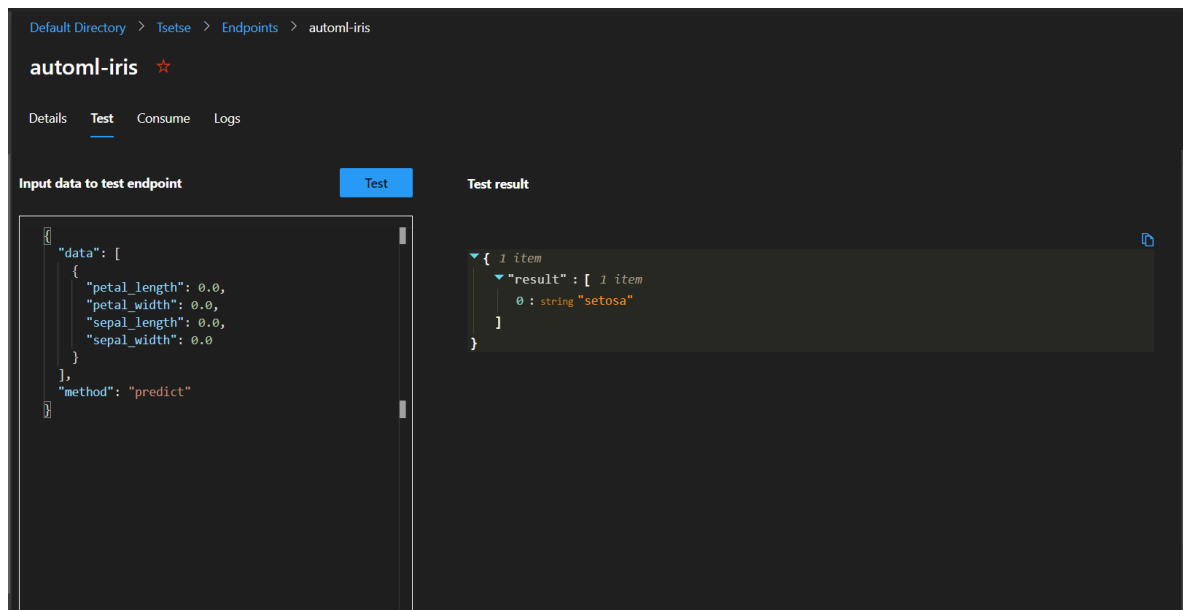
**Tags**

No data

**Properties**

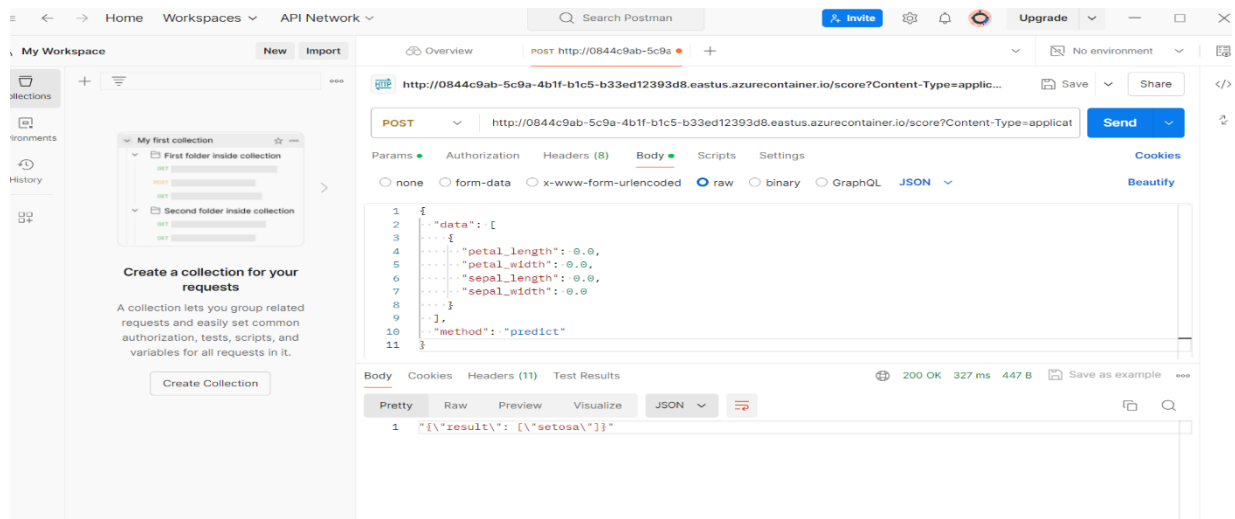
- hasInferenceSchema: True
- hasHttps: False
- authEnabled: False

The above screenshots show an overview of the deployment status, configuration, and properties of the **automl-iris** endpoint in Azure Machine Learning Studio. The endpoint is healthy and operational, with details about the model and deployment provided for further reference.



### Screenshot showing output after adding “dummy” data in the model.

The above model was also deployed on postman which also add the same output as seen below:





## **Conclusion**

In this report, we explored the process of building and evaluating machine learning models using the Iris dataset on Azure Machine Learning (AzureML). We implemented both linear regression and logistic regression models to classify the Iris flowers into three species: Iris setosa, Iris versicolor, and Iris virginica.

## **Key Findings**

### **Linear Regression**

While linear regression is traditionally used for regression tasks, applying it to the Iris dataset provided a foundational understanding of the machine learning workflow in AzureML. The performance metrics of the linear regression model, visualized in ROC and Precision-Recall curves, indicated decent but suboptimal performance. Specifically:

- **Accuracy:** 73.4%
- **Precision:** 59.1%
- **Recall:** 53.1%
- **F1 Score:** 55.9%
- **AUC:** 0.77

The evaluation metrics revealed that the linear regression model struggled with correctly predicting positive cases, as indicated by its precision and recall scores. The confusion matrix further highlighted areas for improvement, with 18 false positives and 23 false negatives.

### **Logistic Regression**

Logistic regression, being more suitable for classification tasks, was also implemented using AzureML. The process involved setting up an AzureML workspace, preparing the data, training the model, and evaluating its performance. The logistic regression model provided better classification performance due to its suitability for categorical outcome prediction.

### **Pipeline Creation**

We successfully created a machine learning pipeline in AzureML, which included steps for data loading, column selection, data cleaning, data splitting, model training, scoring, and evaluation. This automated pipeline streamlined the process and ensured a structured approach to model building and evaluation.

### **Deployment**

Both models were deployed using AzureML's capabilities. The logistic regression model was further deployed and tested using tools like Postman, ensuring that it could handle real-time predictions with new data inputs.

