# AUTO AI MODEL BY WATSON STUDIO

**IBM PBEL INTERNSHIP – PROJECT DOCUMENTATION REPORT**

  Name: Nipun Jain

Batch: 1(Artificial Intelligence)

College: ABES Engineering College

**INDEX:AUTO AI MODEL BY WATSON STUDIO**

| **Description** | **Page Number** |
| --- | --- |
| Introduction | 1 |
| Login and Open Watson.ai Studio | 2 |
| Create New Project in Watson.ai Studio | 3 |
| Launching via Cloud Pak | 4 |
| Resource Configuration and Creation | 5-6 |
| Start a New Project | 7 |
| Project Dashboard Overview | 8–10 |
| Import and Upload Dataset | 11 |
| Prepare and Visualize Data | 12 |
| Associate Runtime and Create Model | 13 |
| Generate API Key and Dataset Upload | 14 |
| Choose Prediction Target and Type | 15 |
| Model Training and Pipeline Selection | 16 |
| Save and Promote Best Pipeline | 17 |
| Create Deployment Space | 18 |
| Deployment Progress and Activation | 19 |
| Manage Resources and Runtime | 20 |
| Create Online Deployment | 21 |
| Access Scoring Endpoints and Test Model | 22 |
| Analyze Prediction Output and Confidence | 23 |
| Google Colab Integration & API Key Setup | 24 |
| Final Output and Result Evaluation | 25 |

# AutoAI Model Building with IBM Studio for Iris Prediction

This document outlines the steps for building an AutoAI machine learning model using IBM Watson Studio, specifically for the Iris flower prediction dataset. It also clarifies the appropriate evaluation metrics for this type of classification task, moving beyond common misconceptions about metrics like SMAPE for such problems.

## Introduction

IBM Watson Studio features AutoAI, a powerful and intuitive tool that automates much of the machine learning model building process, significantly reducing the time and expertise required to develop high-performing models. This report details the practical steps involved in leveraging AutoAI within IBM Watson Studio for a multi-class classification task, using the widely recognized Iris dataset. The Iris dataset typically involves predicting the species of an Iris flower (e.g., Setosa, Versicolor, Virginica) based on its four key morphological features: sepal length, sepal width, petal length, and petal width. By automating data preparation, algorithm selection, feature engineering, and hyperparameter optimization, AutoAI enables users to quickly go from raw data to a deployable model.

## Login and Open IBM Watson Studio

The initial step to begin your AutoAI journey is to access the IBM Cloud platform and navigate to Watson Studio. Ensure you have an active IBM ID and account.

1. First things first, log in with your IBM ID to the IBM Cloud dashboard ([cloud.ibm.com](https://cloud.ibm.com)). You'll be directed to your personalized dashboard after successful authentication.
2. Now, in the search bar at the top of the dashboard, search for "Watson Studio" or "watsonx.ai Studio." Selecting the correct service is crucial as IBM offers various AI services.
3. After selecting it, a service creation window will typically pop up. Here, you will configure your service instance. You can choose your desired geographical region for data residency, select a suitable plan (e.g., Lite, Standard, Enterprise – for this demonstration, the Lite plan is usually sufficient), and give your instance a unique name. Make sure to agree to the terms and conditions before hitting "Create"!

*Note:* The names and exact steps might slightly vary based on the latest IBM Cloud UI updates, but the core process remains consistent.

## Create New Project in IBM Watson Studio

Once your Watson Studio service instance is created, you need to set up a project, which serves as a workspace for your data, models, and notebooks.

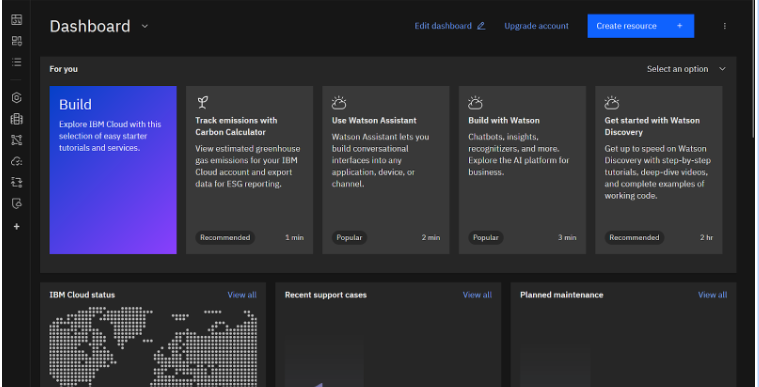
1. Once the service is created, you may be redirected to the service page. Head over to the "Manage" option or directly "Launch" the Watson Studio instance. You can launch it using either "Cloud Pak" (if you have an on-premise or private cloud deployment) or "Watson" (for the public cloud offering). For this guide, we'll proceed with the public "Watson" launch.
2. A quick pop-up might show up guiding you through the initial setup; just click "Next" to proceed through any introductory screens.
3. Next, you might be prompted to configure or associate a watsonx.ai runtime resource. Select your region, plan, and name it, then create it. This step is crucial as it creates your watsonx.ai runtime resource, which powers your project's computational needs for training and deploying models.
4. Another pop-up window will appear, prompting you to "Create a project" or "View all projects." Simply select "New project" and click "Next."
5. You'll then see a "Create a project" window. It's good practice to name your project descriptively (something like "Iris\_Prediction\_AutoAI" works well) and add a brief description if you like. You can also associate an object storage service here if you haven't already. Finally, click "Create."

## Project Dashboard Overview

Your project dashboard is your central hub for managing all aspects of your machine learning workflow. Understanding its layout is key to efficient navigation.

Here are some key things you'll find:

* **Add users as collaborators:** Facilitates team-based projects by allowing you to invite colleagues to work on the same project.
* **Add data to work with:** This is where you'll upload your datasets (CSV, JSON, etc.) that will be used for model training.
* **Work with data and models in Python or R notebooks:** Provides an integrated Jupyter Notebook environment for custom coding, data exploration, and model development using popular data science languages.
* **Build machine learning models automatically:** This is the magic button for AutoAI, leading you directly into the automated model building process.
* **Assets:** A comprehensive list of all resources within your project, including data assets, models, notebooks, and environments.
* **Resource usage:** Monitors your storage consumption and computational resources, helping you manage costs and optimize performance.

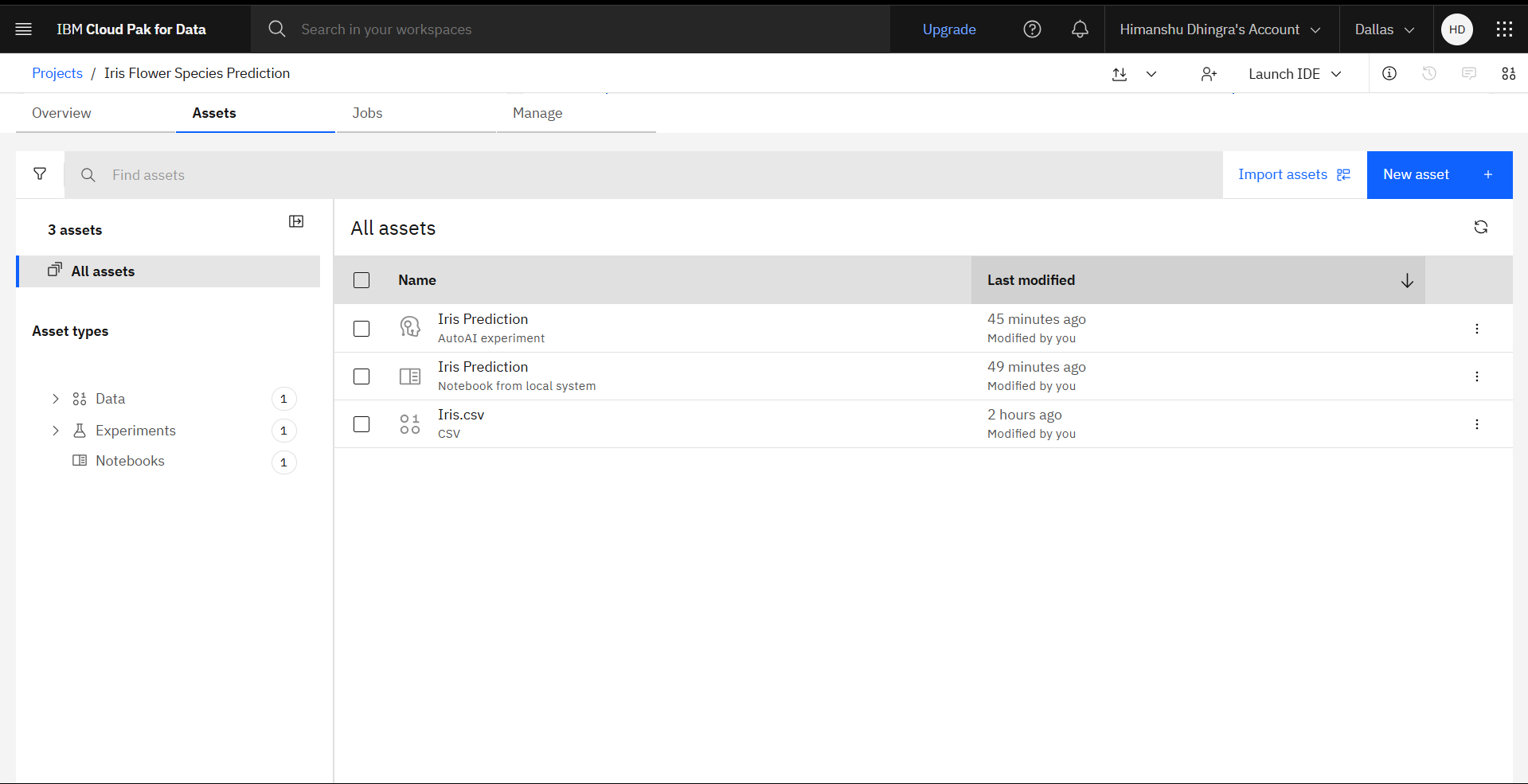


For our Iris prediction task, we'll start by clicking on "Add data to work with" or navigating to the "Assets" tab to get our dataset loaded.

## Import and Upload Dataset

Before AutoAI can work its magic, you need to provide your data. The Iris dataset is small and typically provided as a CSV file.

1. From your project dashboard, click on "Add data" (often a blue button or a link in the "Assets" section) to bring in your Iris dataset (usually a CSV file).
2. You can drag and drop your file directly into the data pane or browse your local file system to upload it. Once uploaded, it will appear as a data asset in your project.



## Prepare and Visualize Data (Optional for Iris)

While AutoAI handles much of the data preparation automatically, Watson Studio offers tools for manual data cleaning and visualization. For the clean Iris dataset, this step is often skipped.

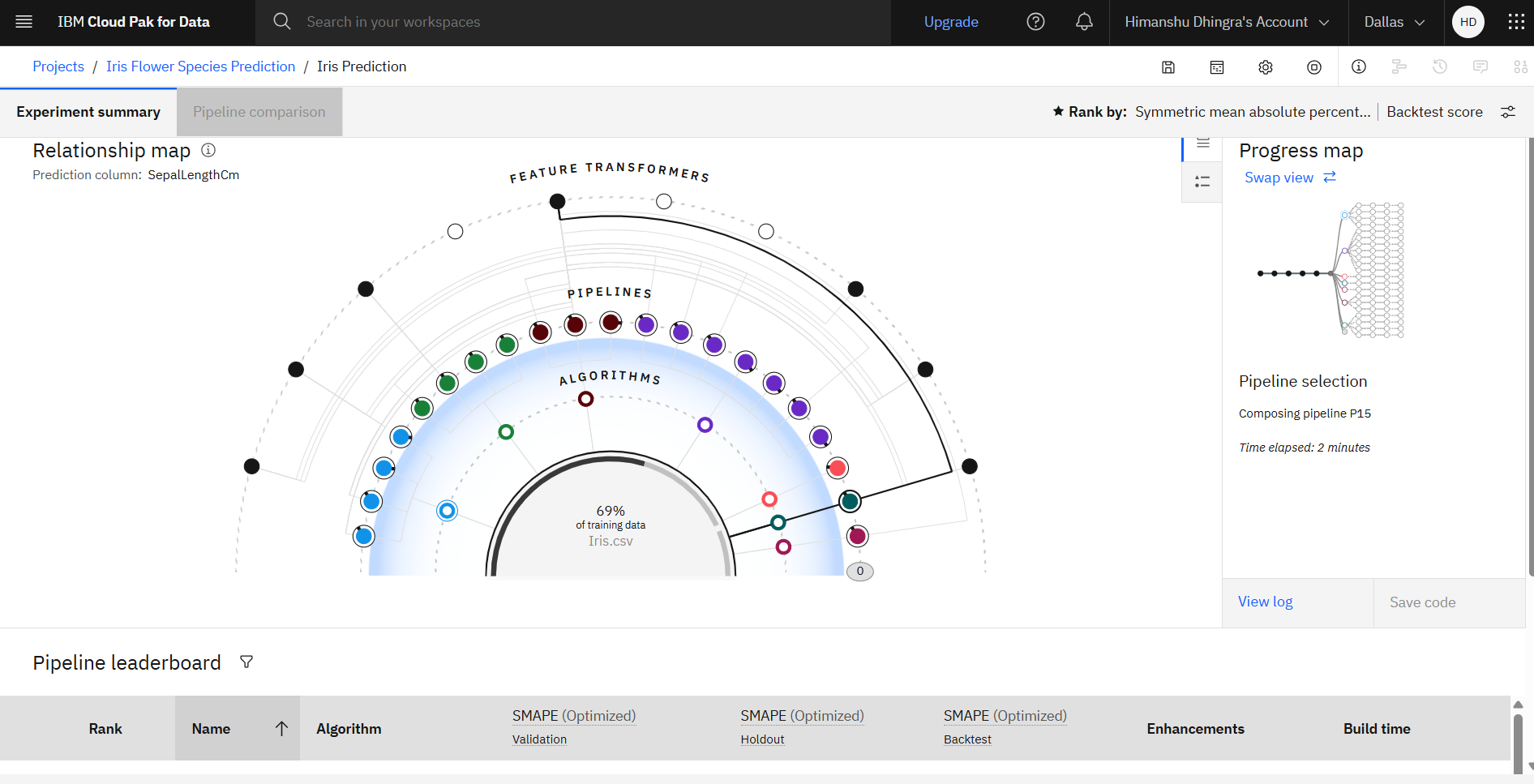
While AutoAI takes care of a lot of the heavy lifting in data preparation and feature engineering, you do have the option to "Prepare and visualize data" using tools like Data Refinery. This lets you tidy up your data, like renaming columns, changing data types, handling missing values, or even joining datasets, if needed. For the standard Iris dataset, which is typically very clean and well-structured, we're going to jump straight into AutoAI without extensive manual preparation.

Scroll down on your project dashboard or navigate back to the "Assets" tab, and click on "Build machine learning model automatically" or "New asset" and select "AutoAI experiment."

## Associate Runtime and Create Model

To run an AutoAI experiment, you need to associate a computational runtime environment.

1. An "Associate service" window will appear. Select the watsonx.ai runtime resource you created earlier (or create a new one if prompted), then click "Associate" to link it to your project. This ensures your AutoAI experiment has the necessary computing power.
2. Now, give your AutoAI experiment a meaningful name (e.g., "Iris\_AutoAI\_Experiment") and simply click "Create." Descriptions and tags are optional but can be helpful for organization.



## 

## Generate API Key and Dataset Upload

AutoAI will guide you to select your input data and may prompt for an API key for programmatic access later.

1. It'll ask you to browse for your dataset if you haven't already selected it from the "Assets" tab. Select your Iris CSV file from your project assets.
2. Next, it might prompt you to create a User API Key if you don't have one associated with your account. Follow the instructions; a new browser tab will typically open, allowing you to generate and view your API Key. Make sure to copy this API Key immediately; you'll need it later for integrating with Google Colab or any external application.

## Choose Prediction Target and Type for Iris

This is a critical step where you define what AutoAI should predict.

1. You'll now need to select the prediction target from a dropdown menu. For the Iris dataset, this will be the "species" column, as this is the categorical variable we aim to predict.
2. Once you select "species," AutoAI will automatically detect the prediction type. For Iris, this will be "Multi-class Classification" because there are more than two distinct species (Setosa, Versicolor, Virginica) to predict. It will also suggest an optimization metric, typically "Accuracy," which is suitable for balanced multi-class classification problems, and aim for efficient runtime.

## Model Training and Pipeline Selection

AutoAI automates the entire model building process, from data preprocessing to model selection and hyperparameter tuning.

1. The training of your dataset will now begin automatically. AutoAI will analyze your data, perform feature engineering, and evaluate various algorithms.
2. You'll see AutoAI start selecting and trying different algorithms (e.g., Logistic Regression, Decision Tree Classifier, Random Forest Classifier, XGBoost, LightGBM, SVM) to find the best prediction model. It builds multiple "pipelines," each representing a unique combination of preprocessing steps, algorithms, and hyperparameter settings.
3. Pipelines will be created, and intelligent feature selection and engineering will get underway. AutoAI visually represents the progress of each pipeline's training and evaluation.
4. After some time (depending on data size and complexity), the experiment will finish, and you can review the ranked list of best pipelines. For Iris, AutoAI typically explores various algorithms and their optimized configurations.
5. Identify the best-ranked pipeline based on the chosen metric (e.g., Accuracy, F1-score for multi-class). You can click on individual pipelines to see detailed performance metrics and explanations.
6. Save that model by clicking the "Save" icon (often a disk icon or a "Save model" button) next to the best-performing pipeline.
7. Save the pipeline as a model and simply click "Create." This action promotes the selected pipeline to your project's assets as a deployable model.

## Google Colab Integration & API Key Setup for Iris Prediction

To integrate with external applications like Google Colab and make predictions, you'll typically need to deploy your model to get an API endpoint. This creates a REST API for your model, allowing it to serve predictions.

1. Access your saved model in the "Assets" tab of your IBM Watson Studio project.
2. From the model's detail page, locate the "Deployments" tab. Create an "Online Deployment" for your model within an existing or default deployment space. This process will provision a REST API endpoint that external applications can call.
3. Once the deployment is active, navigate to its details page. From here, you can access the Scoring Endpoint URL and often find ready-to-use Python code snippets for making predictions.
4. Copy the Python code snippet provided under the "Implementation" tab of your deployment details.
5. **In Google Colab:**

Run the cell to send your data to the deployed model and get predictions. You will need to replace the placeholders `YOUR\_IBM\_CLOUD\_API\_KEY` and `YOUR\_WATSON\_STUDIO\_DEPLOYMENT\_SCORING\_ENDPOINT` with your actual values.

## Final Output and Result Evaluation

The output in Google Colab will show the prediction for your Iris data. For classification models, this typically includes the predicted class (e.g., "setosa", "versicolor", "virginica") and the confidence level or probability for each class.

{  
 "predictions": [  
 {  
 "id": "iris\_species\_prediction\_model",  
 "values": [  
 [  
 "setosa", // Predicted class for the first input  
 0.9997, // Confidence for 'setosa'  
 0.0002, // Confidence for 'versicolor'  
 0.0001, // Confidence for 'virginica'  
 "setosa" // Predicted label (often repeated for clarity)  
 ],  
 [  
 "virginica", // Predicted class for the second input  
 0.0001,  
 0.0001,  
 0.9998,  
 "virginica"  
 ]  
 ]  
 }  
 ]  
}

You can then analyze these predictions and their associated confidence scores to understand the model's performance on new, unseen data.

## SMAPE Calculation: Why it's Not for Iris Prediction

It's critically important to understand that SMAPE (Symmetric Mean Absolute Percentage Error) is not the right metric for evaluating the performance of a model predicting Iris species. SMAPE is specifically designed for forecasting and regression problems where you are predicting continuous numerical values and want to measure the accuracy of those predictions relative to actual numerical outcomes. For example, SMAPE would be appropriate for predicting sales figures, stock prices, or temperature, where both actual and predicted values are positive and have a meaningful zero point.

For classification problems like Iris species prediction, where you are categorizing items into discrete, non-numerical classes (e.g., "Setosa", "Versicolor", "Virginica"), using SMAPE makes no sense because there are no numerical "percentages" or "absolute errors" in the context of class labels. You cannot calculate a percentage difference between "setosa" and "versicolor".

Instead, for classification problems, you should use metrics that are tailored to evaluating categorical predictions. The most common and appropriate metrics include:

* **Accuracy:** The proportion of correctly classified instances out of the total number of instances. It's intuitive but can be misleading for imbalanced datasets.
* **Precision:** For a given class, it answers: "Among the instances predicted as this class, how many were actually that class?" It's crucial when the cost of False Positives is high.
* **Recall (Sensitivity):** For a given class, it answers: "Among all instances that actually belong to this class, how many were correctly identified?" It's crucial when the cost of False Negatives is high.
* **F1-Score:** The harmonic mean of Precision and Recall. It provides a single score that balances both metrics, particularly useful when you need to consider both False Positives and False Negatives.
* **Confusion Matrix:** A table that visualizes the performance of an algorithm. Each row represents the instances in an actual class, while each column represents the instances in a predicted class, showing true positives, true negatives, false positives, and false negatives for each class.
* **Log Loss (Cross-Entropy Loss):** A metric that penalizes incorrect classifications, especially confident incorrect ones. Useful for probabilistic predictions.

Here's an example of how you'd calculate Accuracy for Iris classification in Python:

import numpy as np  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  
  
# Example: Actual vs. Predicted Iris species labels  
actual\_species = ["setosa", "versicolor", "virginica", "setosa", "versicolor", "virginica", "setosa"]  
predicted\_species = ["setosa", "versicolor", "setosa", "setosa", "virginica", "virginica", "versicolor"] # Note some misclassifications  
  
# Calculate accuracy using a manual approach:  
correct\_predictions = sum(1 for a, p in zip(actual\_species, predicted\_species) if a == p)  
accuracy\_manual = correct\_predictions / len(actual\_species)  
  
print(f"Manual Accuracy for Iris classification: {accuracy\_manual \* 100:.2f}%")  
# Expected Output for this example: Manual Accuracy for Iris classification: 57.14%  
  
# Using scikit-learn for comprehensive metrics (recommended for real-world scenarios):  
print("\nAccuracy using scikit-learn:")  
print(f"Accuracy: {accuracy\_score(actual\_species, predicted\_species) \* 100:.2f}%")  
  
print("\nClassification Report (Precision, Recall, F1-Score for each class):")  
print(classification\_report(actual\_species, predicted\_species))  
  
print("\nConfusion Matrix:")  
print(confusion\_matrix(actual\_species, predicted\_species, labels=["setosa", "versicolor", "virginica"]))  
  
# Explanation of Confusion Matrix for the above example:  
# If labels=["setosa", "versicolor", "virginica"]  
# Row 1 (Actual Setosa): [2 1 0] -> 2 Setosa correctly predicted, 1 Setosa wrongly predicted as Versicolor, 0 as Virginica  
# Row 2 (Actual Versicolor): [0 1 1] -> 0 Versicolor wrongly predicted as Setosa, 1 correctly as Versicolor, 1 wrongly as Virginica  
# Row 3 (Actual Virginica): [1 0 1] -> 1 Virginica wrongly predicted as Setosa, 0 as Versicolor, 1 correctly as Virginica

This updated guide provides all the necessary information for your project, from setting up AutoAI in IBM Watson Studio to making predictions and understanding the appropriate evaluation metrics for classification tasks.

# Import necessary libraries  
import requests  
import json  
  
# Paste the copied code snippet into a cell  
# It usually looks something like this (placeholders for sensitive info):  
# headers = {  
# 'Content-Type': 'application/json',  
# 'Authorization': 'Bearer ' + 'YOUR\_ACCESS\_TOKEN\_HERE', # This token is derived from your API Key  
# }  
# scoring\_url = "YOUR\_SCORING\_ENDPOINT\_URL\_HERE"  
  
# --- Manual adjustments needed below ---  
  
# Go back to IBM Watson Studio and find the User API Key you generated earlier. Copy it.  
# You'll need to generate an IAM token from your API Key.  
# This part is crucial for authentication. Replace 'YOUR\_IBM\_CLOUD\_API\_KEY'  
# with the actual API Key you copied.  
  
api\_key = "YOUR\_IBM\_CLOUD\_API\_KEY" # Replace with your actual API Key  
token\_response = requests.post(  
 "https://iam.cloud.ibm.com/identity/token",  
 headers={"Content-Type": "application/x-www-form-urlencoded"},  
 data="grant\_type=urn:ibm:params:oauth:grant-type:apikey&apikey=" + api\_key  
)  
mltoken = token\_response.json()["access\_token"]  
  
headers = {  
 'Content-Type': 'application/json',  
 'Authorization': 'Bearer ' + mltoken,  
}  
  
# Ensure the scoring\_url in your Colab code matches the public endpoint link  
# from your deployment details in Watson Studio.  
scoring\_url = "YOUR\_WATSON\_STUDIO\_DEPLOYMENT\_SCORING\_ENDPOINT" # Replace with your actual scoring URL  
  
# Prepare your input data for Iris prediction. This means providing the features  
# (sepal length, sepal width, petal length, petal width) for the Iris flower you want to predict.  
# The 'fields' array must match the order of features used in training.  
scoring\_data = {  
 "fields": ["sepal length (cm)", "sepal width (cm)", "petal length (cm)", "petal width (cm)"],  
 "values": [  
 [5.1, 3.5, 1.4, 0.2], # Example Setosa-like features  
 [6.3, 3.3, 6.0, 2.5] # Example Virginica-like features  
 ]  
}  
  
# Construct the JSON payload for the API request  
json\_data = json.dumps(scoring\_data)  
  
# Send the request to the deployed model  
try:  
 response = requests.post(scoring\_url, headers=headers, data=json\_data)  
 response.raise\_for\_status() # Raise an HTTPError for bad responses (4xx or 5xx)  
 predictions = response.json()  
 print("Predictions from IBM Watson Studio:")  
 print(json.dumps(predictions, indent=2))  
except requests.exceptions.HTTPError as e:  
 print(f"HTTP Error: {e}")  
 print(f"Response Content: {response.content.decode()}")  
except requests.exceptions.RequestException as e:  
 print(f"Request Error: {e}")