Azure Machine Learning

Workshop - Introduction to Azure Machine Learning Studio

March 21, 2025

Table des matières

Introduction to Azure Machine Learning 4
What is Azure Machine Learning studio? 4
Creating a model 5
Code with Python 5
AutoML 6
Designer 6
Compute 7
Managing data 7
Datastores and labeling 7
MLOps 8
How Azure Machine Learning works 8
Deploying machine learning models 9
Lab 00 - Create an Azure Machine Learning Workspace 10
Lab 01 - Create an Azure Machine Learning (Azure ML) service 10
Lab 02 - Create and manage compute resources 16
Create a Compute Instance 16
Create a Compute Cluster 18
Lab 01 - Explore Classification with Azure ML Designer 20
Lab 11 – What is Classification in Machine Learning 20
Lab 12 – Build a Classification model with Azure ML Designer 21
Create a dataset 21
Create a pipeline in Designer and load data to canvas 25
Add transformations 29
Run the pipeline 31
View the transformed data 31
Add training modules 31
Run the training pipeline 33
Add an Evaluate Model module 33
Create an inference pipeline 35
Deploy a service 38
Lab 02 - Automated Machine Learning in Azure ML 39
Lab 21 – What is Automated ML? 39
Definition 39
How does it work? 39
Lab 21 – Use automated machine learning to train a model 39

Train a Regression model with Auto ML 40
Review the best model 45
Deploy the model 46
Lab 3 - Explore Notebooks in Azure ML 47
Clone the Github repository 47
Introduction to the Azure ML SDK 48

Introduction to Azure Machine Learning

Azure Machine Learning is a cloud-based platform designed to manage the entire machine learning life cycle, from training to deployment, without the need for extensive setup.

It supports various types of machine learning, including classical, deep learning, supervised, and unsupervised learning. The platform is structured to help data scientists and ML engineers leverage their existing skills in data processing and model development, supporting languages like Python and R, as well as open-source platforms such as PyTorch and TensorFlow.

Azure Machine Learning offers built-in services like Azure Machine Learning Studio, which provides a user-friendly interface, and Automated Machine Learning capabilities to assist in model selection and training. It is an **agnostic framework** , allowing users to work with their preferred technologies, including TensorFlow, PyTorch, ONNX, MLFlow, and Scikit-learn. The platform centralizes training scripts, models, logs, and computes in a shared workspace, enabling users to focus on model development while Azure Machine Learning handles the rest.

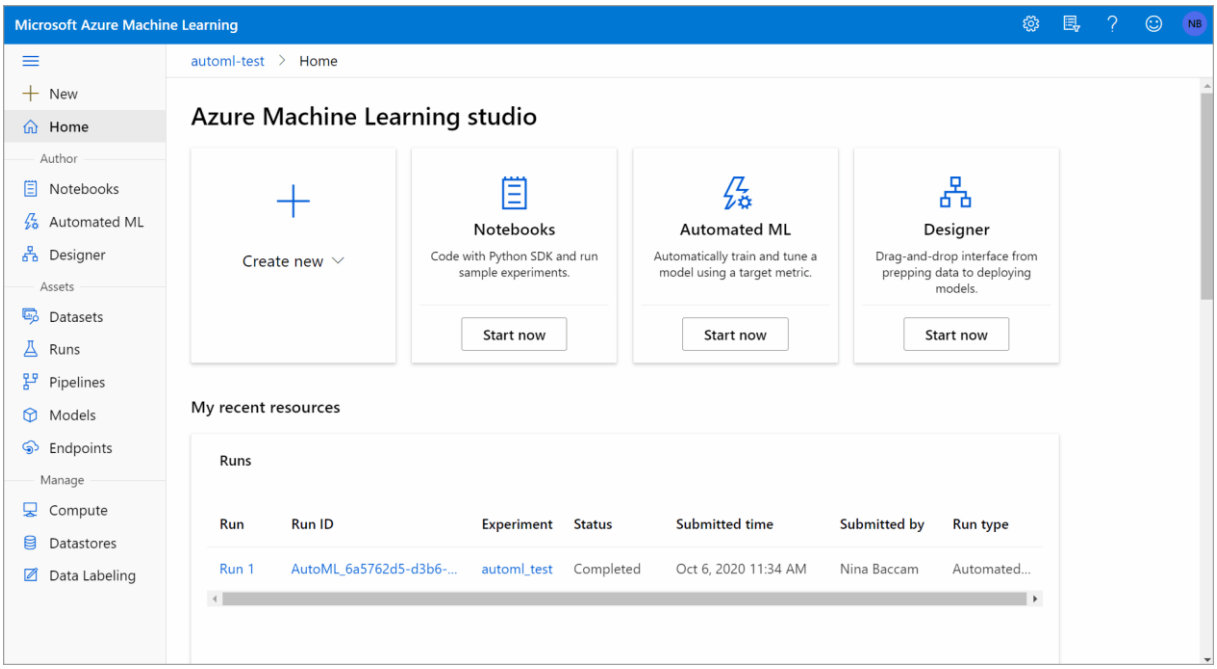
Additionally, Azure Machine Learning includes features for logging, monitoring, and governance of data and models, ensuring that models perform as intended and are effectively communicated to stakeholders.

What is Azure Machine Learning studio?

Azure Machine Learning studio is a browser-based service that provides **no-code and code-first solutions** to visually create, train, and manage models through a web UI.

Azure Machine Learning studio allows the *Python* SDK to seamlessly integrate with the natively supported *Jupyter Notebooks* for collaborative notes and coding. Data within Azure Machine

Learning studio is simple to manage with intuitive data visualization and AI-assisted image or text labeling features.

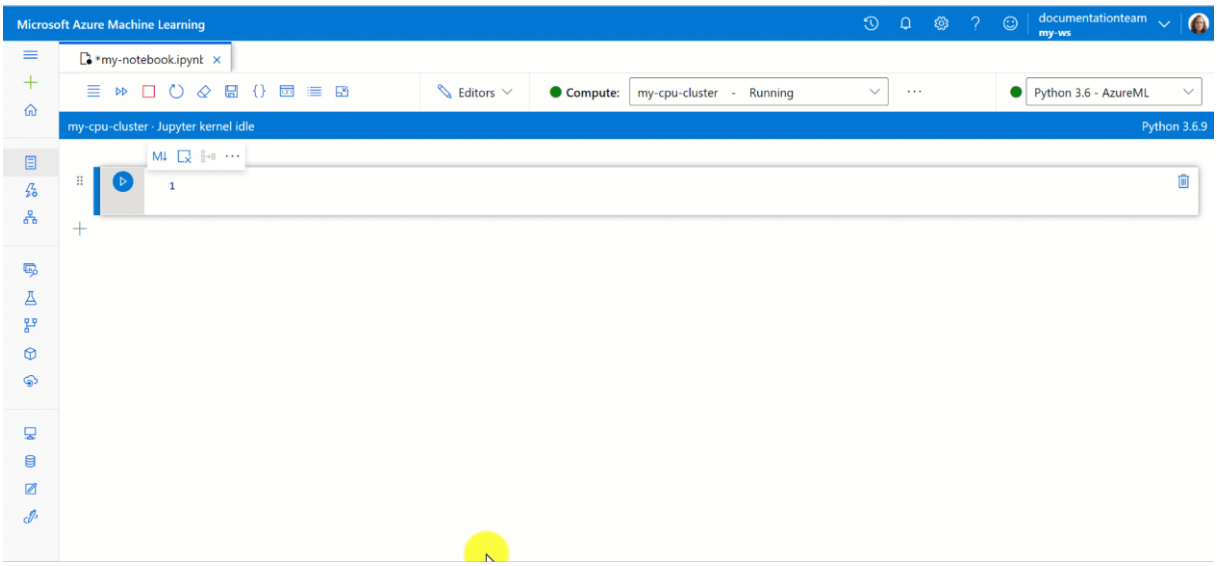


Creating a model

Models can be created in Azure Machine Learning in several ways. Training can take place on a local machine or the Azure cloud, such as a virtual machine or so called compute cluster.

Code with Python

With the *Azure Machine Learning SDK for Python*, you can interact with the service from multiple environments—including Jupyter Notebooks. Notebooks provide a collaborative environment for runnable code, visualizations, and comments. Those Included in studio are sample notebooks you can use to get started with Azure Machine Learning.



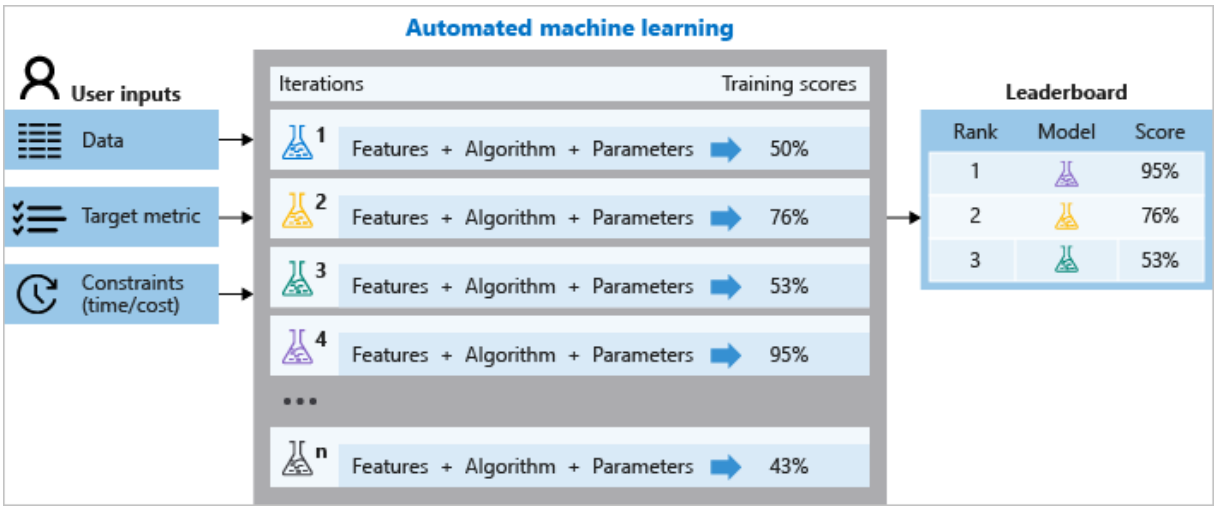
AutoML

Automated Machine Learning (AutoML) automates creating the best machine learning models, helping you find the best model for your data—no matter what your data science expertise is. **Specializing in classification, regression, and time-series forecasting,** AutoML experiments with different features, algorithms, and parameters depending on the task—then provides scores on models it thinks are the best fit. These models can then be deployed as they are, or exported to an ONNX format that can run on various platforms and devices.

AutoML's versatility and speed mean it's often used as a starting point by both experienced and novice data scientists.

You can use AutoML in the Azure Machine Learning studio or through the Python SDK.

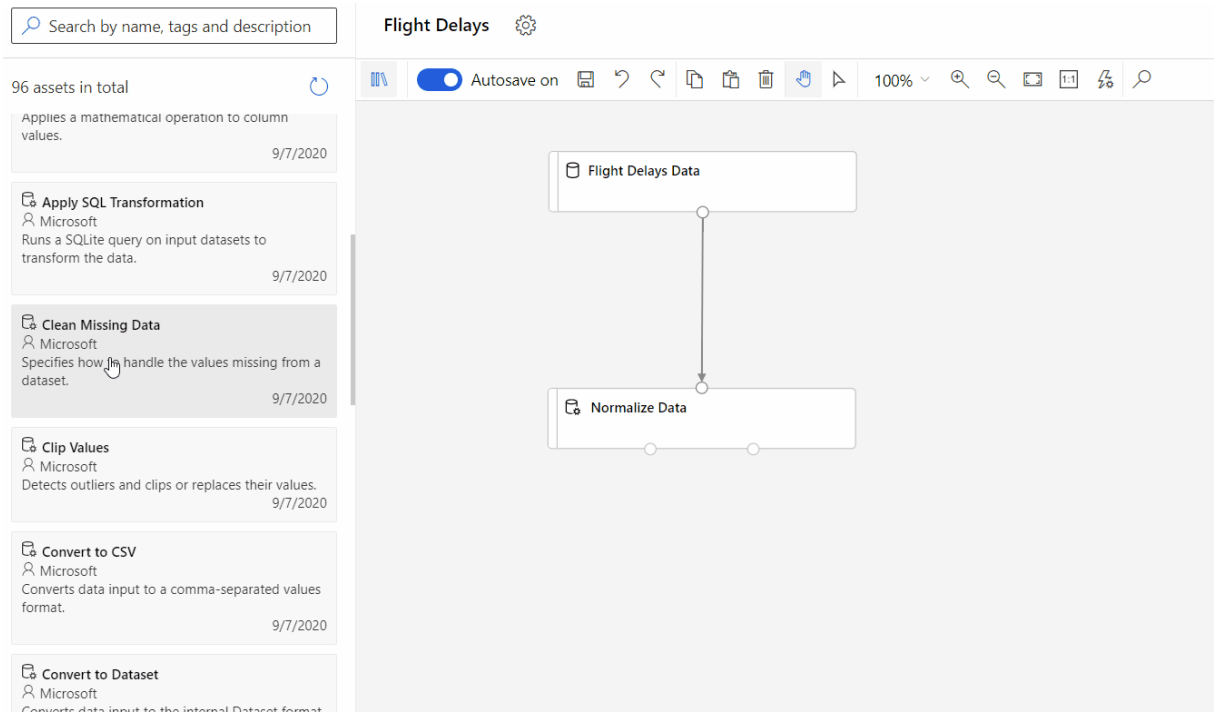
In the following image, we can see an overview of the AutoML pipeline building and recommending a model.



Designer

If you prefer a low code option, *Azure Machine Learning Designer* within the Azure Machine Learning studio gives you a **visual canvas with drag and drop controls to manipulate datasets and modules.**

Modules within **Azure Machine Learning Designer** are algorithms that can have a range of purposes, from simple data ingestion functions to training, scoring, and validation processes. These linked modules create effective ML pipelines to build, test, and deploy ML models, as seen in the following example.



Compute

The computing resources you need for your ML environment can be allocated or attached through Azure Machine Learning.

From the compute that powers your Jupyter notebooks for exploratory data analysis, to the clusters you'll use for training and the managed endpoints you'll deploy to for production inferencing at scale, all can be created and managed through Azure Machine Learning.

Managing data

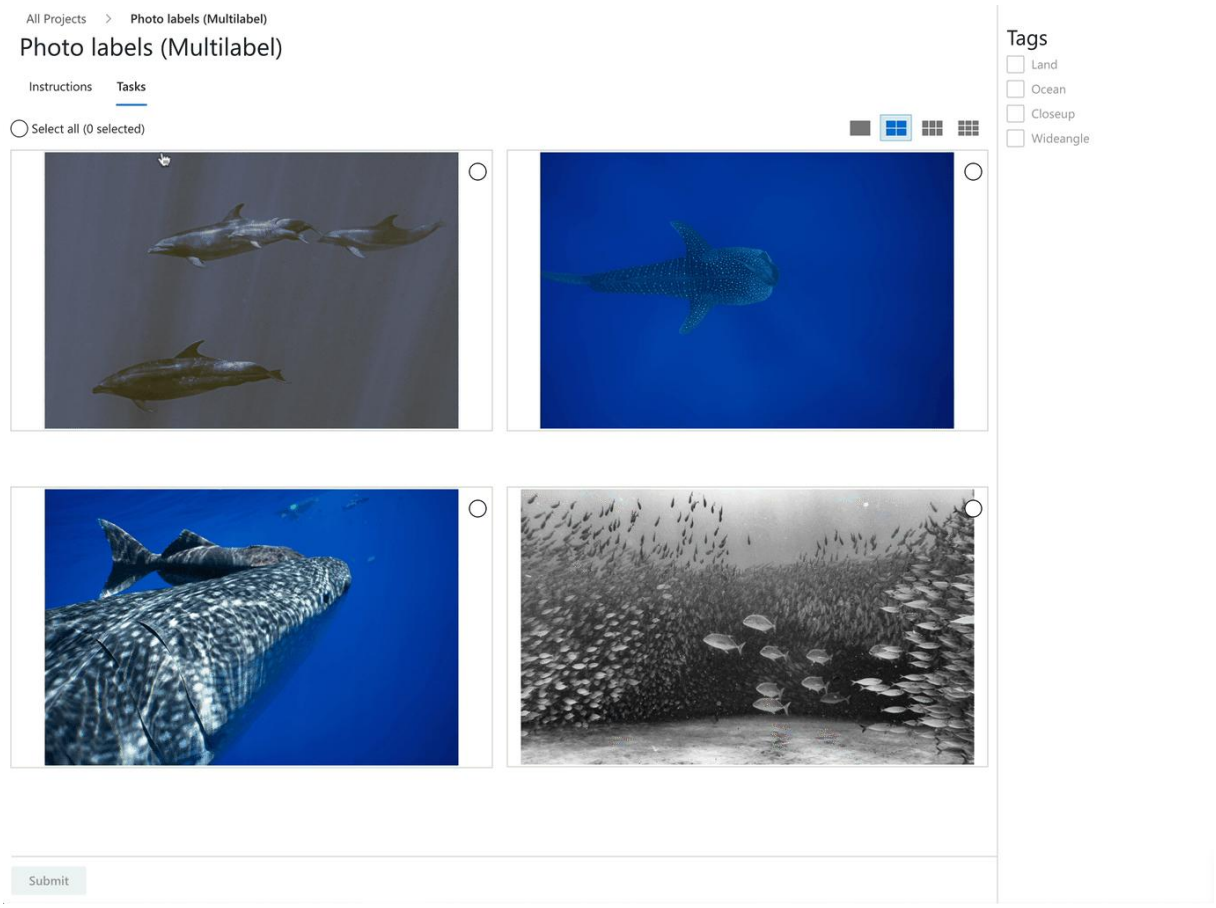
With Azure Machine Learning, the time-intensive process of data preparation and ingestion can be streamlined and collaboratively worked on. The platform smoothly integrates with *Azure Synapse, Azure Databricks,* and a suite of other Azure services to assist data engineering pipelines to extract, transform, and load (ETL) raw data into datastores.

Datastores and labeling

Azure Machine Learning securely stores your raw data in the datastore, so you don’t have to rely on external sources for your scripts, and your training sets can be experimented upon without risking the integrity of the original raw data.

Once stored, you can *clean, transform, and label data* to create ML datasets directly from the datastore.

NB: Azure Machine Learning offers tools to help label tabular, image, and text data—with built-in machine learning systems that can suggest labels or fully automate data labeling. The following example shows a human led multi-label classification project underway in the Azure Machine Learning studio.



MLOps

All models—including those that work perfectly when deployed—require monitoring and retraining over time to maintain high performance. Azure Machine Learning provides *Machine Learning Operations (MLOps)* capabilities to create repeatable steps for data preparation, training, scoring, and reusable software environments for easy deployment. These reproducible pipelines and environments deliver a **continuous integration/continuous delivery (CI/CD)** experience to your machine learning workflow.

In addition to these features, Azure Machine Learning also provides MLOps monitoring tools to notify you of events within the ML lifecycle—and even react to them. With this control, you can quickly identify and respond to diminishing model performance or issues within datasets. Governance information is also provided within Azure Machine Learning, so you can view a full run history and track team members' actions within the model life cycle.

How Azure Machine Learning works

Using Azure Machine Learning requires an Azure account and Azure subscription. The subscription has standard and free pricing tiers and provides an endpoint and subscription key to access the service.

You can access Azure Machine Learning on the cloud or on your local machine through the Python software development kit (SDK), REST API, and Command Line Interface (CLI) extension.

If you prefer low or no-code options, then Azure Machine Learning studio can be used to quickly train and deploy machine learning models.

Azure Machine Learning manages all the resources you need for the ML lifecycle inside a workspace. Workspaces can be shared by multiple people and include things like the computing resources available for your notebooks, training clusters, and pipelines. Workspaces are also the logical containers for your data stores and a repository for models and anything else within the model life cycle.

Deploying machine learning models

Azure Machine Learning can package and run models in Docker containers for deployment. These containers are separate from the run script, so you can quickly swap or update your models with improved ones while leaving the script unmodified.

You can also download the model into the *Open Neural Network Exchange (ONNX)* format, which imparts broad flexibility in potential deployment platforms and devices—such as iOS, Android, and Linux.

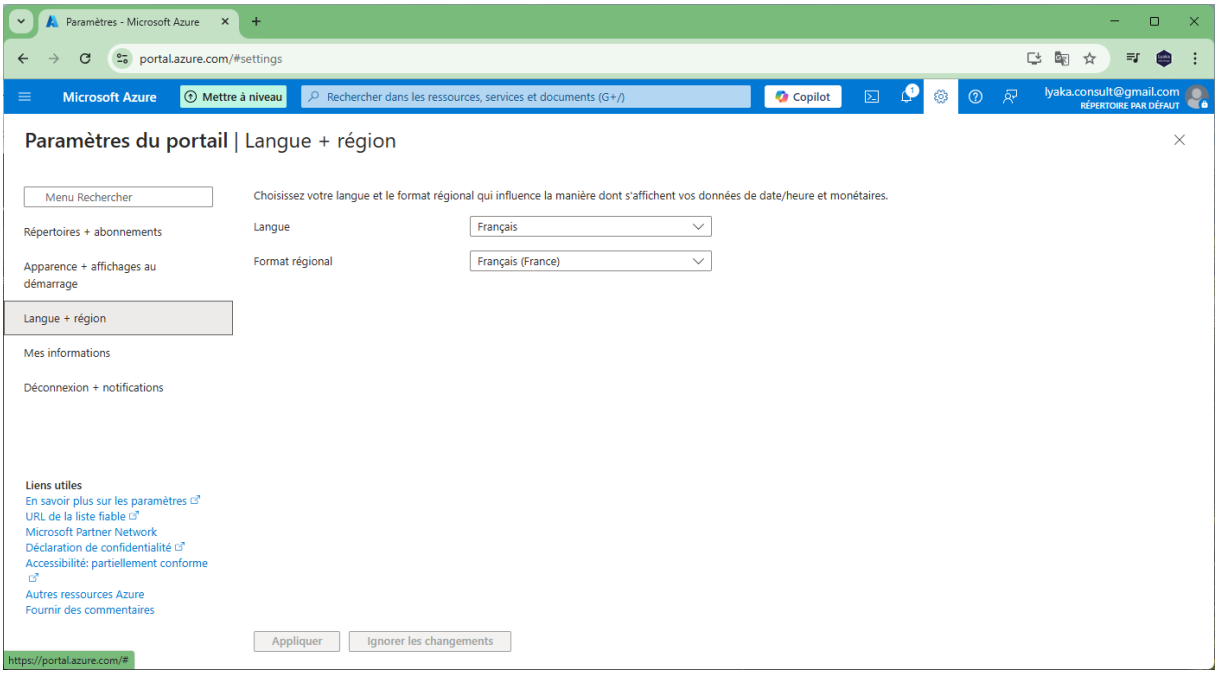
Azure Machine Learning provides pre-packaged container images that include stable environments, with preloaded Python packages and settings. These container images help you deploy to popular machine learning frameworks such as TensorFlow and PyTorch with minimal setup.

Lab 00 - Create an Azure Machine Learning Workspace

To get access to an Azure Machine Learning workspace, you first need to create the **Azure Machine Learning** service in your Azure subscription.

The **workspace** is a critical place where you can work with all resources and assets available to train and deploy machine learning models. For reproducibility, the workspace stores a history of all training jobs, including logs, metrics, outputs, and a snapshot of your code.

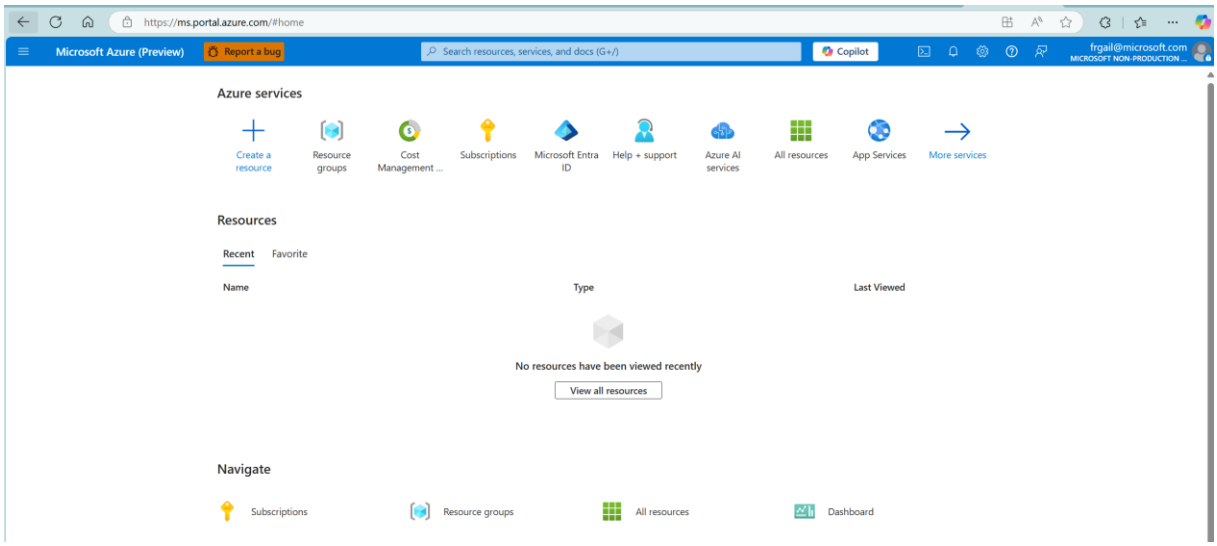
You can also take time to change some parameters as language setting:



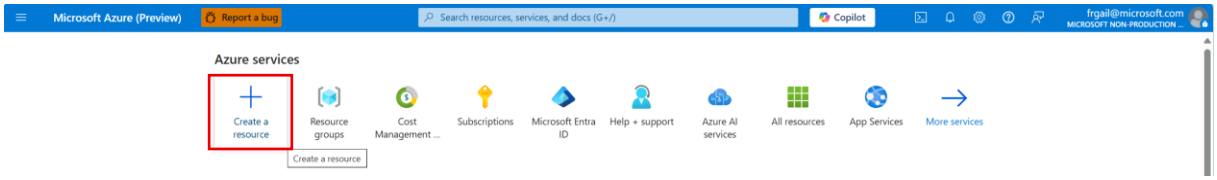
Lab 01 - Create an Azure Machine Learning (Azure ML) service

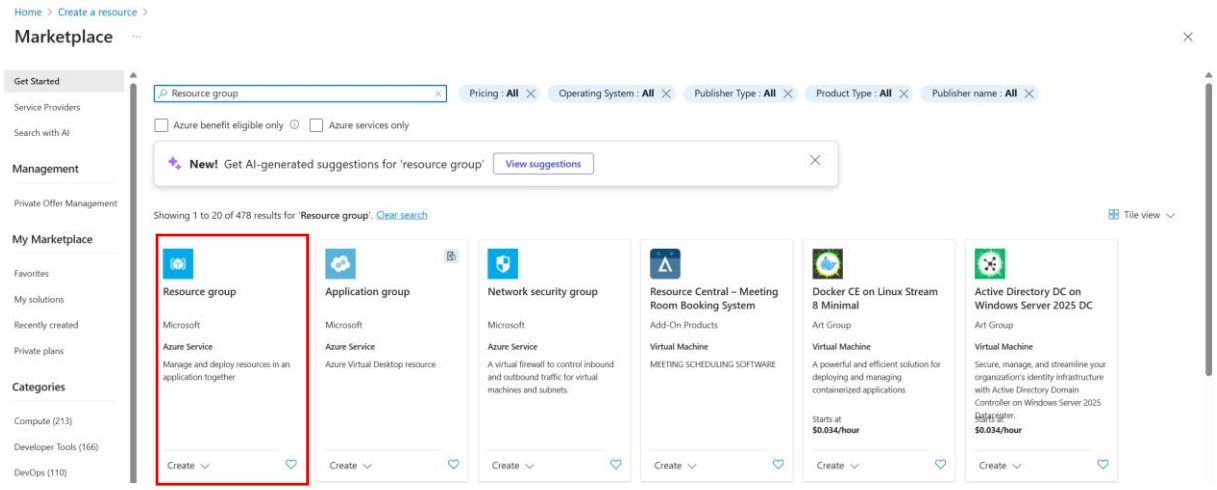
To create an Azure Machine Learning service, you must:

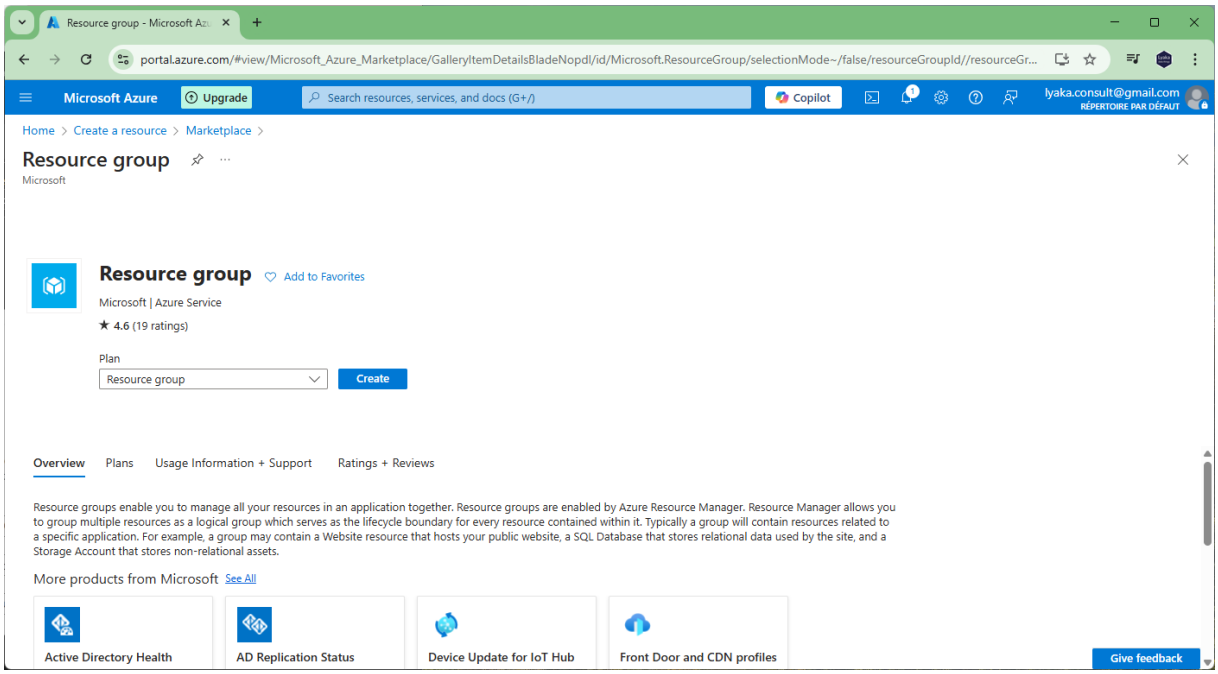
- Get access to **Azure Portal** (Home - Microsoft Azure) and **sign in** to get access to an **Azure subscription:**

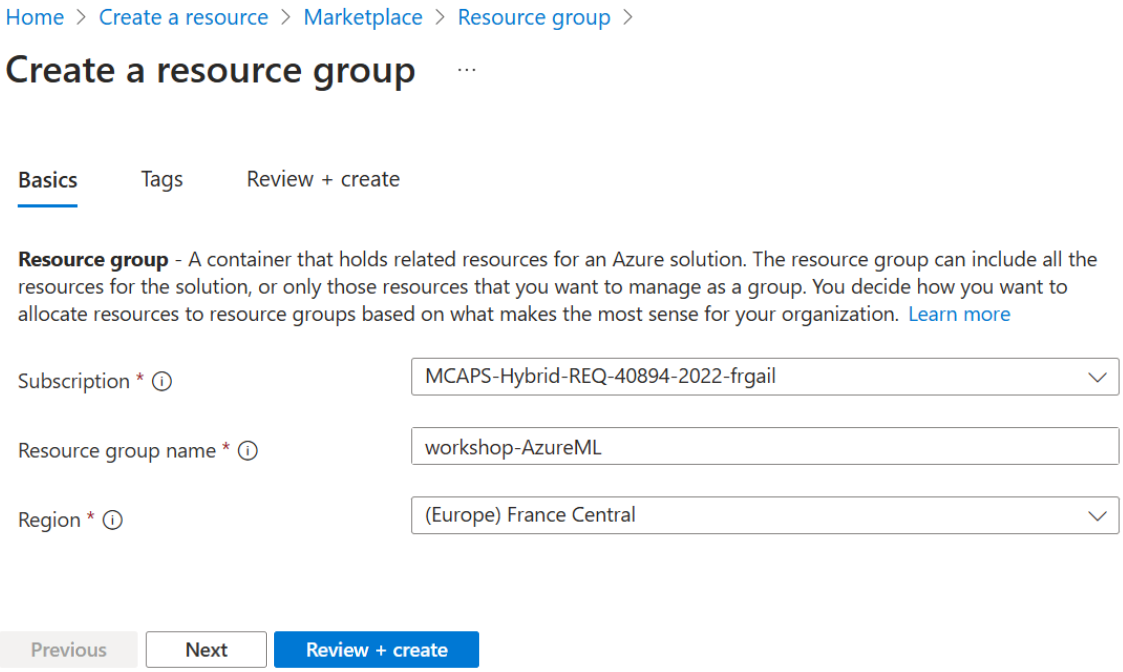


- Create a **resource group** within your subscription:

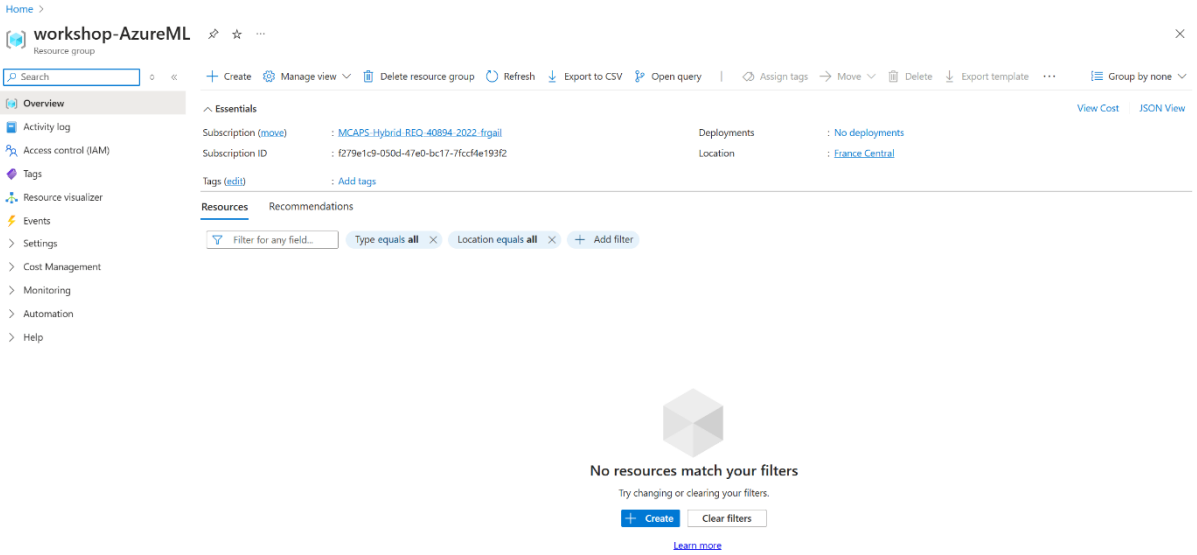




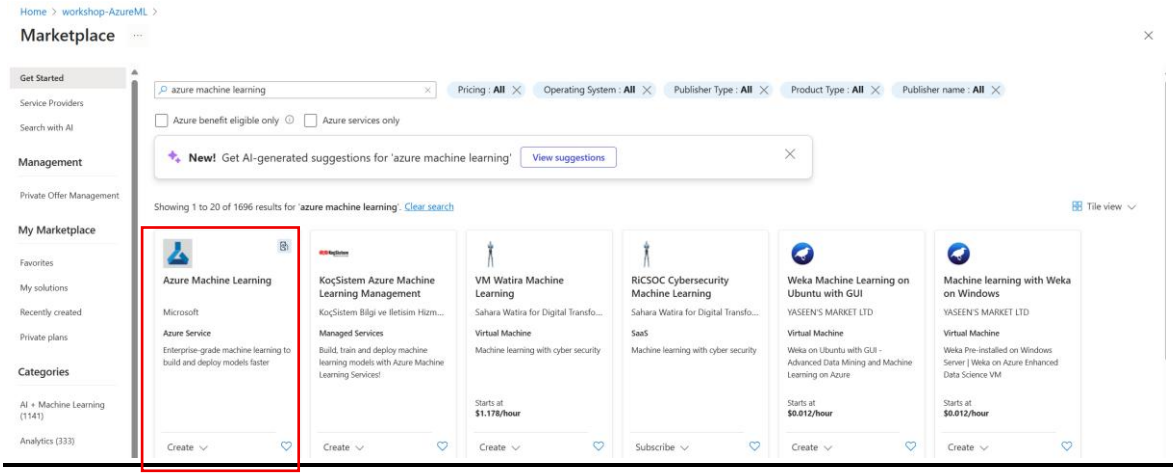


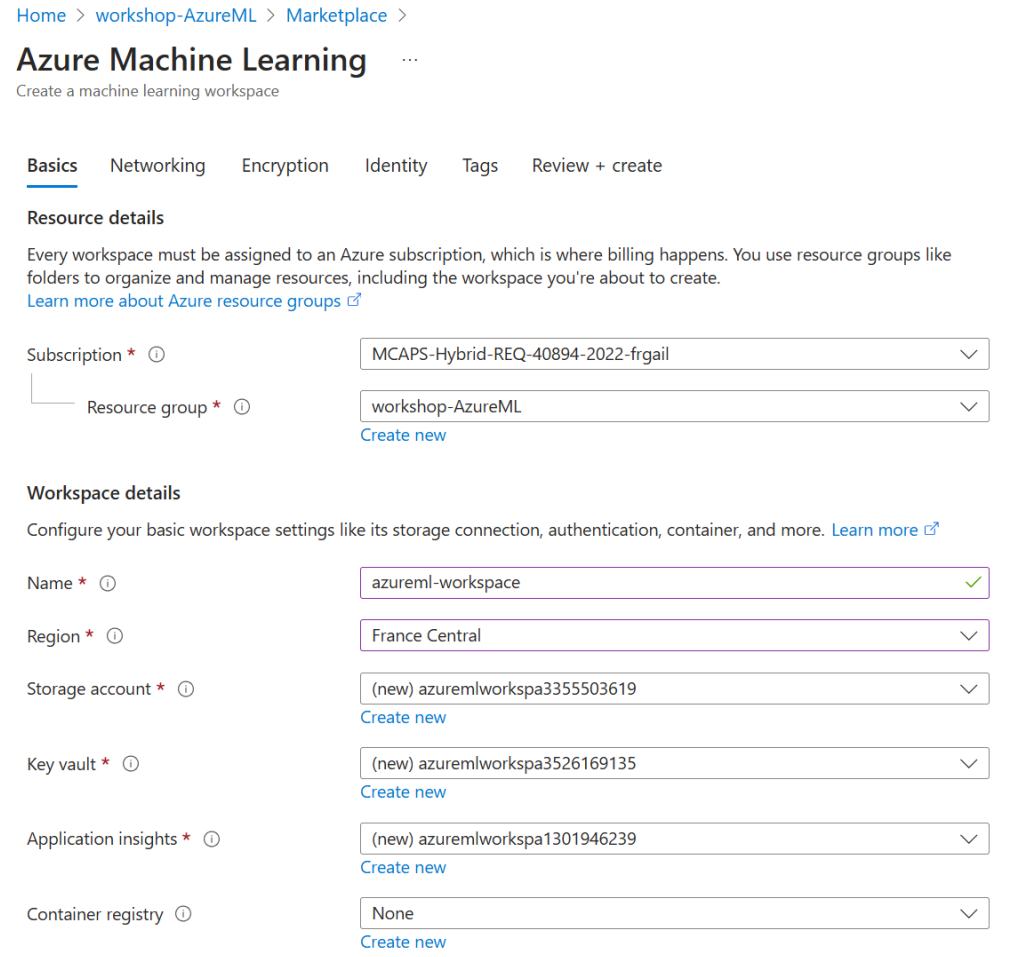


- **Click on** Review + create **and** Create
- Click to access the resource group “workshop-AzureML”

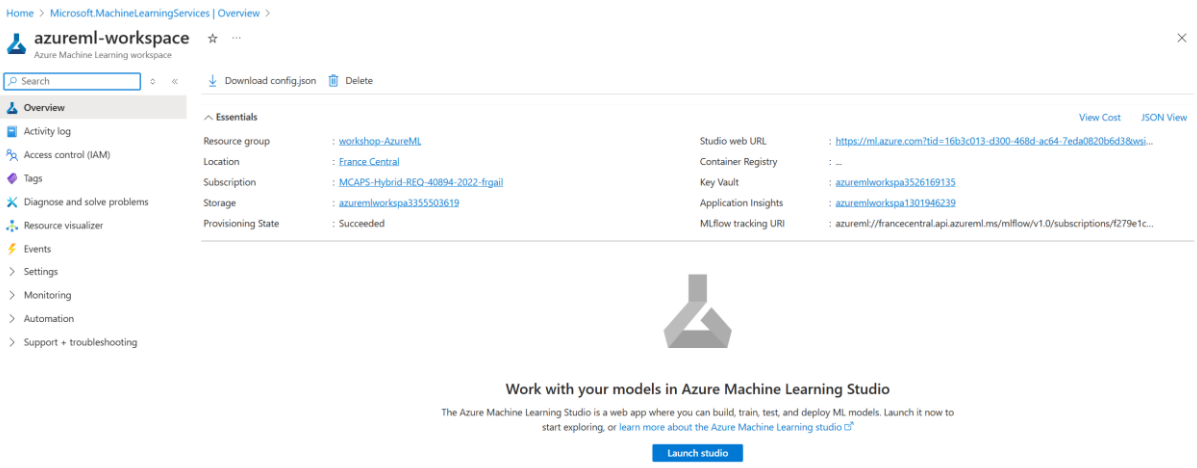


- Now that you have a resource group, we can create a "**workspace"** which is a kind of **Azure Machine Learning Services.**
- **Click on "+Create"**
- **Search for "Azure Machine Learning" then click on "Create"**

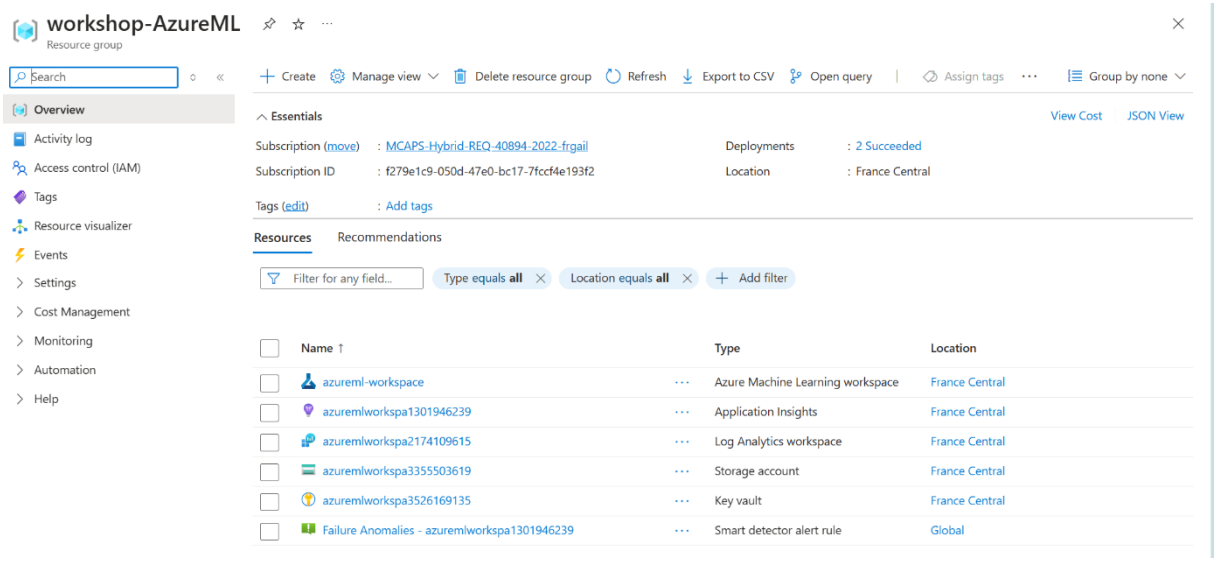




- **Click on** Review + create **and** Create
- Wait until the creation process succeed



- When a workspace is provisioned, Azure automatically creates other Azure resources within the same resource group to support the workspace:
    1. **Azure Storage Account**: To store files and notebooks used in the workspace, and to store metadata of jobs and models.
    2. **Azure Key Vault**: To securely manage secrets such as authentication keys and credentials used by the workspace.
    3. **Application Insights**: To monitor predictive services in the workspace.
    4. **Azure Container Registry**: Created when needed to store images for Azure Machine Learning environments.



Lab 02 - Create and manage compute resources

The **workspace** is the top-level resource for Azure Machine Learning. Data scientists need access to the workspace to train and track models, and to deploy the models to endpoints.

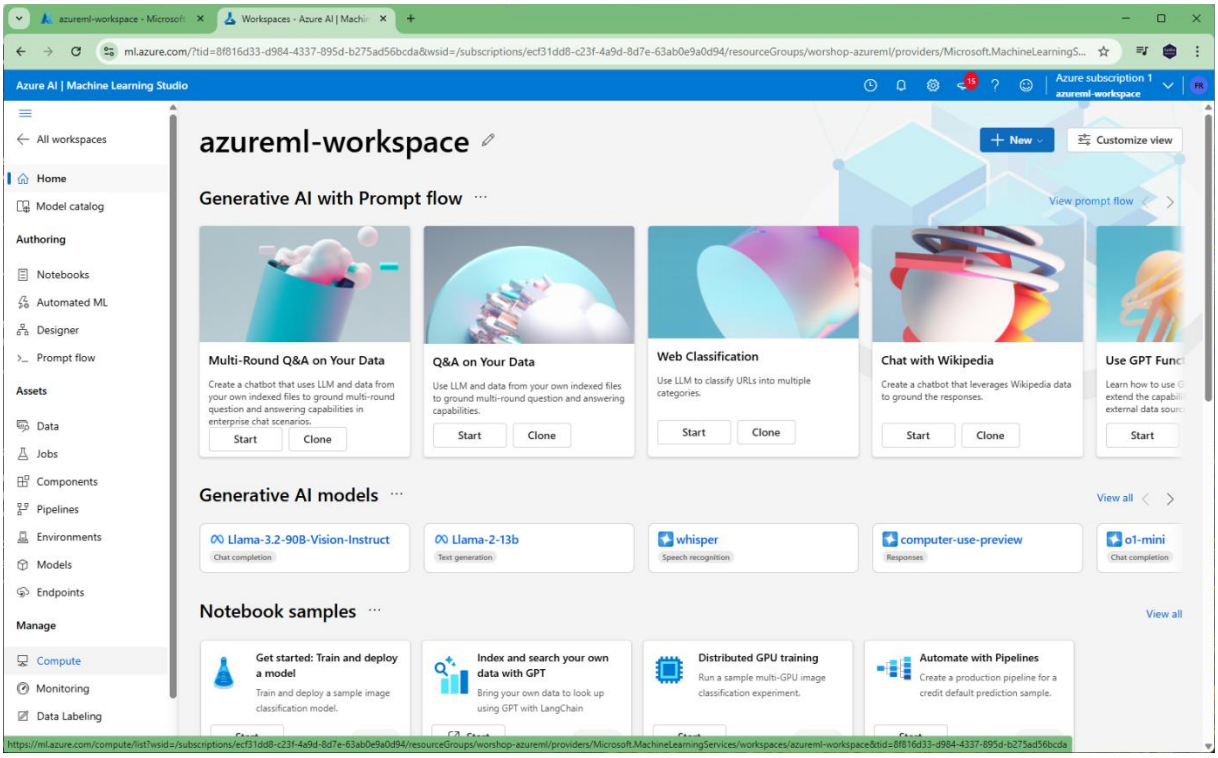
One of the most important resources you need when training or deploying a model is **compute**. There are five types of computes in the Azure Machine Learning workspace:

- **Compute instances**: A kind of virtual machine in the cloud, managed by the workspace. Ideal to use as a development environment to run (Jupyter) notebooks.
- **Compute clusters**: On-demand clusters of CPU or GPU compute nodes in the cloud, managed by the workspace. Ideal to use for production workloads as they automatically scale to your needs.
- **Kubernetes clusters**: Allows you to create or attach an Azure Kubernetes Service (AKS) cluster. Ideal to deploy trained machine learning models in production scenarios.
- **Attached computes**: Allows you to attach other Azure computes resources to the workspace, like Azure Databricks or Synapse Spark pools.
- **Serverless compute**: A fully managed, on-demand compute you can use for training jobs.

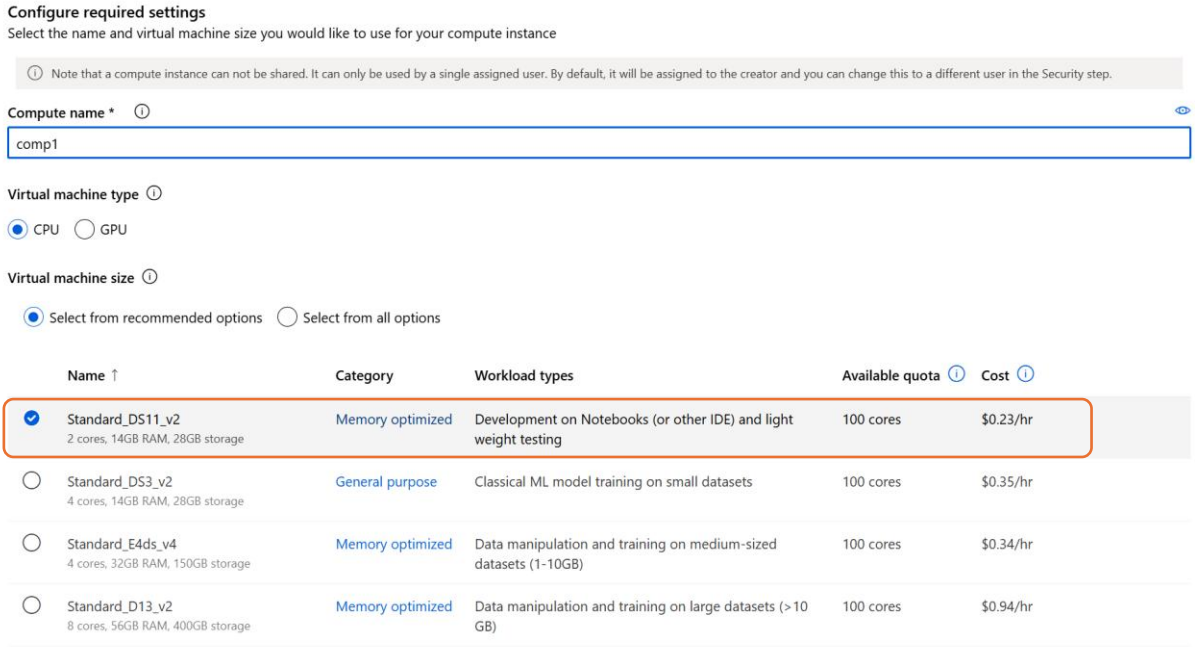
Create a Compute Instance

A compute instance in Azure Machine Learning is a fully managed virtual machine that serves as a development environment in the cloud. It allows you to run multiple jobs in parallel, manage a job queue, and execute tasks securely within a containerized environment.

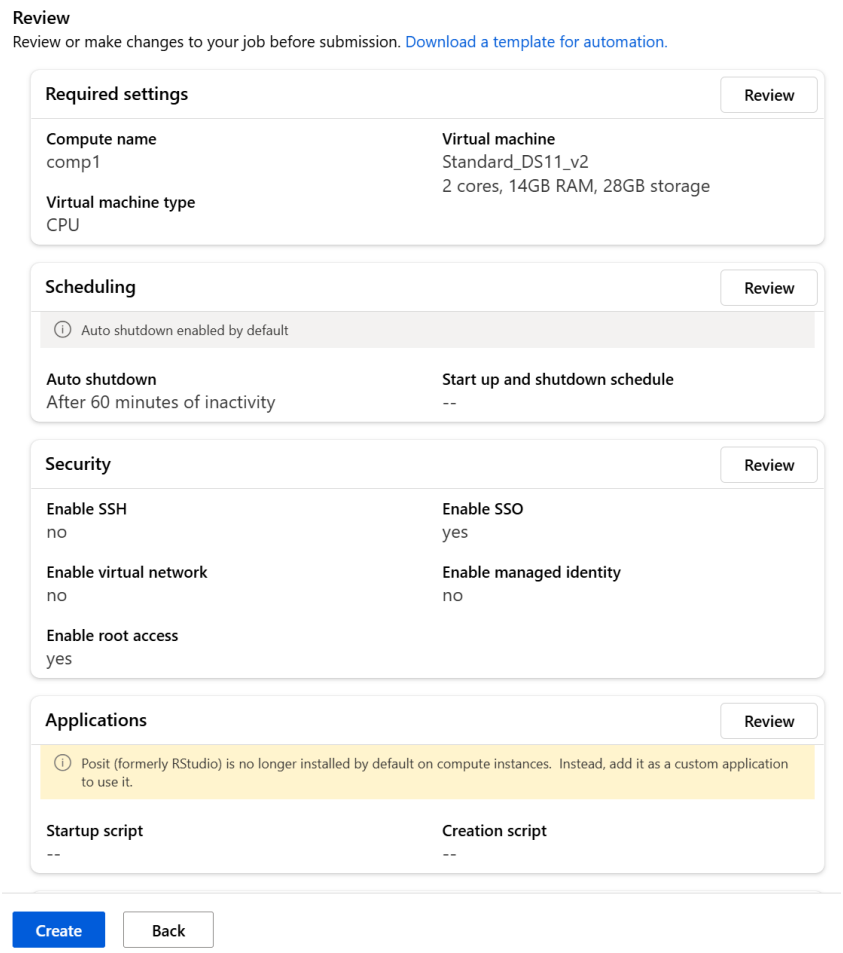
- Lunch Azure ML studio

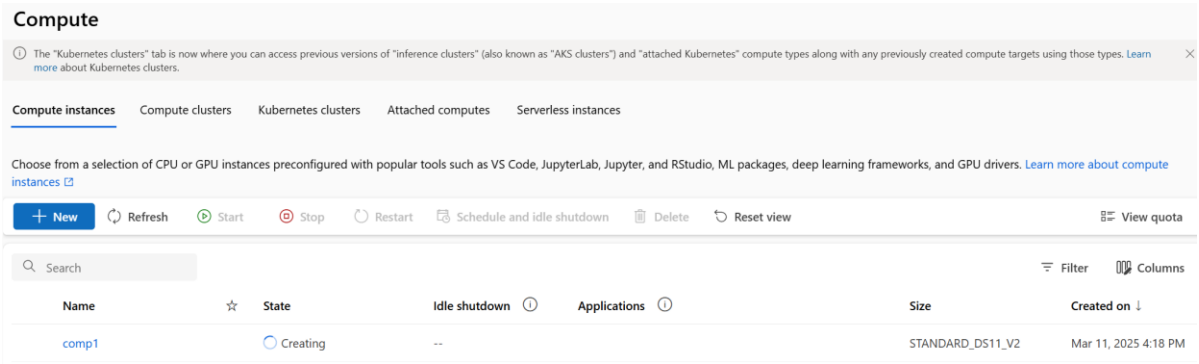


- To create a compute instance in Azure Machine Learning, navigate to your Azure Machine Learning workspace and select the **"Compute"** tab.
- From there, click on **"New"** and choose **"Compute Instance."** You will need to specify the name, type, and size of the compute instance.



- Click on **“Next”** until the last page and click on **“Create”**.

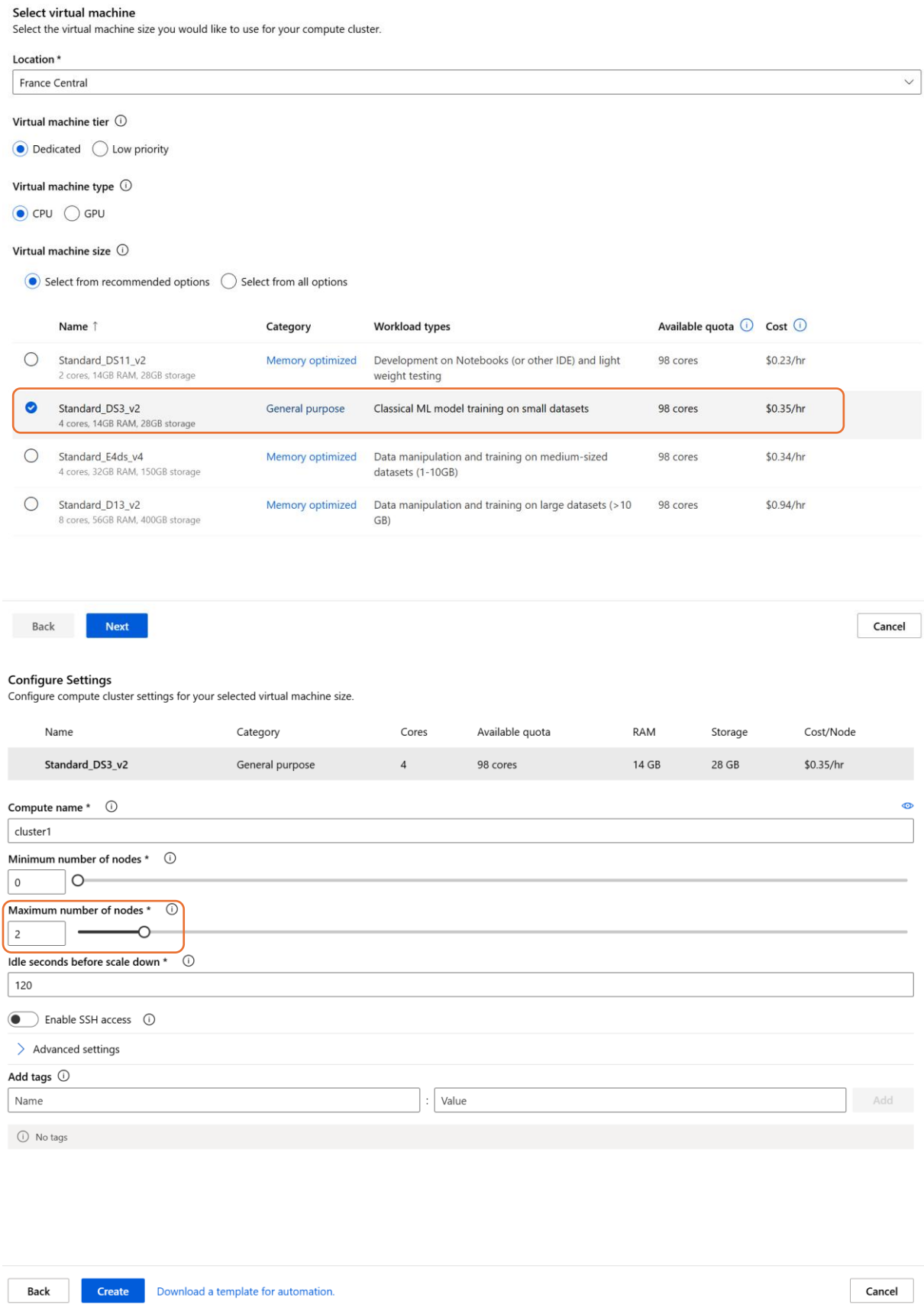




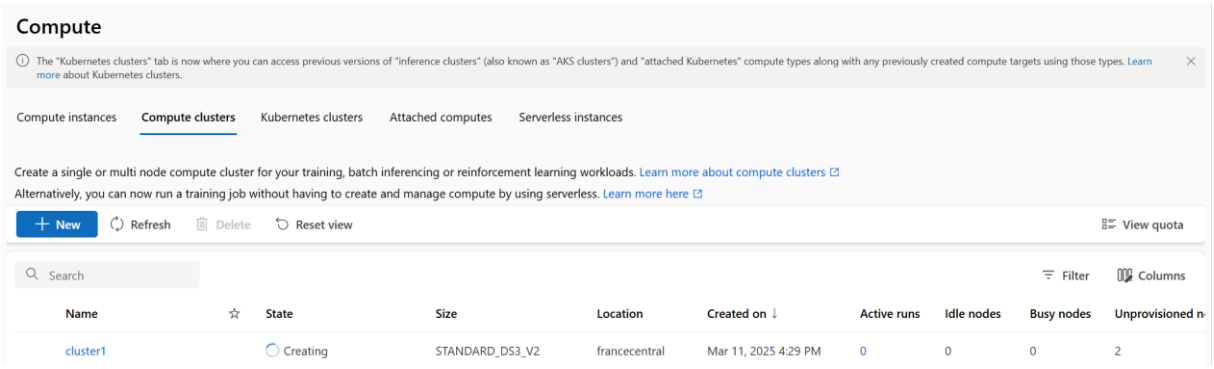
Create a Compute Cluster

Compute clusters are designed to handle large-scale machine learning workloads by distributing tasks across multiple nodes, allowing for efficient parallel processing. They support both CPU and GPU resources, and can automatically scale based on the demands of your jobs

- To create a compute cluster in Azure Machine Learning, navigate to your Azure Machine Learning workspace and select the **"Compute"** tab.
- From there, click on **"New"** and choose **"Compute Cluster."** You will need to specify the name, type, and size of the cluster.



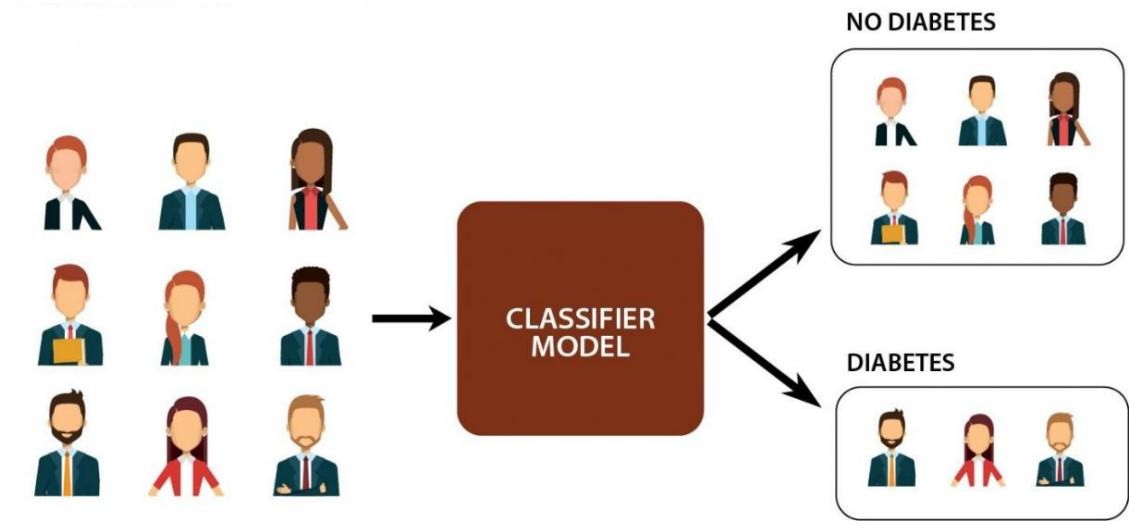
- Click on **“Next”** until the last page and click on **“Create”**.



Lab 01 - Explore Classification with Azure ML Designer

Azure ML Designer is a user-friendly, drag-and-drop interface within Azure Machine Learning that simplifies the process of building machine learning models. It allows users to visually design and experiment with machine learning pipelines by leveraging pre-built components and custom algorithms.

Lab 11 – What is Classification in Machine Learning



Classification in machine learning is a predictive modeling process where algorithms are used to categorize data into predefined classes or categories. This process involves training a model to recognize patterns and relationships within a dataset, enabling it to make accurate predictions about new, unseen data.

The classification process typically involves two main steps: **learning** and **prediction**.

During the **learning phase**, the model is trained using labeled data, where each data point is associated with a specific class label. The model analyzes the features of the data points and learns to identify the characteristics that define each class.

For example, in a dataset of medical records, the model might learn to distinguish between individuals suffering from diabetes and those who do not based on features such as blood glucose levels, age, body mass index (BMI), and family medical history.

Once the model has been trained, it enters the **prediction phase**. Here, the model applies the knowledge it gained during training to classify new data points. The model evaluates the features of the new data and assigns them to the most appropriate class based on the patterns it learned.

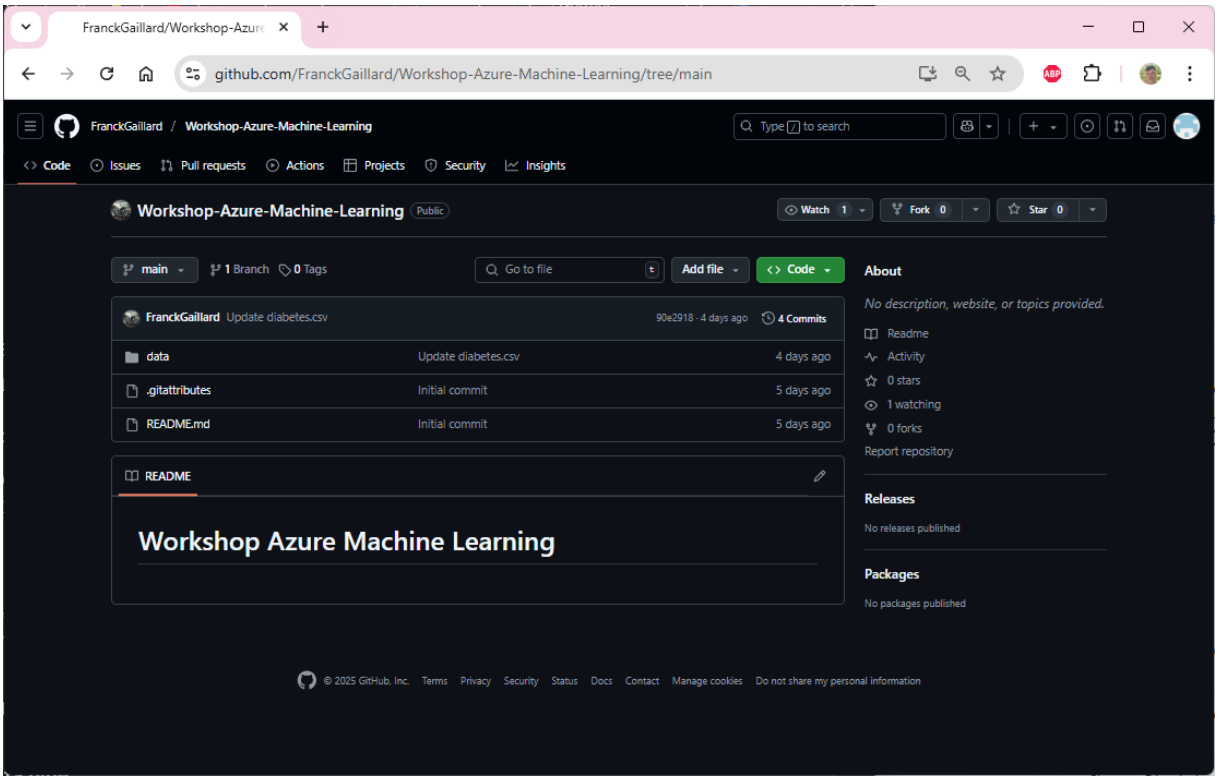
Lab 12 – Build a Classification model with Azure ML Designer

In this chapter, we will explore how to build a classification model using Azure ML Designer. Azure ML Designer offers a user-friendly, drag-and-drop interface that simplifies the process of creating machine learning models. Whether you are a novice or an experienced data scientist, Azure ML Designer provides the tools and components needed to design, train, and evaluate classification models efficiently. By leveraging pre-built modules and custom algorithms, you can visually construct your machine learning pipelines and gain insights into your data, ultimately enhancing the accuracy and performance of your models.

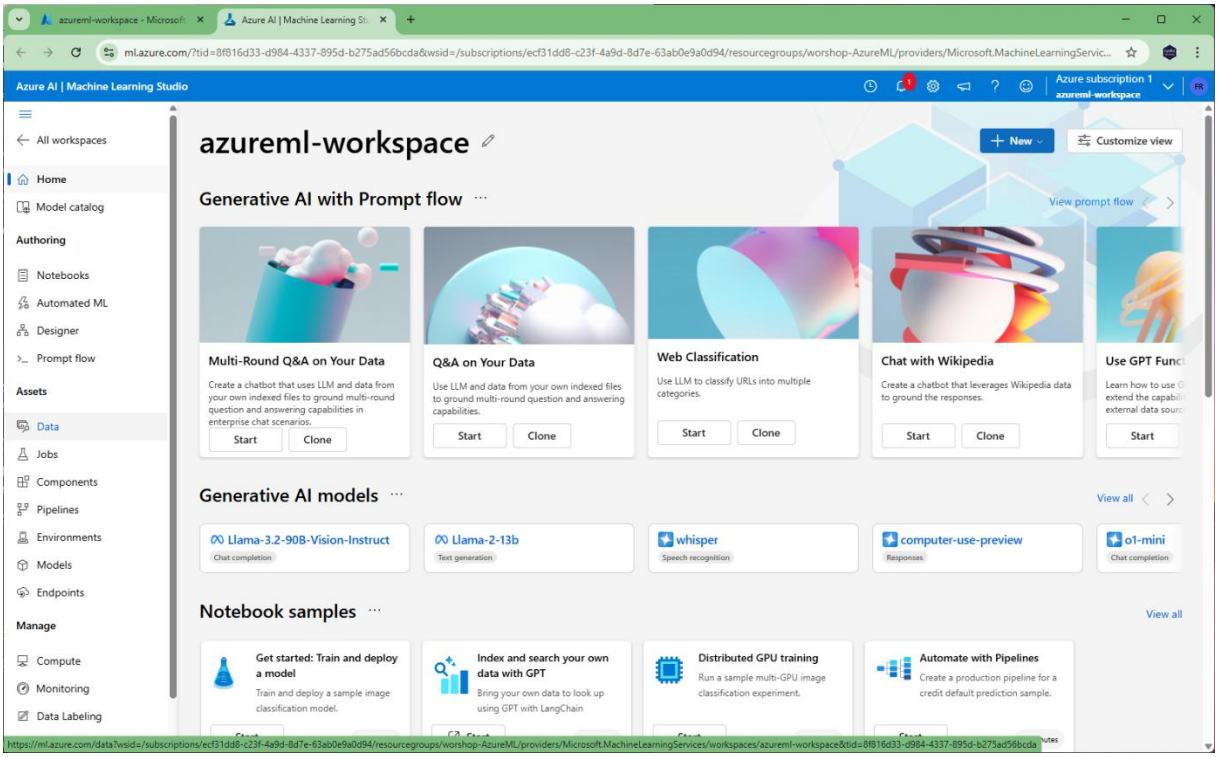
Create a dataset

In Azure Machine Learning, a dataset is essentially a reference to the location of your data source along with its metadata. This means that the data remains in its existing location, and you don't incur extra storage costs or risk the integrity of your data sources.

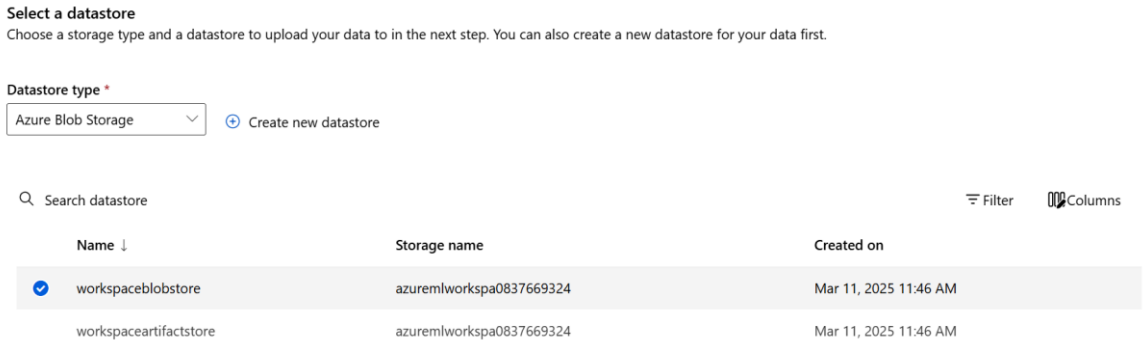
Prerequisite: download the **"diabetes.csv"** file from this link: Workshop-Azure-Machine-Learning/data at main · FranckGaillard/Workshop-Azure-Machine-Learning



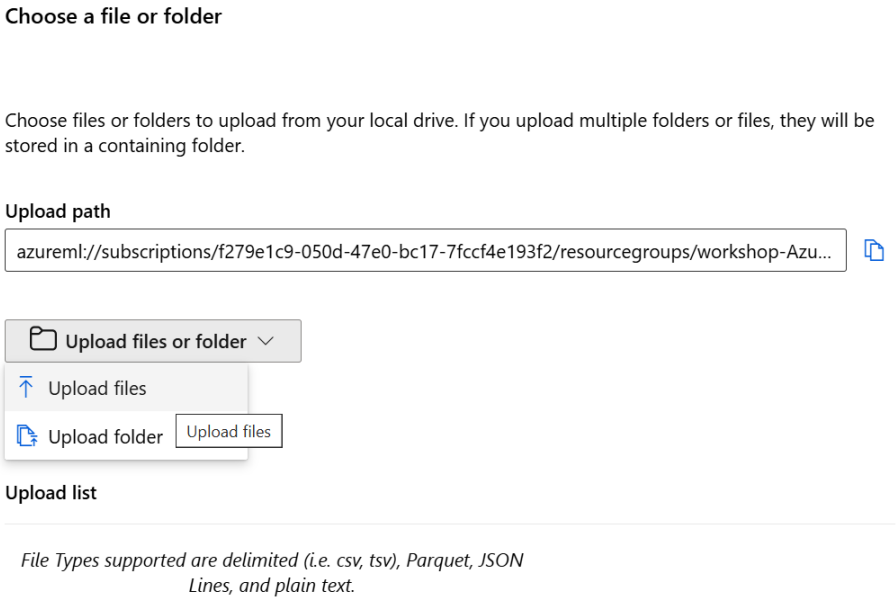
- In Azure ML studio, expand the left pane by selecting the menu icon at the top left of the screen. Select the **Data** page (under **Assets**). The Data page contains specific data files or tables that you plan to work with in Azure ML. You can create datasets from this page as well.



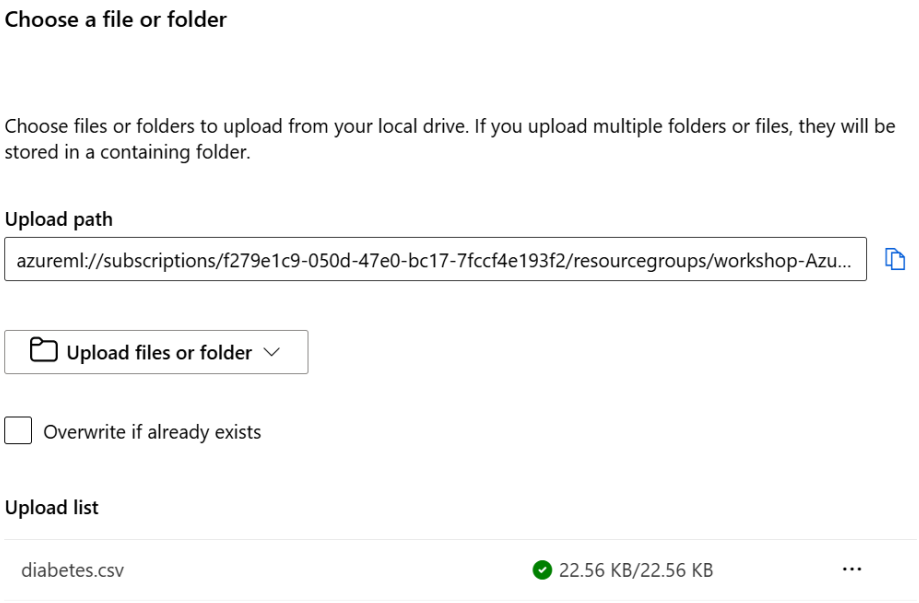
- On the **Data** page, under the **Data assets** tab, select **+ Create**. Then configure a data asset with the following settings:
    - **Data type**:
        - **Name**: diabetes-data
        - **Description**: Diabetes data
        - **Dataset type**: Tabular
    - **Data source**: From local files
    - Leave the default options:



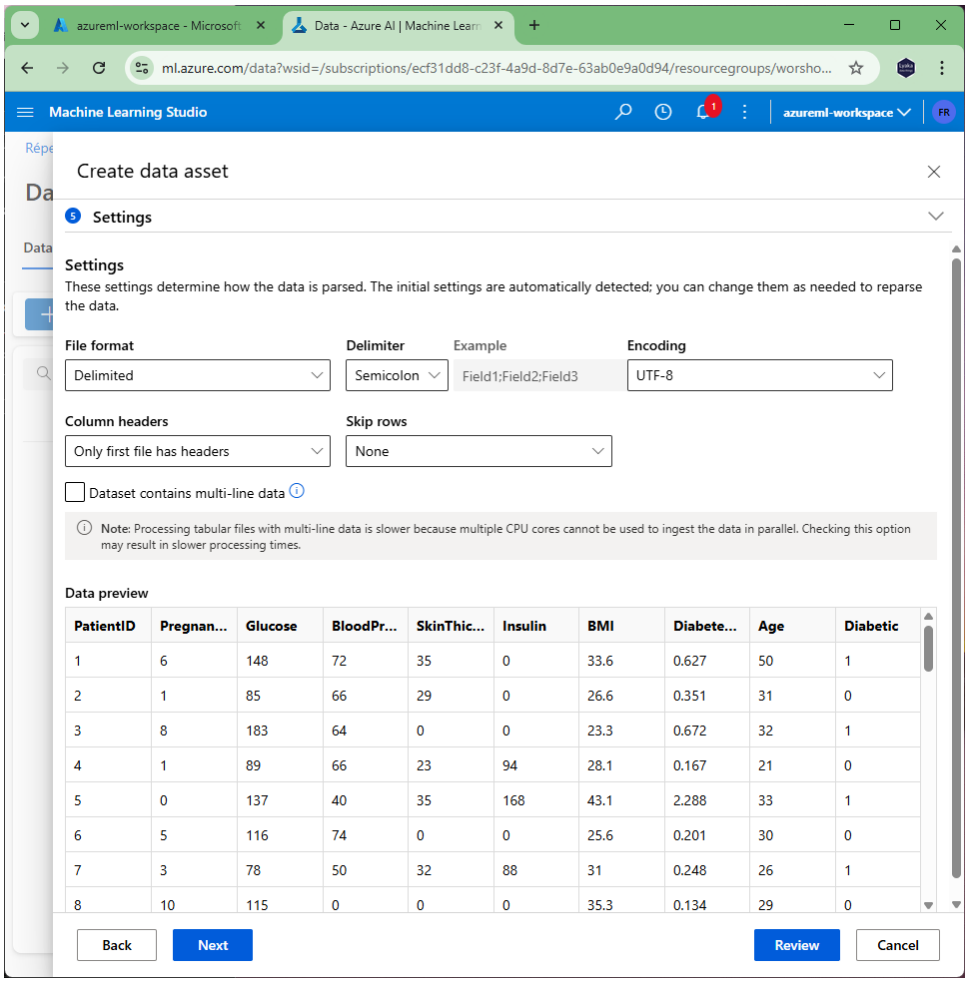
    - Click on **Upload files or folder**, and choose **Upload files:**



    - Select the file **"diabetes.csv"** you previously downloaded:



- **Settings**:
  - **File format**: Delimited
  - **Delimiter**: Comma_
  - **Encoding**: UTF-8
  - **Column headers**: Only first file has headers
  - **Skip rows**: None
  - **Dataset contains multi-line data**: *do not select*



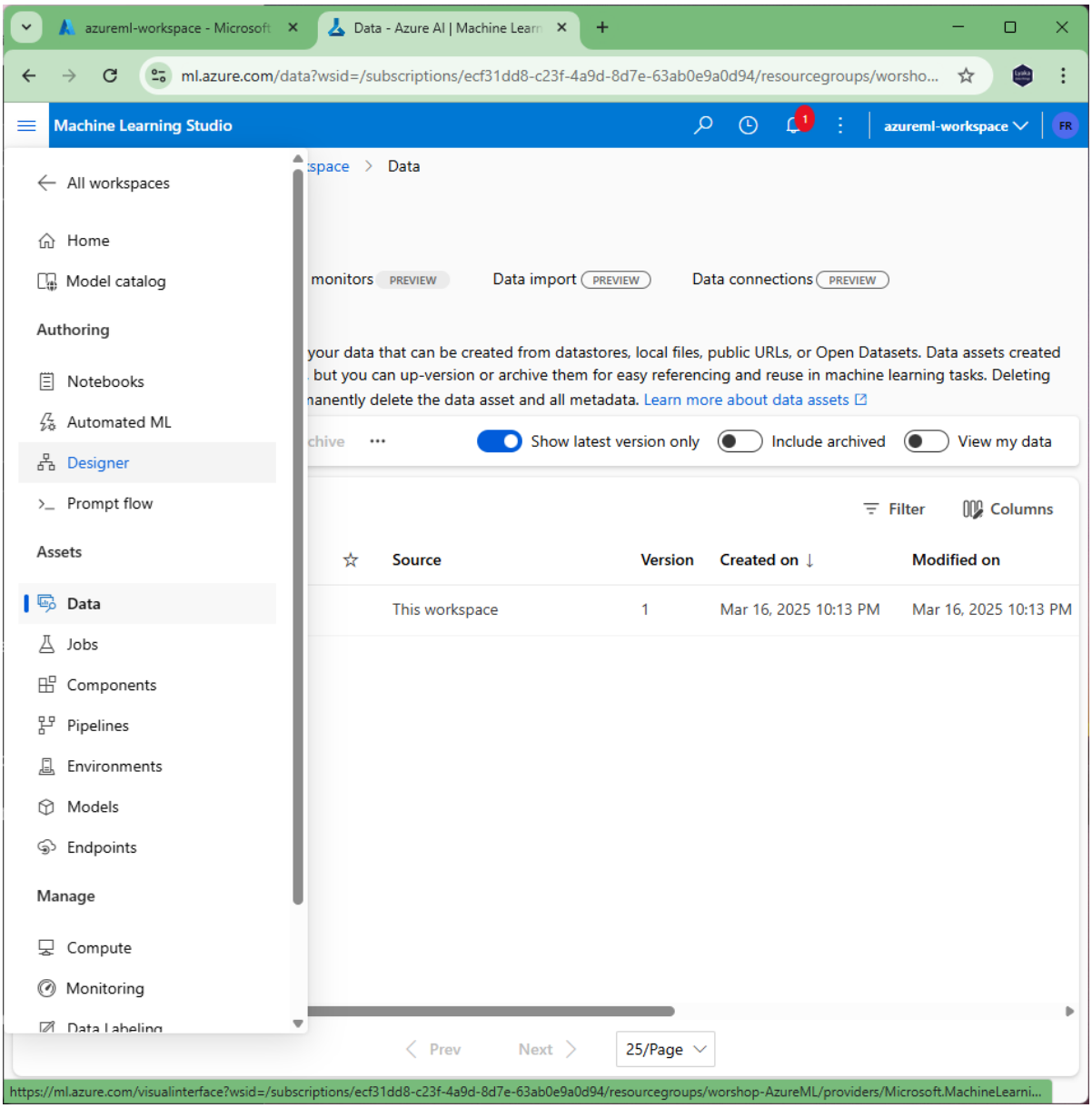
- **Schema**:
  - Include all columns other than **Path**
  - Review the automatically detected types
- **Review**
  - Select **Create**

- After the dataset has been created, open it and view the **Explore** page to see a sample of the data. This data represents details from patients who have been tested for diabetes.

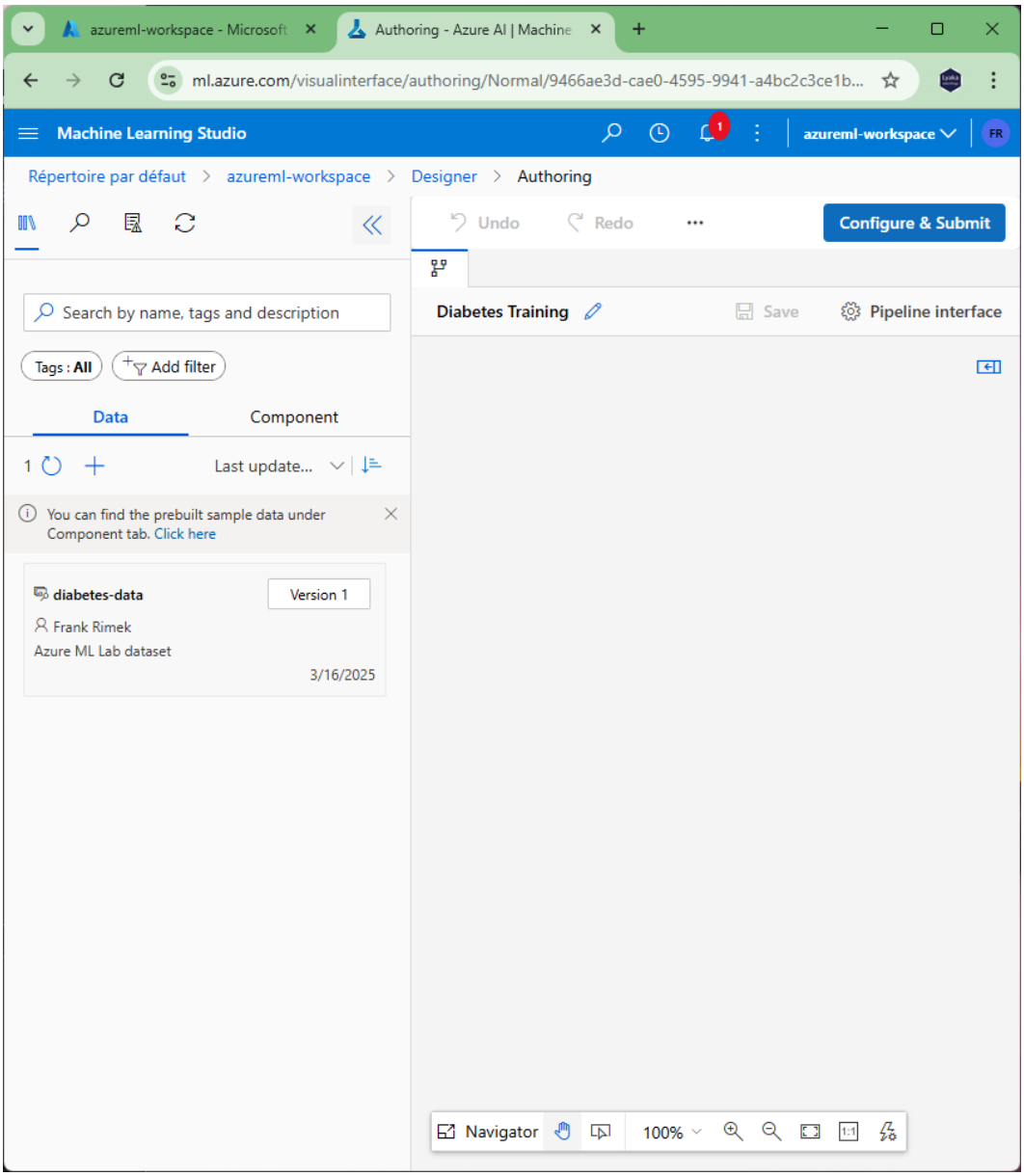
Create a pipeline in Designer and load data to canvas

To get started with Azure Machine Learning designer, first you must create a pipeline and add the dataset you want to work with.

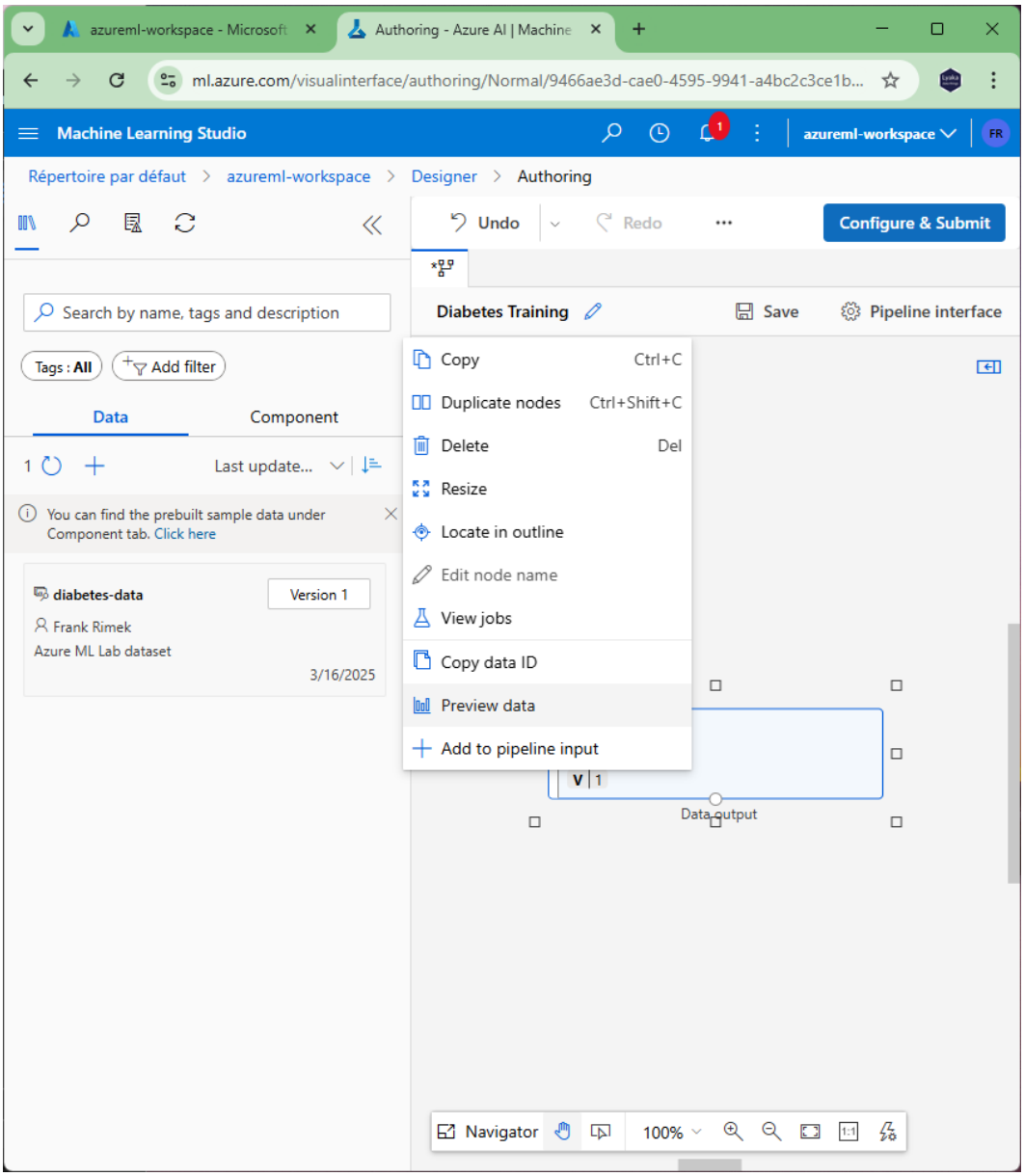
- In Azure ML studio, on the left pane select the **Designer** item (under **Authoring**), and then select **+** to create a new pipeline.



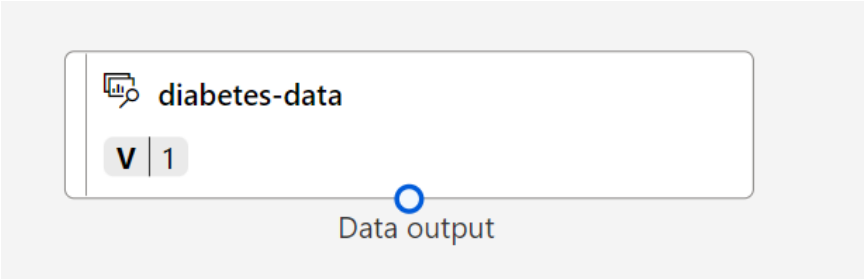
- Change the draft name from **Pipeline-Created-on-*date*** to **Diabetes Training**.
- Then in the project, next to the pipeline name on the left, select the arrows icon to expand the panel if it is not already expanded. The panel should open by default to the **Asset library** panel, indicated by the book's icon at the top of the panel. Note that there is a search bar to locate assets. Notice two buttons, **Data** and **Component**.



- Select **Data**. Search for and place the **diabetes-data** dataset onto the canvas.
- Right-click (Ctrl+click on a Mac) the **diabetes-data** dataset on the canvas and select **Preview data**.



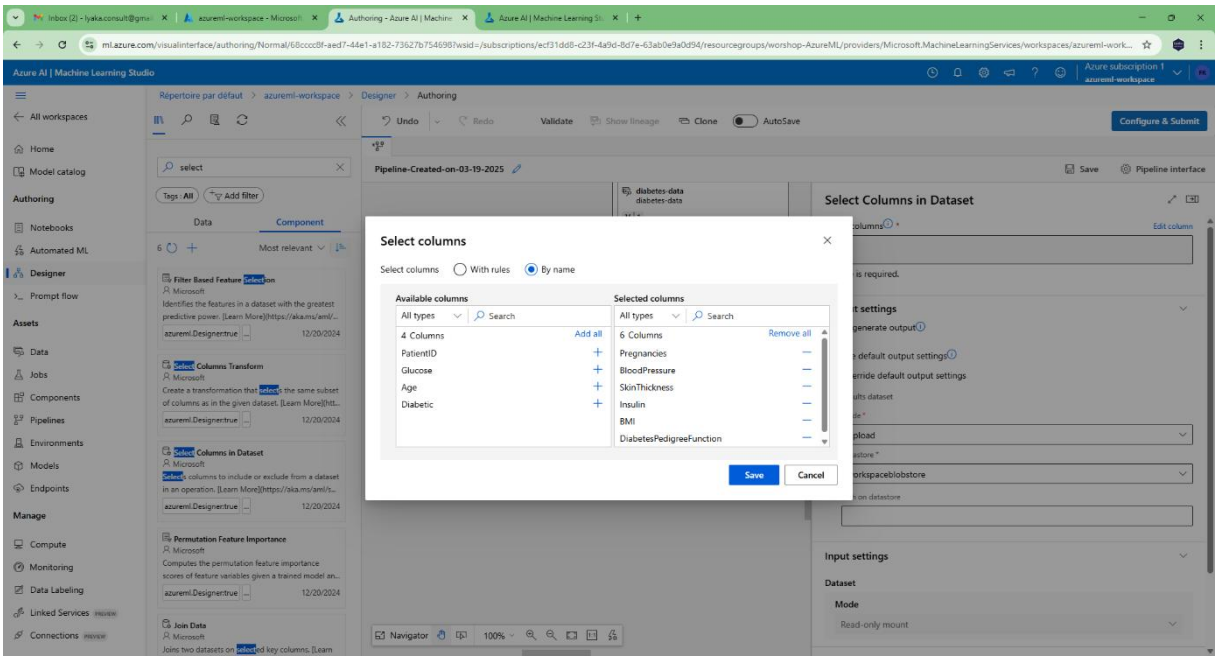
- Review the schema of the data in the *Profile* tab, noting that you can see the distributions of the various columns as histograms.
- Scroll down and select the column heading for the **Diabetic** column and note that it contains two values **0** and **1**. These values represent the two possible classes for the *label* that your model will predict, with a value of **0** meaning that the patient does not have diabetes, and a value of **1** meaning that the patient is diabetic.
- Scroll back up and review the other columns, which represent the *features* that will be used to predict the label. Note that most of these columns are numeric, but each feature is on its own scale. For example, **Age** values range from 21 to 81, while **DiabetesPedigree** values range from 0.08 to 2.42. When training a machine learning model, it is sometimes possible for larger values to dominate the resulting predictive function, reducing the influence of features that on a smaller scale. Typically, data scientists mitigate this possible bias by *normalizing* the numeric columns so they're on the similar scales.
- Close the **DataOutput** tab so that you can see the dataset on the canvas like this:



Add transformations

Before you can train a model, you typically need to apply some pre-processing transformations to the data.

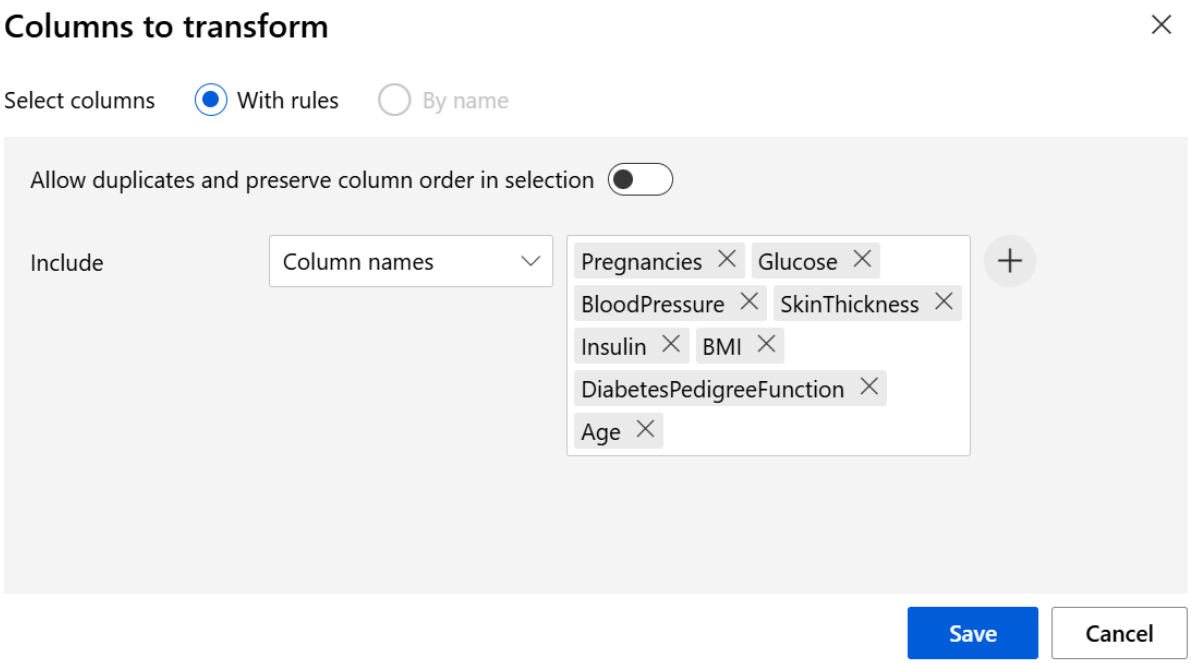
- In the **Asset library** pane on the left, select **Component**, which contains a wide range of modules you can use for data transformation and model training. You can also use the search bar to quickly locate modules.
- Find the **Select Columns in Dataset** module and place it on the canvas below the **diabetes-data** dataset. Then connect the output from the bottom of the **diabetes-data** dataset to the input at the top of the **Select Columns in Dataset** module.
- Double click on the **Select Columns in Dataset** module to access a settings pane on the right. Select **Edit column**. Then in the **Select columns** window, select **By name** and **Add all** the columns. Then remove **PatientID** and click **Save**.



- Find the **Normalize Data** module and place it on the canvas below the **Select Columns in Dataset** module. Then connect the output from the bottom of the **Select Columns in Dataset** module to the input at the top of the **Normalize Data** module, like this:

- Double-click the **Normalize Data** module to view its settings, noting that it requires you to specify the transformation method and the columns to be transformed.
- Set the *Transformation method* to **MinMax** and the *Use 0 for constant columns when checked* to **True**. Edit the columns to transform with **Edit columns**. Select columns **With Rules** and copy and paste the following list under include column names:

  Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age



- Click **Save** and close the selection box. The data transformation is normalizing the numeric columns to put them on the same scale, which should help prevent columns with large values from dominating model training. You'd usually apply a whole bunch of pre-processing transformations like this to prepare your data for training, but we'll keep things simple in this exercise

Run the pipeline

To apply your data transformations, you need to run the pipeline as an experiment.

- Select **Configure & Submit** at the top of the page to open the **Set up pipeline job** dialogue.
- On the **Basics** page select **Create new** and set the name of the experiment to **mslearn-diabetes-training** then select **Next**.
- On the **Inputs & outputs** page select **Next** without making any changes.
- On the **Runtime settings** page an error appears as you don ´t have a default compute to run the pipeline. In the **Select compute type** drop-down select *Compute cluster* and in the **Select Azure ML compute cluster** drop-down select your recently created compute cluster.
- Select **Review + Submit** to review the pipeline job and then select **Submit** to run the training pipeline.
- Wait a few minutes for the run to finish. You can check the status of the job by selecting **Jobs** under the **Assets**. From there, select the **mslearn-diabetes-training** experiment and then the **Diabetes Training** job.

View the transformed data

When the run has completed, the dataset is now prepared for model training.

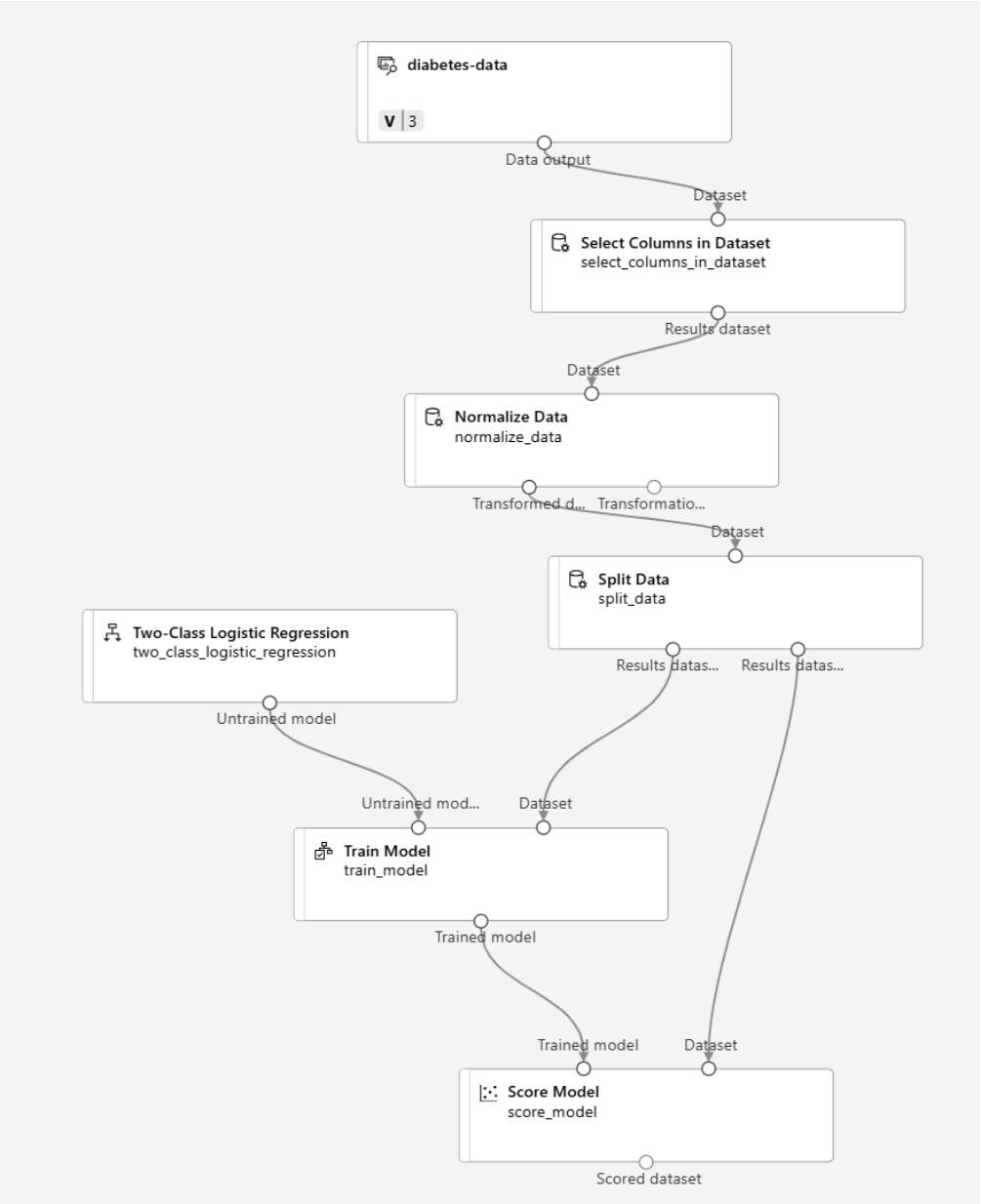
- Right-click (Ctrl+click on a Mac) the **Normalize Data** module on the canvas and select **Preview data**. Select **Transformed dataset**.
- View the data, noting that the numeric columns you selected have been normalized to a common scale.
- Close the normalized data result visualization. Return to the previous tab.

After you've used data transformations to prepare the data, you can use it to train a machine learning model.

Add training modules

It's common practice to train the model using a subset of the data, while holding back some data with which to test the trained model. This enables you to compare the labels that the model predicts with the actual known labels in the original dataset.

In this exercise, you're going to work through steps to extend the **Diabetes Training** pipeline as shown here:



Follow the steps below, using the image above for reference as you add and configure the required modules.

- Return to the **Designer** page and select the **Diabetes Training** pipeline.
- In the **Asset library** pane on the left, in **Component**, search for and place a **Split Data** module onto the canvas under the **Normalize Data** module. Then connect the *Transformed Dataset* (left) output of the **Normalize Data** module to the input of the **Split Data** module.
- Select the **Split Data** module, and configure its settings as follows:
    - **Splitting mode**: Split Rows
    - **Fraction of rows in the first output dataset**: 0.7
    - **Randomized split**: True
    - **Random seed**: 123
    - **Stratified split**: False

- In the **Asset library**, search for and place a **Train Model** module to the canvas, under the **Split Data** module. Then connect the *Results dataset1* (left) output of the **Split Data** module to the *Dataset* (right) input of the **Train Model** module.
- The model we're training will predict the **Diabetic** value, so select the **Train Model** module and modify its settings to set the **Label column** to **Diabetic**. The **Diabetic** label the model will predict is a class (0 or 1), so we need to train the model using a *classification* algorithm. Specifically, there are two possible classes, so we need a *binary classification* algorithm.
- In the **Asset library**, search for and place a **Two-Class Logistic Regression** module to the canvas, to the left of the **Split Data** module and above the **Train Model** module. Then connect its output to the *Untrained model* (left) input of the **Train Model** module. To test the trained model, we need to use it to *score* the validation dataset we held back when we split the original data - in other words, predict labels for the features in the validation dataset.
- In the **Asset library**, search for and place a **Score Model** module to the canvas, below the **Train Model** module. Then connect the output of the **Train Model** module to the *Trained model* (left) input of the **Score Model** module; and connect the *Results dataset2* (right) output of the **Split Data** module to the *Dataset* (right) input of the **Score Model** module.

Run the training pipeline

Now you're ready to run the training pipeline and train the model.

- Select **Configure & Submit** and run the pipeline using the existing experiment named **mslearn-diabetes-training**.
- Wait for the experiment run to finish. This may take 5 minutes or more.
- Check the status of the job by selecting **Jobs** under the **Assets**. From there, select the **mslearn-diabetes-training** experiment and then select the latest **Diabetes Training** job.
- On the new tab, right-click (Ctrl+click on a Mac) the **Score Model** module on the canvas, select **Preview data** and then select **Scored dataset** to view the results.
- Scroll to the right, and note that next to the **Diabetic** column (which contains the known true values of the label) there is a new column named **Scored Labels**, which contains the predicted label values, and a **Scored Probabilities** column containing a probability value between 0 and 1. This indicates the probability of a *positive* prediction, so probabilities greater than 0.5 result in a predicted label of ***1*** (diabetic), while probabilities between 0 and 0.5 result in a predicted label of ***0*** (not diabetic).
- Close the **Scored_dataset** tab.

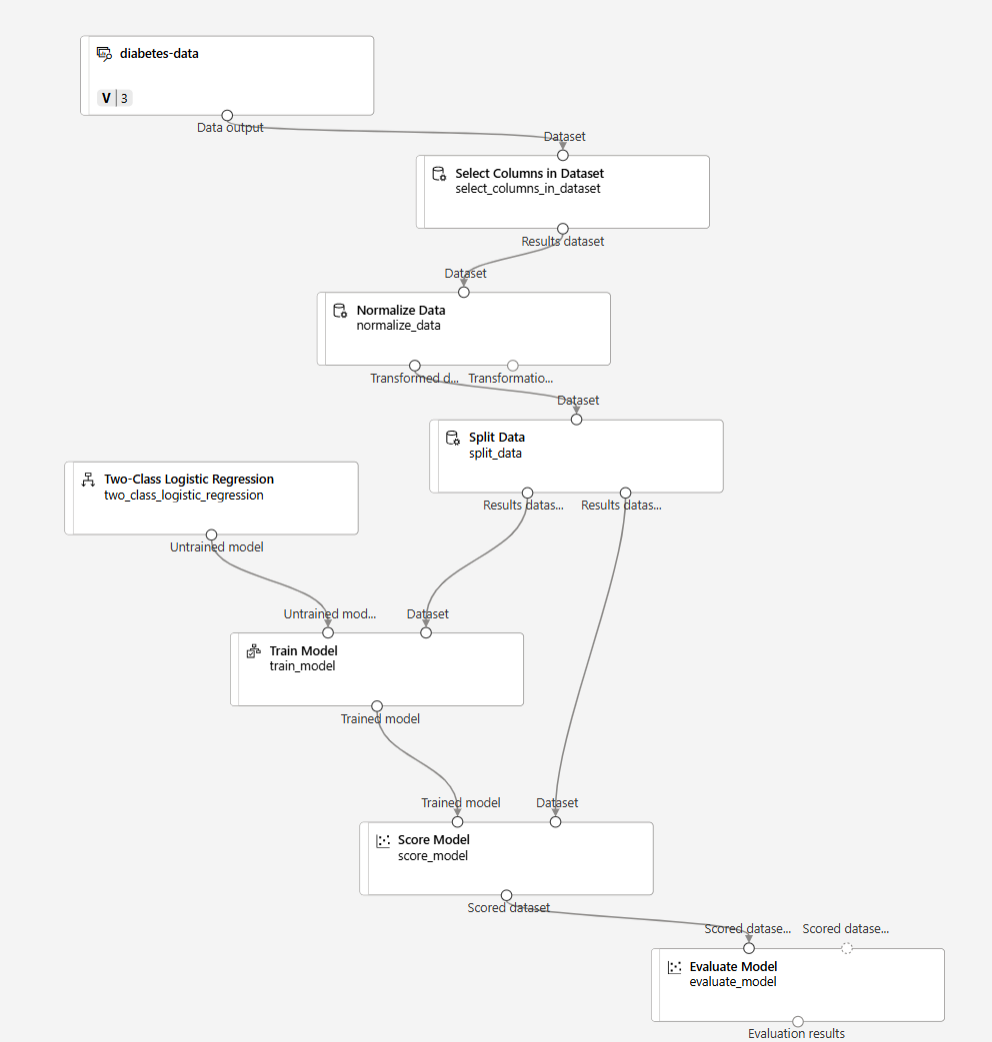
The model is predicting values for the **Diabetic** label, but how reliable are its predictions? To assess that, you need to evaluate the model.

The validation data you held back and used to score the model includes the known values for the label. So, to validate the model, you can compare the true values for the label to the label values that were predicted when you scored the validation dataset. Based on this comparison, you can calculate various metrics that describe how well the model performs.

Add an Evaluate Model module

- Return to **Designer** and open the **Diabetes Training** pipeline you created.

- In the **Asset library**, search for and place an **Evaluate Model** module to the canvas, under the **Score Model** module, and connect the output of the **Score Model** module to the *Scored dataset* (left) input of the **Evaluate Model** module.
- Ensure your pipeline looks like this:



- Select **Configure & Submit** and run the pipeline using the existing experiment named **mslearn-diabetes-training**.
- Wait for the experiment run to finish.
- Check the status of the job by selecting **Jobs** under the **Assets**. From there, select the **mslearn-diabetes-training** experiment and then select the latest **Diabetes Training** job.
- On the new tab, right-click (Ctrl+click on a Mac) the **Evaluate Model** module on the canvas, select **Preview data** then select **Evaluation results** to view the performance metrics. These metrics can help data scientists assess how well the model predicts based on the validation data.
- Scroll down to view the *confusion matrix* for the model. Observe the predicted and actual value counts for each possible class.
- Review the metrics to the left of the confusion matrix, which include:

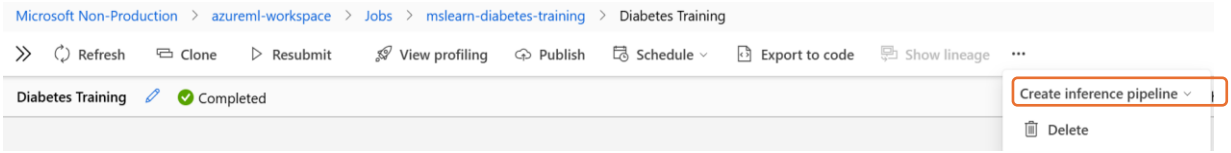
    - **Accuracy**: In other words, what proportion of diabetes predictions did the model get right?
    - **Precision**: In other words, out of all the patients that *the model predicted* as having diabetes, the percentage of time the model is correct.
    - **Recall**: In other words, out of all the patients *who actually have* diabetes, how many diabetic cases did the model identify correctly?
    - **F1 Score**

- Use the **Threshold** slider located above the list of metrics. Try moving the threshold slider and observe the effect on the confusion matrix. If you move it all the way to the left (0), the Recall metric becomes 1, and if you move it all the way to the right (1), the Recall metric becomes 0.
- Look above the Threshold slider at the **ROC curve** and **AUC** metric listed with the other metrics below. To get an idea of how this area represents the performance of the model, imagine a straight diagonal line from the bottom left to the top right of the ROC chart. This represents the expected performance if you just guessed or flipped a coin for each patient - you could expect to get around half of them right, and half of them wrong, so the area under the diagonal line represents an AUC of 0.5. If the AUC for your model is higher than this for a binary classification model, then the model performs better than a random guess.
- Close the **Evaluation_results** tab.

The performance of this model isn't all that great, partly because we performed only minimal feature engineering and pre-processing. You could try a different classification algorithm, such as **Two-Class Decision Forest**, and compare the results. You can connect the outputs of the **Split Data** module to multiple **Train Model** and **Score Model** modules, and you can connect a second **Score Model** module to the **Evaluate Model** module to see a side-by-side comparison. The point of the exercise is simply to introduce you to classification and the Azure Machine Learning designer interface, not to train a perfect model.

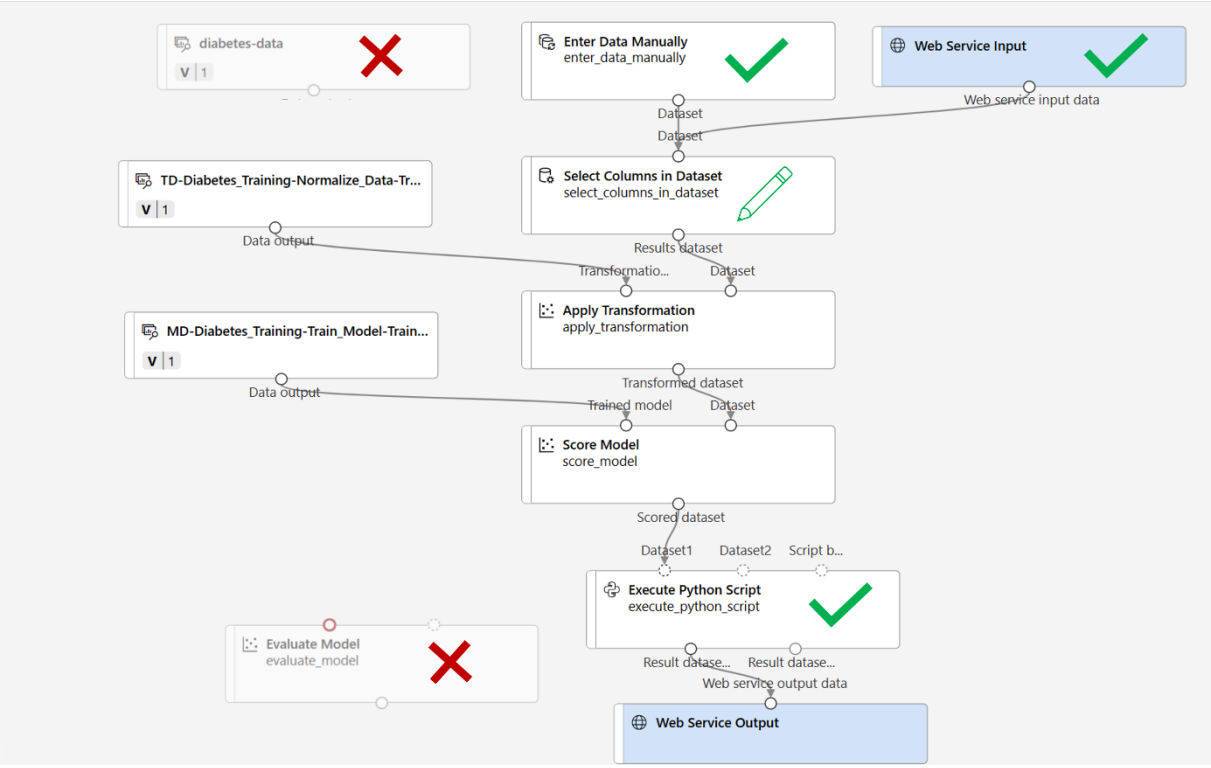
Create an inference pipeline

Locate the menu above the canvas and select **Create inference pipeline**. You may need to expand your screen to full and click on the three dots icon **...** on the top right hand corner of the screen in order to find **Create inference pipeline** in the menu.



- In the **Create inference pipeline** drop-down list, select **Real-time inference pipeline**. After a few seconds, a new version of your pipeline named **Diabetes Training-real time inference** will be opened.
- Rename the new pipeline to **Predict Diabetes**, then review the new pipeline. Some of the transformations and training steps are a part of this pipeline. The trained model will be used to score the new data. The pipeline also contains a web service output to return results.

You're going to make the following changes to the inference pipeline:



- Add a **web service input** component for new data to be submitted.
- Replace the **diabetes-data** dataset with an **Enter Data Manually** module that doesn't include the label column (**Diabetic**).
- Edit the columns selected in the **Select Columns in Dataset** module.
- Remove the **Evaluate Model** module.
- Insert an **Execute Python Script** module before the web service output to return only the patient ID, predicted label value, and probability.

- The pipeline does not automatically include a **Web Service Input** component for models created from custom data sets. Search for a **Web Service Input** component from the asset library and place it at the top of the pipeline. Connect the output of the **Web Service Input** component to the **Select Columns in Dataset** component that is already on the canvas.
- The inference pipeline assumes that new data will match the schema of the original training data, so the **diabetes-data** dataset from the training pipeline is included. However, this input data includes the **Diabetic** label that the model predicts, which is not included in new patient data for which a diabetes prediction hasn't yet been made. Delete this module and replace it with an **Enter Data Manually** module, containing the following CSV data, which includes feature values without labels for three new patient observations:

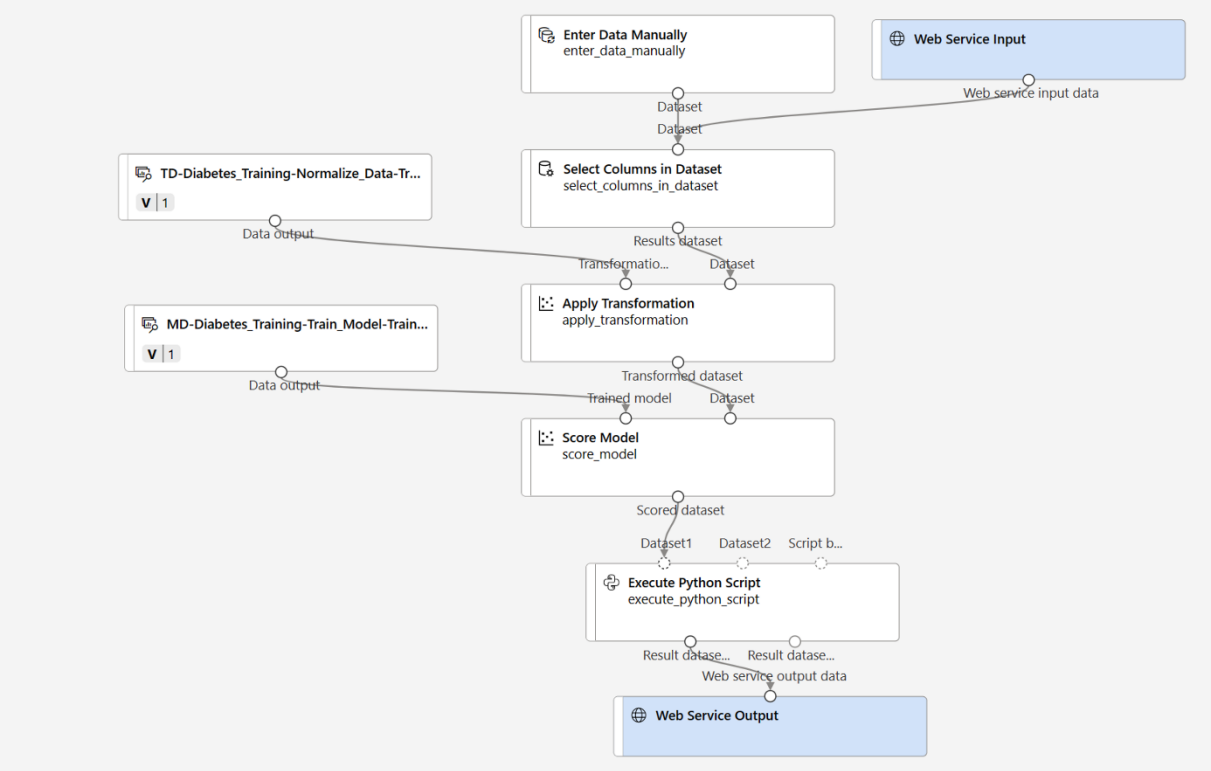
```
PatientID,Pregnancies,PlasmaGlucose,DiastolicBloodPressure,TricepsThickness,SerumInsulin
,BMI,DiabetesPedigree,Age
 1882185,9,104,51,7,24,27.36983156,1.350472047,43
 1662484,6,73,61,35,24,18.74367404,1.074147566,75
 1228510,4,115,50,29,243,34.69215364,0.741159926,59
```

- Connect the new **Enter Data Manually** module to the same **Dataset** input of the **Select Columns in Dataset** module as the **Web Service Input**.
- Edit the **Select Columns in Dataset** module. Remove **Diabetic** from the *Selected Columns*.

- The inference pipeline includes the **Evaluate Model** module, which isn't useful when predicting from new data, so delete this module.
- The output from the **Score Model** module includes all of the input features and the predicted label and probability score. To limit the output to only the prediction and probability:
    - Delete the connection between the **Score Model** module and the **Web Service Output**.
    - Add an **Execute Python Script** module, replacing all of the default python script with the following code (which selects only the **PatientID**, **Scored Labels** and **Scored Probabilities** columns and renames them appropriately):

```
import pandas as pd

def azureml_main(dataframe1=None, dataframe2=None):

    scored_results = dataframe1[['Scored Labels', 'Scored Probabilities']]

    scored_results.rename(columns={'Scored Labels': 'DiabetesPrediction',

                                   'Scored Probabilities': 'Probability'},

                          inplace=True)

    return scored_results
```

- Connect the output from the **Score Model** module to the *Dataset1* (left-most) input of the **Execute Python Script**, and connect the *Result dataset* (left) output of the **Execute Python Script** module to the **Web Service Output**.
- Verify that your pipeline looks similar to the following image:



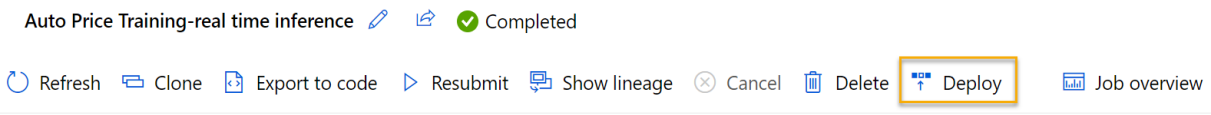
- Run the pipeline as a new experiment named **mslearn-diabetes-inference** on your compute cluster. The experiment may take a while to run.
- Return to the **Jobs** tab. From there, select the **mslearn-diabetes-inference** experiment and then select the **Predict Diabetes** job.
- When the pipeline has completed, select the **Execute Python Script** module. Select the **Preview data** and select **Result dataset** to see the predicted labels and probabilities for the three patient observations in the input data.

Your inference pipeline predicts whether patients are at risk for diabetes based on their features. Now you're ready to publish the pipeline so that client applications can use it.

After you've created and tested an inference pipeline for real-time inferencing, you can publish it as a service for client applications to use.

Deploy a service

- At the top of the **Predict Diabetes** job window, select **Deploy**.



- In the **Set up real-time endpoint** select **Deploy new real-time endpoint** and use the following settings:
    - **Name**: predict-diabetes
    - **Description**: Classify diabetes
    - **Compute type**: Azure Container Instance

- Select **Deploy** and wait for the web service to be deployed - this can take several minutes.

Lab 02 - Automated Machine Learning in Azure ML

Lab 21 – What is Automated ML?

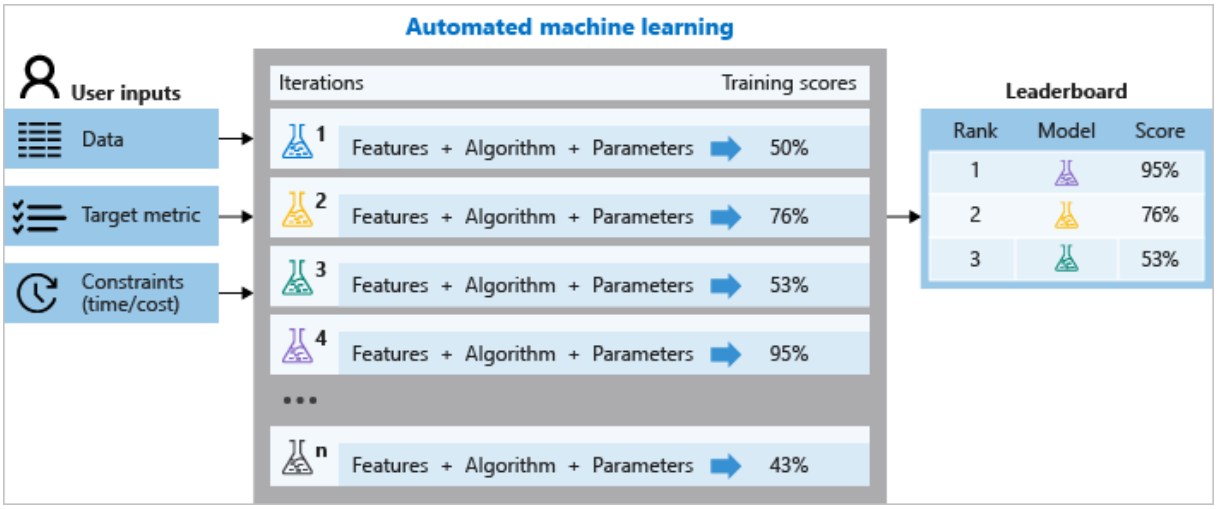
Definition

Automated machine learning, also referred to as automated ML or AutoML, is the process of automating the time-consuming, iterative tasks of machine learning model development. It allows data scientists, analysts, and developers to build ML models with high scale, efficiency, and productivity all while sustaining model quality.

How does it work?

During training, Azure Machine Learning creates many pipelines in parallel that try different algorithms and parameters for you. The service iterates through ML algorithms paired with feature selections, where each iteration produces a model with a training score. The better the score for the metric you want to optimize for, the better the model is considered to "fit" your data. It stops once it hits the exit criteria defined in the experiment.

The following diagram illustrates this process:



Lab 21 – Use automated machine learning to train a model

Automated machine learning enables you to try multiple algorithms and parameters to train multiple models and identify the best one for your data. In this exercise, you'll use a dataset of

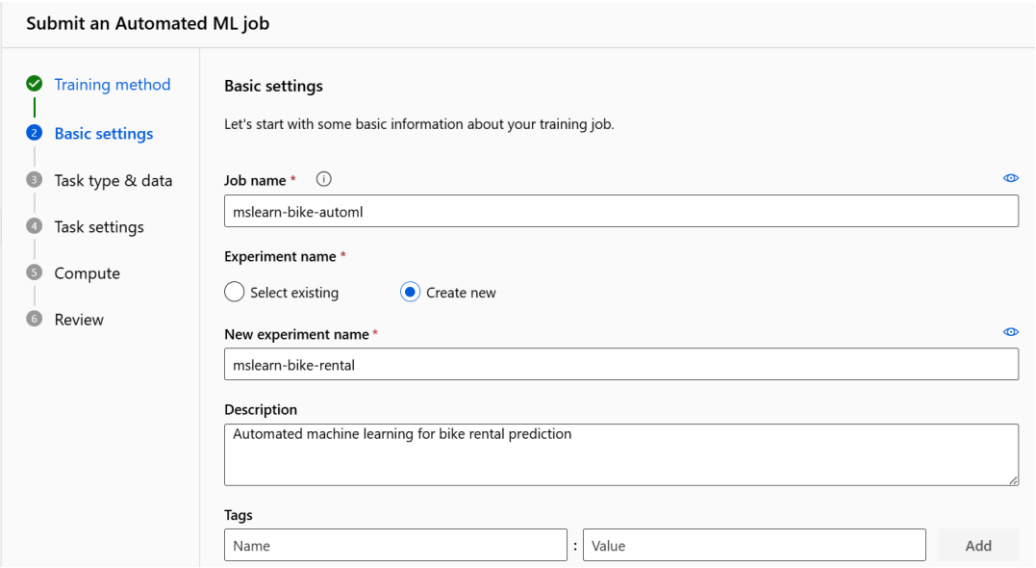
historical bicycle rental details to train a model that predicts the number of bicycle rentals that should be expected on a given day, based on seasonal and meteorological features.

Train a Regression model with Auto ML

- In Azure Machine Learning studio, view the **Automated ML** page (under **Authoring**).
- Create a new Automated ML job with the following settings, using **Next** as required to progress through the user interface:

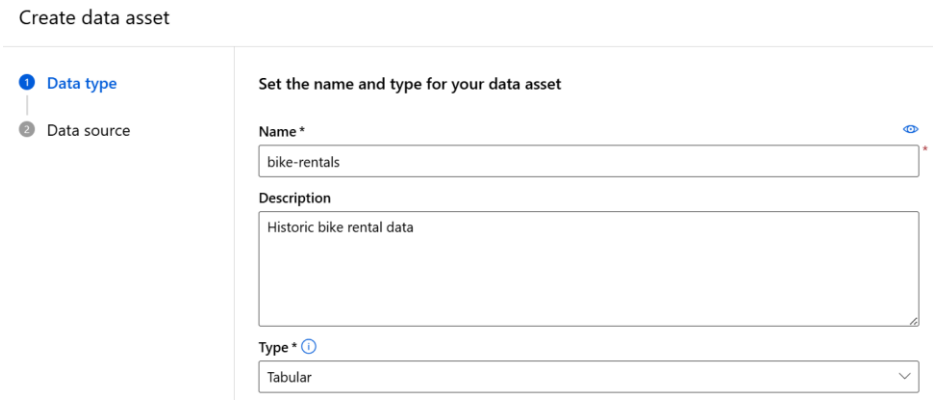
  Basic settings:

  - **Job name**: mslearn-bike-automl
  - **New experiment name**: mslearn-bike-rental
  - **Description**: Automated machine learning for bike rental prediction
  - **Tags**: *none*



  Task type & data:

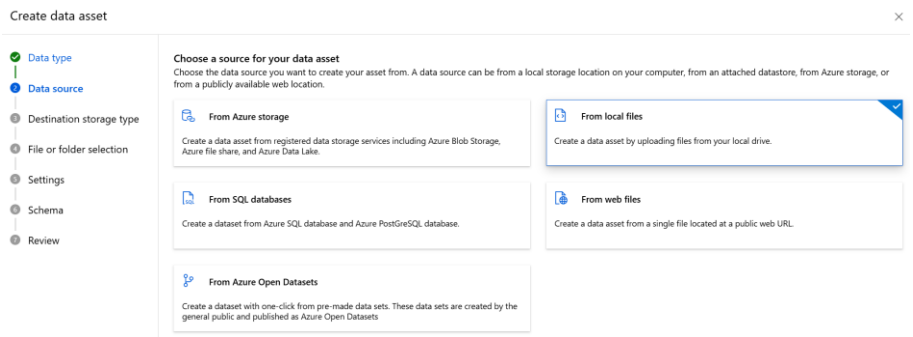
  - **Select task type**: Regression
  - **Select dataset**: Create a new dataset with the following settings:
    - **Data type**:
      - **Name**: bike-rentals
      - **Description**: Historic bike rental data
      - **Type**: Tabular



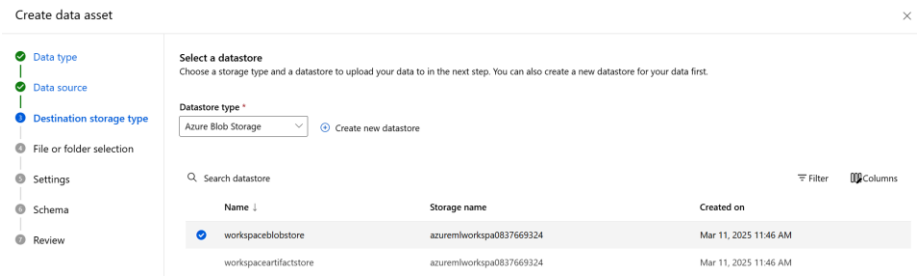
- **Data source:**

  Donwload the **“daily-bike-share.csv”** file from: [Workshop-Azure-Machine-Learning/data/daily-bike-share.csv at main · FranckGaillard/Workshop-Azure-Machine-Learning]

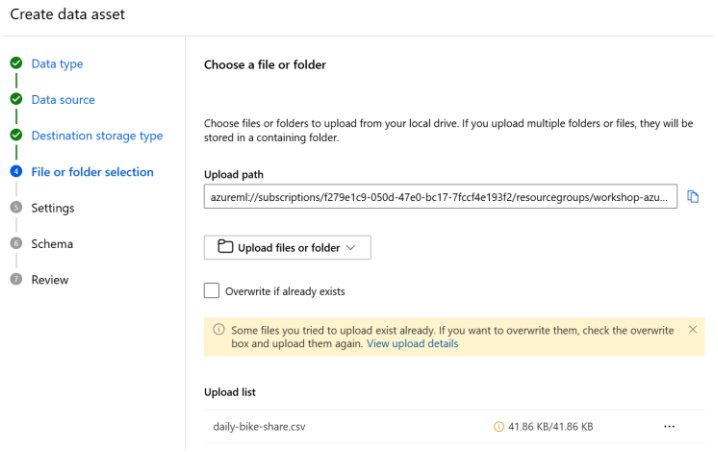
  - Select **From local files**



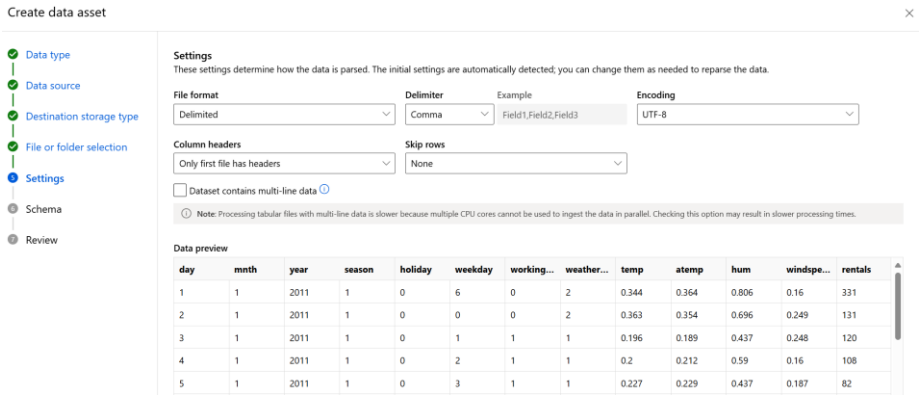
  - Click on Next



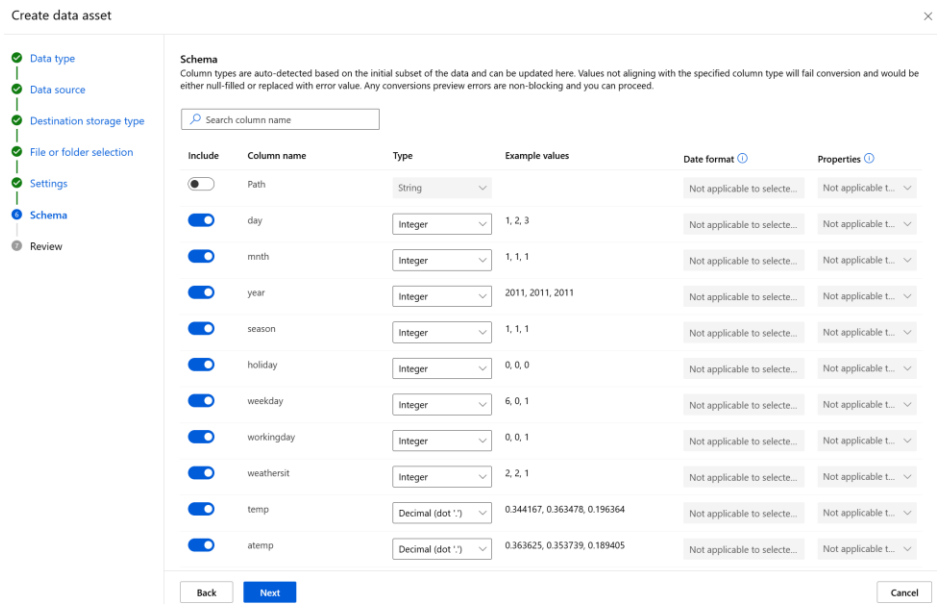
  - Click on Next
  - Upload the file you have just downloaded and click on Next



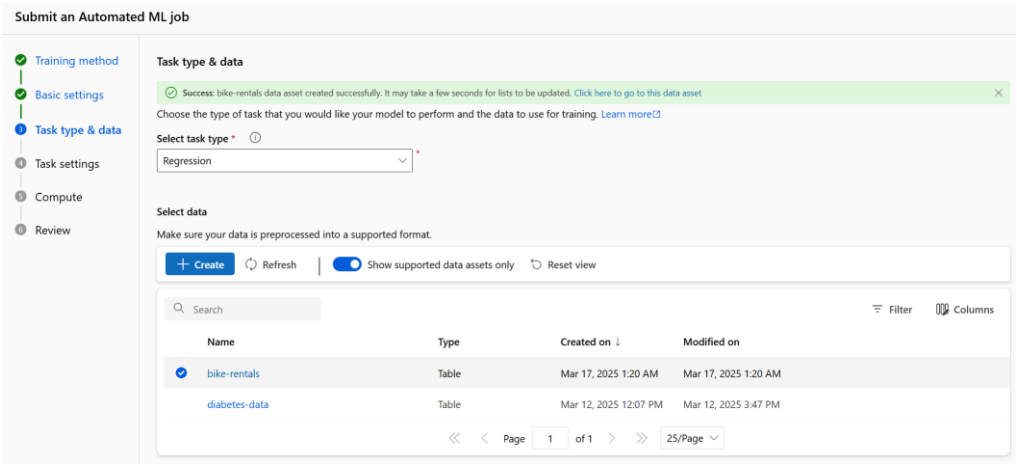
- **Settings**:
  - **File format**: Delimited
  - **Delimiter**: Comma
  - **Encoding**: UTF-8
  - **Column headers**: Only first file has headers
  - **Skip rows**: None
  - **Dataset contains multi-line data**: *do not select*



- **Schema**:
  - Include all columns other than **Path**
  - Review the automatically detected types



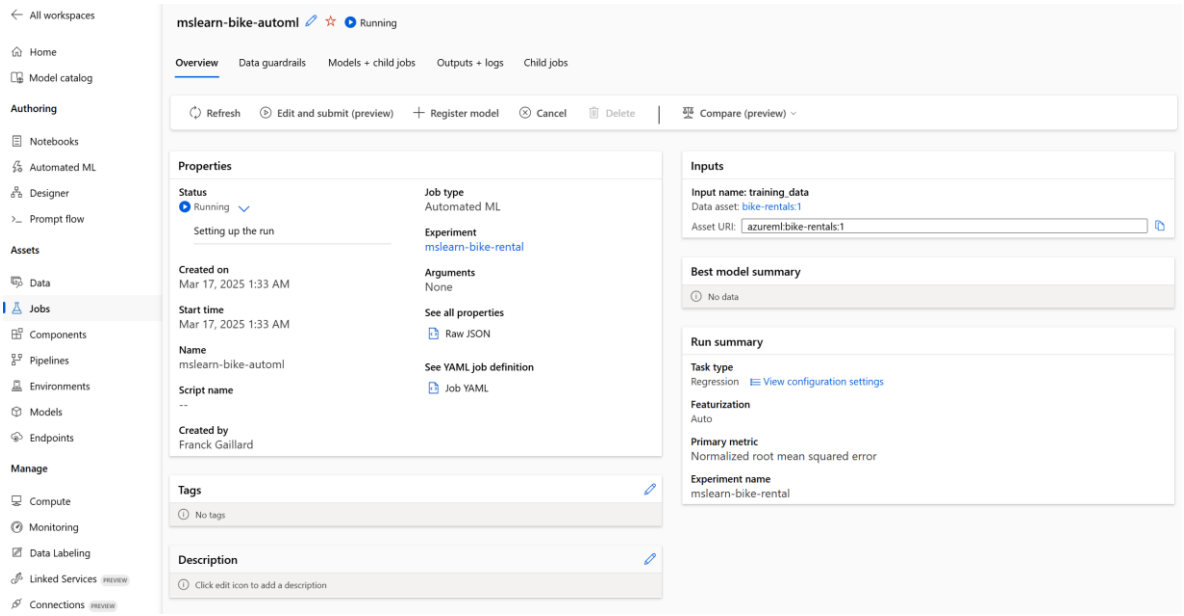
Select the **bike-rentals** dataset after you've created it.



Task settings:

- **Task type**: Regression
- **Dataset**: bike-rentals
- **Target column**: Rentals (integer)
- **Additional configuration settings**
  - **Primary metric**: Normalized root mean squared error
  - **Explain best model**: *Unselected*
  - **Use all supported models**: Unselected. *You'll restrict the job to try only a few specific algorithms.*
  - **Allowed models**: *Select only **RandomForest** and **LightGBM** — normally you'd want to try as many as possible, but each model added increases the time it takes to run the job.*
- **Limits**: *Expand this section*

        - **Max trials**: 3
        - **Max concurrent trials**: 3
        - **Max nodes**: 3
        - **Metric score threshold**: 0.85 (*so that if a model achieves a normalized root mean squared error metric score of 0.085 or less, the job ends.*)
        - **Timeout**: 15
        - **Iteration timeout**: 15
        - **Enable early termination**: *Selected*
    - **Validation and test**:
        - **Validation type**: Train-validation split
        - **Percentage of validation data**: 10
        - **Test dataset**: None
    - **Compute**:
        - **Select compute type**: Serverless
        - **Virtual machine type**: CPU
        - **Virtual machine tier**: Dedicated
        - **Virtual machine size**: Standard_DS3_V2
        - **Number of instances**: 1
- Submit the training job. It starts automatically.

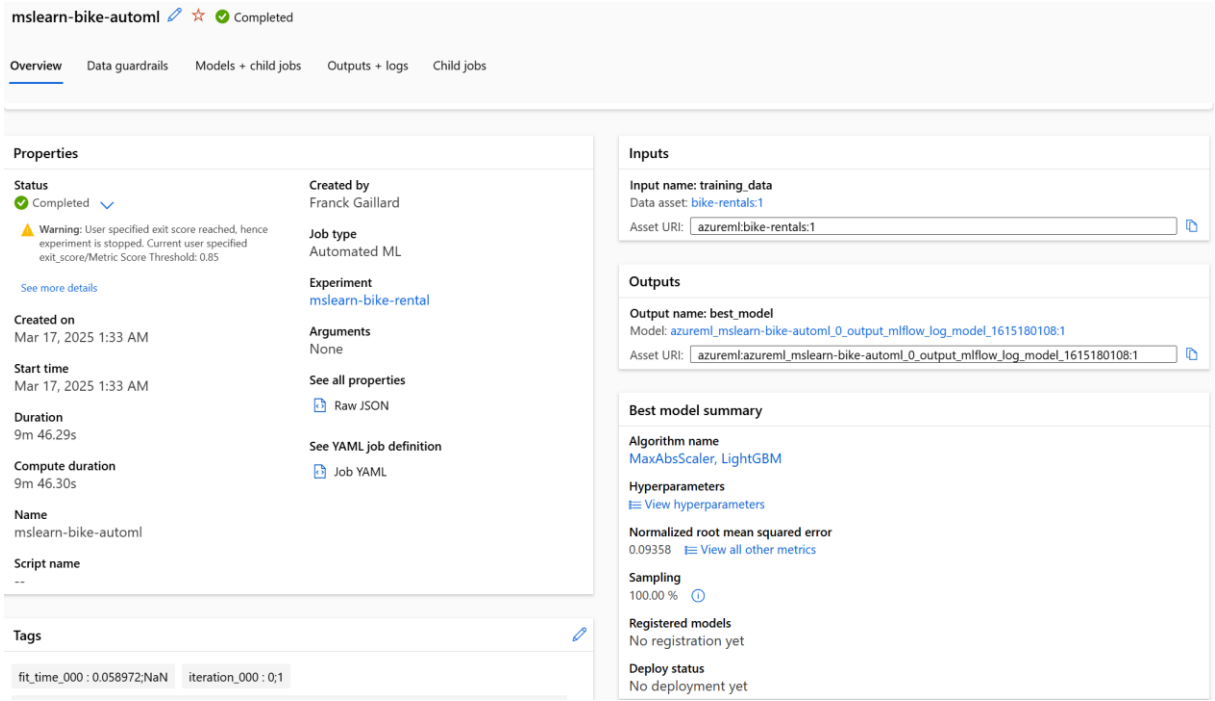


- Wait for the job to finish. It might take a while.

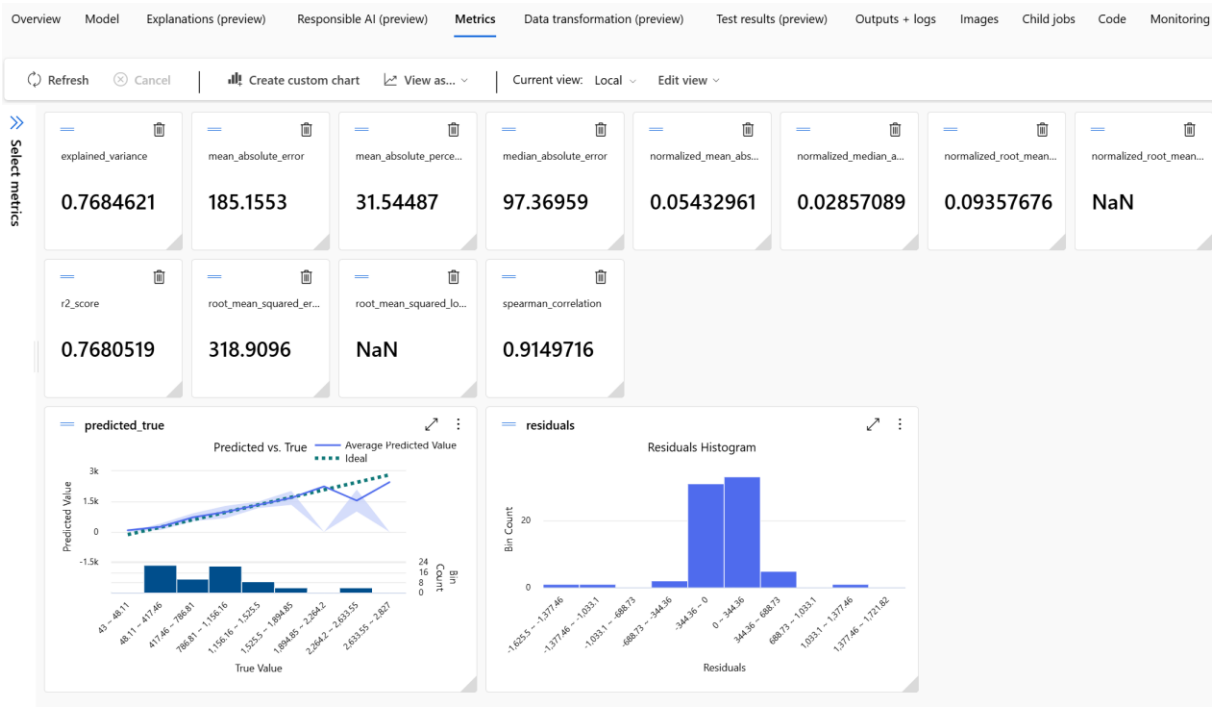
Review the best model

When the automated machine learning job has completed, you can review the best model it trained.

- On the **Overview** tab of the automated machine learning job, note the best model summary.

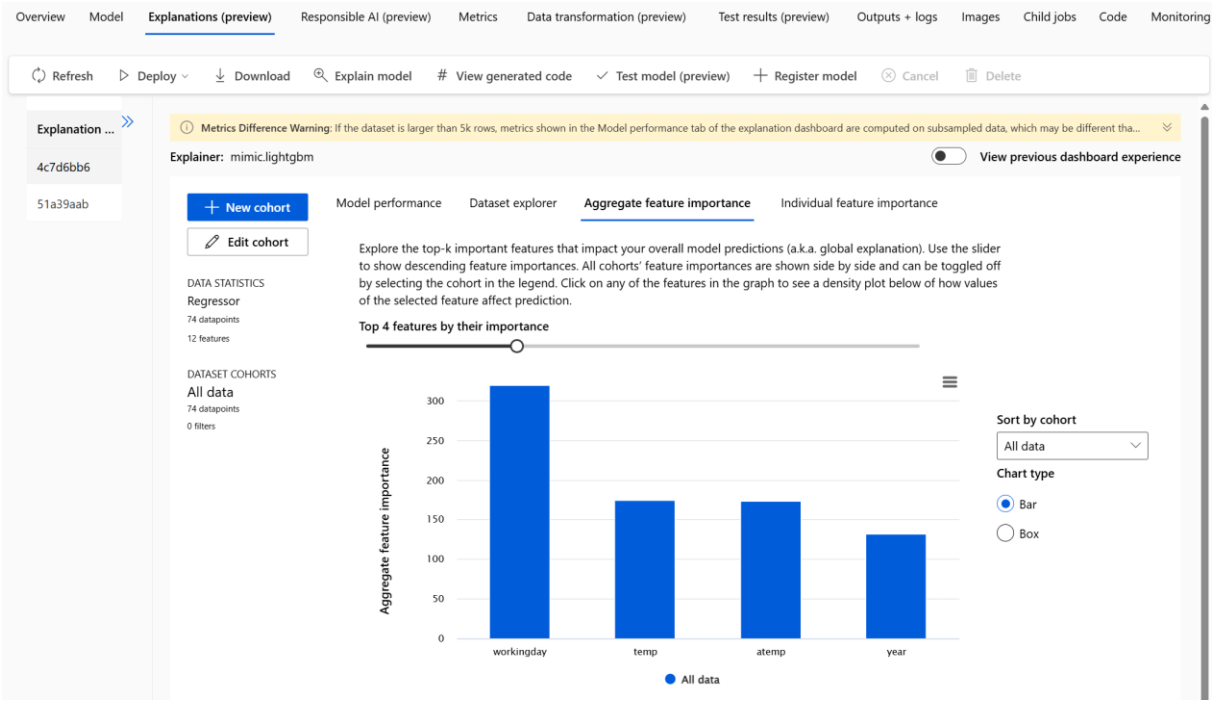


- Select the text under **Algorithm name** for the best model to view its details.
- Select the **Metrics** tab and select the **residuals** and **predicted_true** charts if they are not already selected. Navigate in the different metrics to explain the model.



Review the charts which show the performance of the model. The **residuals** chart shows the *residuals* (the differences between predicted and actual values) as a histogram. The **predicted_true** chart compares the predicted values against the true values.

- Navigate to **Explanations (preview)** tab:

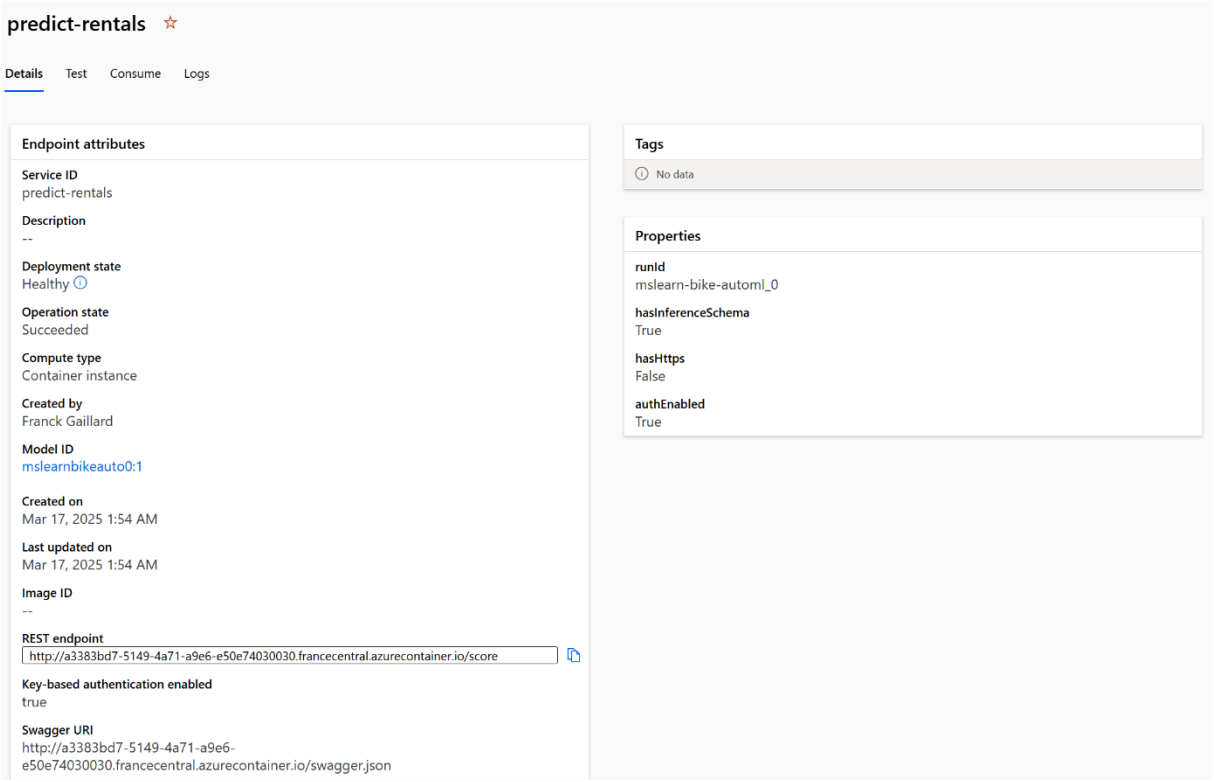


Deploy the model

- On the **Model** tab for the best model trained by your automated machine learning job, select **Deploy** and use the **Web service** option to deploy the model with the following settings:
    - **Name**: predict-rentals

- **Description**: Predict cycle rentals
- **Compute type**: Azure Container Instance
- **Enable authentication**: *Selected*

- Wait for the deployment to start - this may take a few seconds. The **Deploy status** for the **predict-rental** endpoint will be indicated in the main part of the page as *Running*.
- Wait for the **Deploy status** to change to *Succeeded*. This may take 5-10 minutes.

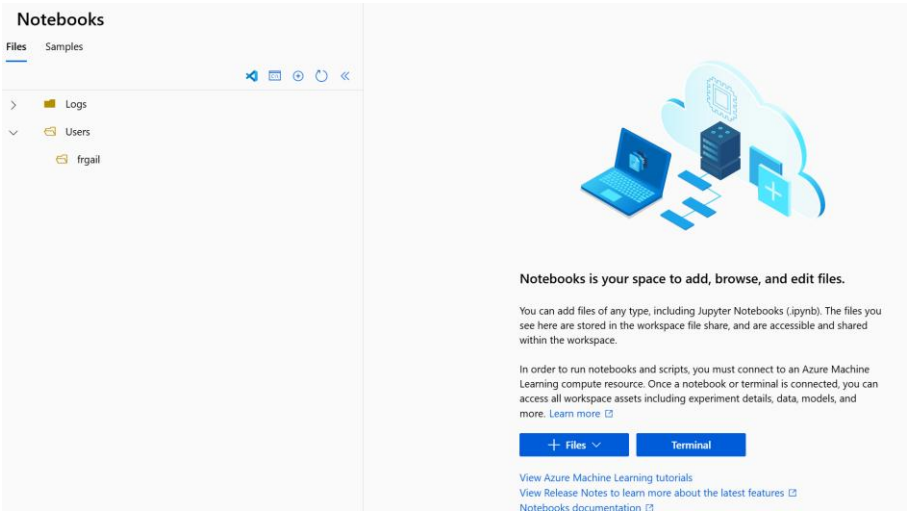


Lab 03 - Explore Notebooks in Azure ML

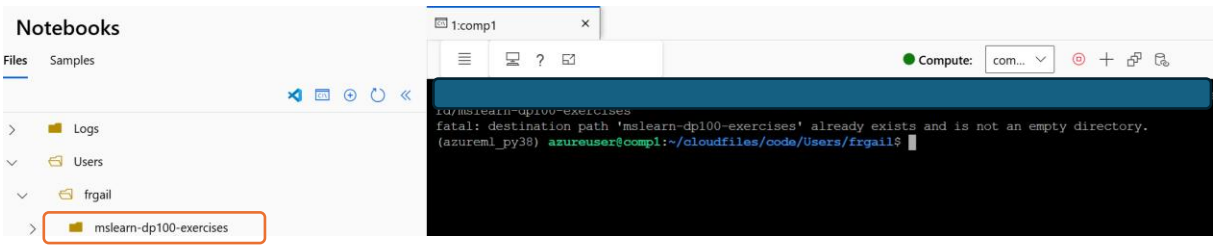
In Azure Machine Learning, notebooks provide a powerful and flexible environment for developing and experimenting with machine learning models. These notebooks, often based on Jupyter, allow data scientists and developers to write and execute code interactively, making it easier to explore data, build models, and visualize results. By leveraging Azure ML notebooks, users can seamlessly integrate with Azure ML services, access cloud resources, and collaborate on projects in real-time. This chapter will guide you through the process of creating, managing, and utilizing notebooks within Azure ML, enabling you to harness the full potential of this versatile tool for your machine learning workflows.

Clone the Github repository

- Click on Notebooks under Authoring and click on Terminal



- Clone this repo: FranckGaillard/mslearn-dp100-exercises: Lab files for Azure Machine Learning exercises to Azure ML Notebooks
    - On the terminal window, type:
        - git clone https://github.com/FranckGaillard/mslearn-dp100-exercises



- The folder “mslearn-dp100-exercises” has been copied on the VM

Introduction to the Azure ML SDK

The **Azure Machine Learning SDK for Python** is a powerful tool that allows data scientists and AI developers to build and run machine learning workflows using the Azure Machine Learning service. This SDK can be used in various Python environments, including Jupyter Notebooks, Visual Studio Code, or any preferred Python IDE.

- Open the notebook **“04 – Run Experiments.ipynb”**
- Execute step by step the notebook, explaining the different concepts such as “Experiment”, “Job”, “Run”.
- Show in Azure ML Studio the different artefacts created as a result of running the notebook and comment the results.