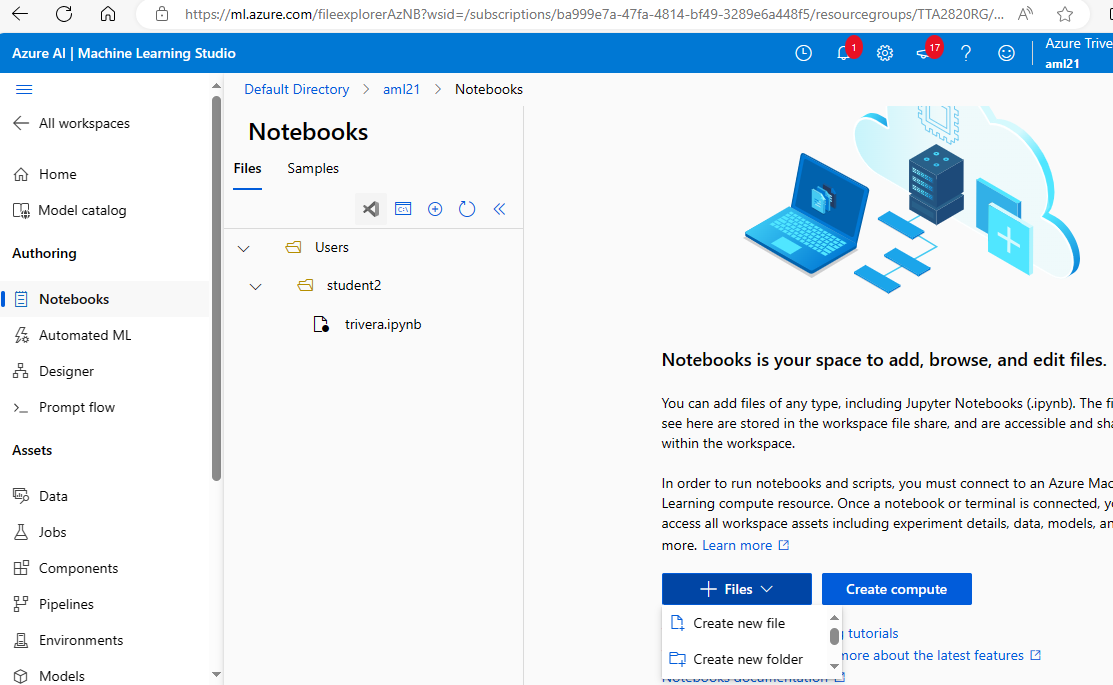
**Lab 2: Creating and Training a Machine Learning Model in Azure Machine Learning Studio**

**Objectives**

* Set up a notebook in Azure Machine Learning Studio.
* Create a serverless Spark compute.
* Build and train a machine learning model using external data.

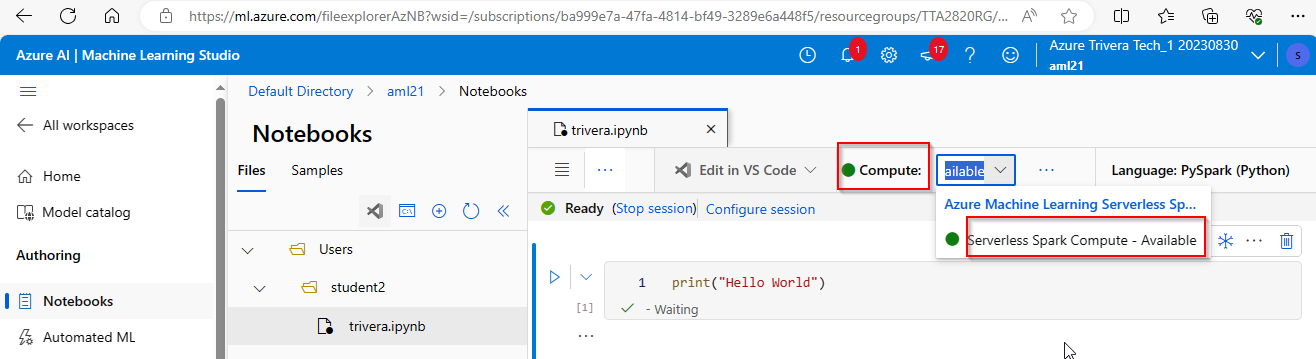
**Instructions**

1. **Setting Up a Notebook:**
   * Navigate to your Azure Machine Learning Studio workspace.
   * On the left-hand panel, click on “Notebooks” to access the notebook service.
   * Click “+ New file”, then select “Notebook” from the dropdown menu.
   * OR YOU CAN DO THIS:



* + Name your notebook (e.g., "ModelTrainingNotebook") and select a file location if necessary.
  + Click “Create” to open your new notebook.

1. **Creating a Serverless Spark Compute:**
   * In the Azure Machine Learning Studio, go to “Compute” in the right-hand panel.
   * Click on the down arrow for “Compute”.



* + Choose “Serverless Spark” as the compute type.
  + Click “Create” to initiate the serverless Spark compute. It may take a few minutes to provision.
  + In the first cell- type: print(“Hello World”) and press the play button to start the session.

1. **Building and Training the Model:**
   * In your newly created notebook, start by importing necessary libraries:

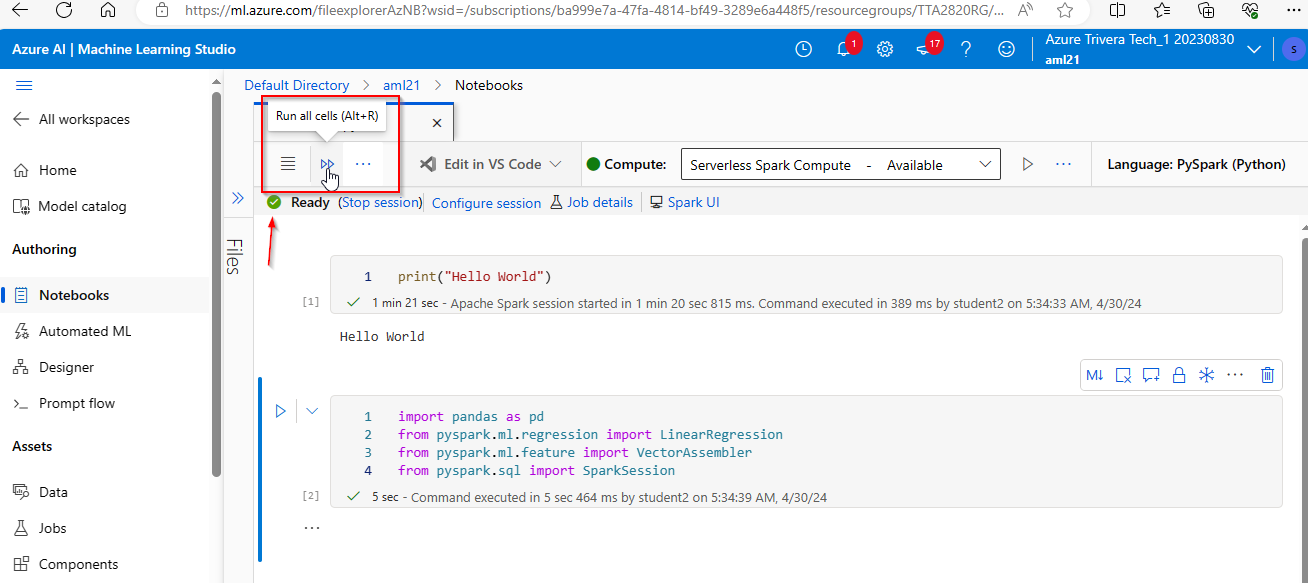
from pyspark.ml.feature import StringIndexer, VectorAssembler

from pyspark.ml.regression import LinearRegression

import pandas as pd

from pyspark.sql import SparkSession

Note – if you run into problems running – try to run all cells and make sure the session is running:



Load your data in the next cell using a URL:

# Load data

url = "https://raw.githubusercontent.com/fenago/datasets/main/iris.csv"

data = pd.read\_csv(url)

spark\_df = spark.createDataFrame(data)

Preprocess the data (assuming it's a regression task):

# Encode the target variable 'Flower'

indexer = StringIndexer(inputCol="Flower", outputCol="FlowerIndex")

indexed\_data = indexer.fit(spark\_df).transform(spark\_df)

# Assemble features

feature\_columns = spark\_df.columns[:-1]  # all columns except the last one (including 'Flower')

assembler = VectorAssembler(inputCols=feature\_columns, outputCol="features")

data\_ready = assembler.transform(indexed\_data)

Split the data into training and testing sets:

train\_data, test\_data = data\_ready.randomSplit([0.7, 0.3], seed=42)

Initialize and train the model:

# Initialize and fit model

lr = LinearRegression(featuresCol='features', labelCol='FlowerIndex')

model = lr.fit(train\_data)  # Ensure that train\_data now includes 'FlowerIndex'

Make Predictions on New Values:

from pyspark.sql import Row

# Example new data (make sure to provide realistic feature values based on your specific features)

new\_data = [

Row(\*\*{"Sepal Length": 5.1, "Sepal Width": 3.5, "Petal Length": 1.4, "Petal Width": 0.2}),

Row(\*\*{"Sepal Length": 6.7, "Sepal Width": 3.0, "Petal Length": 5.2, "Petal Width": 2.3})

]

# Create DataFrame from the new data

new\_df = spark.createDataFrame(new\_data)

# Transform the new data to include features column

new\_features = assembler.transform(new\_df)

# Make predictions

predictions = model.transform(new\_features)

predictions.select("features", "prediction").show()

Evaluate the Performance:

from pyspark.ml import Pipeline

from pyspark.ml.feature import StringIndexer, VectorAssembler

from pyspark.ml.regression import LinearRegression

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.sql import SparkSession

# Assume 'spark' is already started

# Assume data loading and preprocessing have been done similar to previous steps

# Split data into training and test sets might have been done already

# train\_data, test\_data = data.randomSplit([0.7, 0.3], seed=42)

# Assuming the pipeline model has already been fitted on the training data

# model = pipeline.fit(train\_data)

# Make sure to transform the test\_data using the same model to include 'FlowerIndex'

predictions = model.transform(test\_data)

# Now evaluate the model with the correct test set that includes the 'FlowerIndex'

evaluator = RegressionEvaluator(labelCol="FlowerIndex", predictionCol="prediction", metricName="rmse")

rmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

# Optionally evaluate more metrics

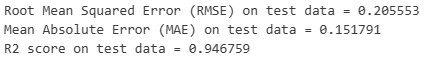
mae = evaluator.evaluate(predictions, {evaluator.metricName: "mae"})

r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

print("Mean Absolute Error (MAE) on test data = %g" % mae)

print("R2 score on test data = %g" % r2)

Interpretation:



1. **Root Mean Squared Error (RMSE) on test data = 0.205553**
   * **What it means:** This number tells us how far, on average, the model's predictions are from the actual outcomes. In this case, the model's predictions are usually about 0.21 units off from the true value.
   * **How to interpret:** Lower values are better because it means the model's predictions are closer to the actual values. An RMSE of 0.21 suggests the model is quite accurate, considering the values it predicts.
2. **Mean Absolute Error (MAE) on test data = 0.151791**
   * **What it means:** This is another way to measure error. It shows the average difference between the predicted values and the actual values, but without considering the direction of the error (i.e., it doesn't matter if the prediction was too high or too low).
   * **How to interpret:** Just like RMSE, a lower MAE is better. A MAE of about 0.15 means that, on average, the model's predictions are about 0.15 units away from the actual values.
3. **R2 score on test data = 0.946759**
   * **What it means:** This is a statistical measure that shows how well the variations in the data are explained by the model. An R2 score of 1 means the model explains all the variability of the data around its mean, perfectly.
   * **How to interpret:** Closer to 1 is better. An R2 score of 0.95 is excellent, suggesting that 95% of the variance in the data is explained by the model. This means the model does a great job in understanding how the input data (like the sepal length, sepal width, etc.) influences the predicted value (type of iris flower).

These metrics tell us that the model performs very well with the iris dataset, accurately predicting the type of iris flower based on measurements like sepal length and width, and petal length and width.