ESPECIALIZACIÓN EN MACHINE LEARNING ENGINEER

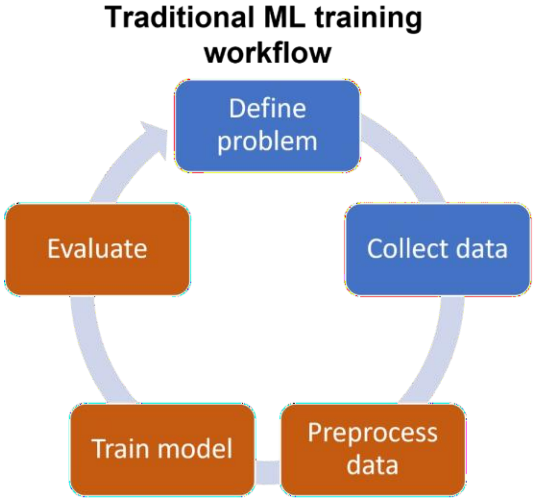
Tema: Fundamentos de MLE

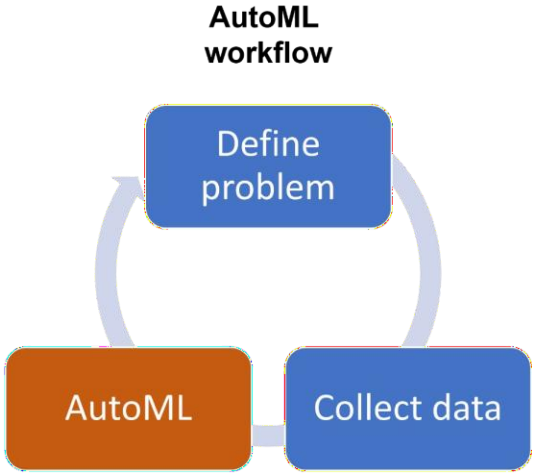
Sesión 4

Docente: Arnaldo Alvarado

¿Qué es AutoML?

El Machine Learning Automatizado (**AutoML**) representa un conjunto de procesos y técnicas que permiten la **automatización de tareas** relacionadas con el aprendizaje automático. Su objetivo es hacer que el Machine Learning sea más accesible, reduciendo o eliminando la necesidad de **expertise especializado en ciertas etapas** del proceso de modelado. Esto incluye desde la preparación de los datos, selección de modelos, hasta la optimización de hiperparámetros.





PyCaret: Conceptos

PyCaret es una **biblioteca de AutoML en Python** que ofrece una interfaz de alto nivel para varias bibliotecas y frameworks de ML, como scikit-learn, XGBoost, LightGBM, y otros. Está diseñada para **automatizar el flujo de trabajo de Machine Learning**, permitiendo a los usuarios pasar de la preparación de datos a la implementación de modelos en pocas líneas de código.

Simplicidad: PyCaret está diseñado para ser **simple y fácil de usar**. Los usuarios pueden realizar tareas complejas de ML con un **mínimo esfuerzo y conocimiento técnico**.
Eficiencia: Permite a los usuarios **experimentar y comparar docenas de modelos** y técnicas de preprocesamiento de forma rápida y eficiente.

Flexible: Aunque es de alto nivel y fácil de usar, PyCaret **no sacrifica la flexibilidad**. Los usuarios pueden **personalizar el pipeline** de ML según sea necesario.
Integrado: Soporta tareas de clasificación, regresión, clustering, detección de anomalías, procesamiento de lenguaje natural (NLP), y más, todo dentro de un marco unificado.

Instalación

```
pip install pycaret
```

PyCaret: Conceptos

Preparación de datos

La calidad de los datos de entrada es **fundamental para el éxito de cualquier proyecto** de Machine Learning. La preparación de los datos implica varios pasos críticos para asegurar que tus **modelos funcionen correctamente** y sean capaces de generar **predicciones precisas.**

Limpieza de datos: Elimina o corrige registros corruptos o inexactos de tus datos. Esto puede incluir el manejo de **valores faltantes, la eliminación de duplicados, o la corrección de errores** de formato.

Transformación de datos: Convierte los datos a un **formato adecuado** para el modelado. Esto puede implicar la **normalización o estandarización** de las características numéricas, la **codificación de variables categóricas**, y la **transformación de fechas** y otros tipos de datos en formatos útiles.

Selección de features: Identifica las **características más relevantes** para tu problema. Esto puede implicar el **análisis de la correlación** entre diferentes características y la etiqueta objetivo, así como la eliminación de características redundantes o irrelevantes.

División de datos: Divide tus datos en **conjuntos de entrenamiento y prueba** (y, opcionalmente, de validación). Esto es crucial para evaluar el rendimiento de tus modelos de manera objetiva, entrenándolos en un conjunto de datos y probándolos en otro distinto.

PyCaret **simplifica muchos de estos pasos con su función setup**, la cual realiza automáticamente muchas tareas de preprocesamiento y validación de datos, permitiéndote especificar tus requisitos y preferencias a través de parámetros. Sin embargo, es **importante realizar un análisis exploratorio de datos (EDA)** antes de este paso para entender tus datos y determinar las transformaciones y limpiezas necesarias.

PyCaret: AutoML

Paso 1: Cargar dataset

PyCaret ofrece datasets de muestra que pueden ser cargados directamente. Para cargar el dataset de Iris:

```
from pycaret.datasets import get_data
dataset = get_data('iris')
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Si quieres importar tus propios datos, puedes utilizar pandas y luego pasar el DataFrame a PyCaret:

```
import pandas as pd
dataset = pd.read_csv('mi_dataset.csv')
```

PyCaret: AutoML

Paso 2: Configuración del experimento

Para inicializar el entorno en PyCaret y configurar el experimento, usamos setup(). Este paso es crucial ya que prepara los datos para el modelado y establece las bases del experimento.

```
from pycaret.classification import setup
exp_clf101 = setup(data              = dataset,
                   target            = 'species',
                   session_id        = 123,
                   train_size        = 0.7,
                   numeric_features = ["sepal_length",
"sepal_width", "petal_length", "petal_width"])
```

https://pycaret.readthedocs.io/en/stable/api/classification.html

	Description	Value
0	Session id	123
1	Target	species
2	Target type	Multiclass
3	Target mapping	Iris-setosa: 0, Iris-versicolor: 1, Iris-virginica: 2
4	Original data shape	(150, 5)
5	Transformed data shape	(150, 5)
6	Transformed train set shape	(105, 5)
7	Transformed test set shape	(45, 5)
8	Numeric features	4
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Fold Generator	StratifiedKFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	clf-default-name
19	USI	079f

Elaborado por: José Enrique Mejía Gamarra

PyCaret: AutoML

Paso 3: Comparación de modelos

Para comparar diferentes modelos y encontrar el de mejor rendimiento según una métrica determinada, utilizamos compare_models().

```
from pycaret.classification import compare_models

best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.9718	0.0000	0.9718	0.9780	0.9712	0.9573	0.9609	0.8210
knn	K Neighbors Classifier	0.9718	0.0000	0.9718	0.9780	0.9712	0.9573	0.9609	0.4020
qda	Quadratic Discriminant Analysis	0.9718	0.0000	0.9718	0.9780	0.9712	0.9573	0.9609	0.0150
lda	Linear Discriminant Analysis	0.9718	0.0000	0.9718	0.9780	0.9712	0.9573	0.9609	0.0150
lightgbm	Light Gradient Boosting Machine	0.9536	0.0000	0.9536	0.9634	0.9528	0.9298	0.9356	0.1000
nb	Naive Bayes	0.9445	0.0000	0.9445	0.9525	0.9438	0.9161	0.9207	0.0110
et	Extra Trees Classifier	0.9445	0.0000	0.9445	0.9586	0.9426	0.9161	0.9246	0.0530
gbc	Gradient Boosting Classifier	0.9355	0.0000	0.9355	0.9416	0.9325	0.9023	0.9083	0.0990
dt	Decision Tree Classifier	0.9264	0.0000	0.9264	0.9502	0.9201	0.8886	0.9040	0.0100
rf	Random Forest Classifier	0.9264	0.0000	0.9264	0.9343	0.9232	0.8886	0.8956	0.0720
ada	Ada Boost Classifier	0.9155	0.0000	0.9155	0.9401	0.9097	0.8720	0.8873	0.0370
ridge	Ridge Classifier	0.8227	0.0000	0.8227	0.8437	0.8186	0.7320	0.7454	0.0110
svm	SVM - Linear Kernel	0.7618	0.0000	0.7618	0.6655	0.6888	0.6333	0.7048	0.0150
dummy	Dummy Classifier	0.2864	0.0000	0.2864	0.0822	0.1277	0.0000	0.0000	0.0100

PyCaret: AutoML

Paso 4: Creación de un modelo específico

Si tienes interés en un modelo específico, puedes crearlo directamente. Por ejemplo, para entrenar un modelo de Árbol de Decisión:

```
from pycaret.classification import
create_model
dt = create_model('lr')
```

'lr' se refiere a Logistic Regression. Cada tipo de modelo tiene una identificación específica en PyCaret.

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8182	0.0000	0.8182	0.8788	0.8061	0.7250	0.7642
1	0.9091	0.0000	0.9091	0.9273	0.9076	0.8625	0.8735
2	0.9091	0.0000	0.9091	0.9273	0.9076	0.8625	0.8735
3	0.7273	0.0000	0.7273	0.8442	0.6826	0.5875	0.6674
4	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	0.9000	0.0000	0.9000	0.9250	0.8971	0.8485	0.8616
6	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9264	0.0000	0.9264	0.9502	0.9201	0.8886	0.9040
Std	0.0893	0.0000	0.0893	0.0552	0.1011	0.1351	0.1119

PyCaret: AutoML

Paso 5: Optimización de hiperparámetros

Para optimizar los hiperparámetros de un modelo y mejorar su rendimiento, utilizamos tune_model().

```
from pycaret.classification import tune_model
tuned_dt = tune_model(dt, optimize = 'Accuracy')
```

Esto intentará diferentes combinaciones de hiperparámetros para el Árbol de Decisión (dt) y se centrará en optimizar la precisión ('Accuracy').

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.9091	0.0000	0.9091	0.9273	0.9076	0.8625	0.8735
2	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	0.7273	0.0000	0.7273	0.8442	0.6826	0.5875	0.6674
4	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	0.9000	0.0000	0.9000	0.9250	0.8971	0.8485	0.8616
6	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9536	0.0000	0.9536	0.9696	0.9487	0.9298	0.9402
Std	0.0843	0.0000	0.0843	0.0510	0.0967	0.1276	0.1049

PyCaret: AutoML

Paso 6: Visualización del modelo

PyCaret facilita la visualización de diferentes aspectos de un modelo entrenado. Por ejemplo, para visualizar la matriz de confusión del modelo optimizado:

```
from pycaret.classification import plot_model
plot_model(tuned_dt, plot = 'confusion_matrix')
```

PyCaret soporta una variedad de tipos de gráficas que pueden ser especificadas con el parámetro plot. Algunas opciones incluyen 'auc' para la curva ROC, 'feature' para la importancia de las características, entre otras.

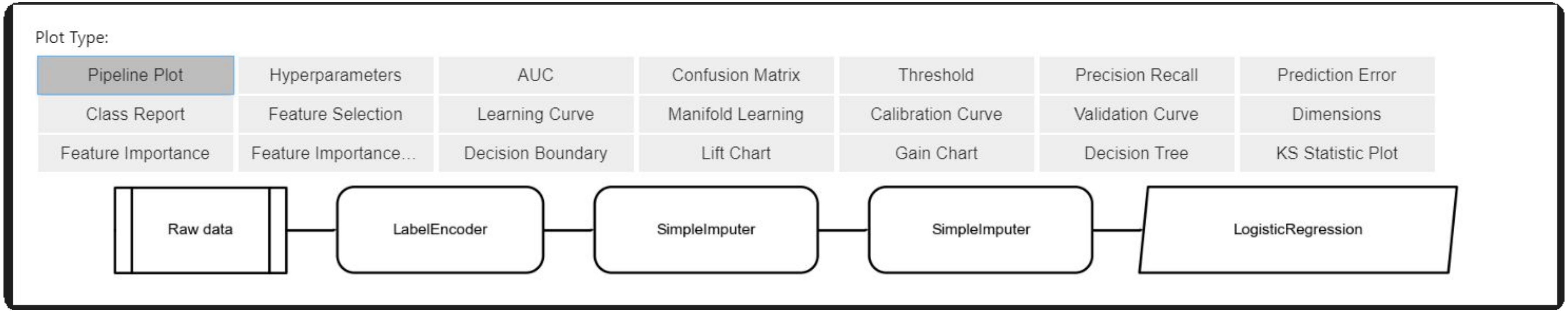
PyCaret: AutoML

Paso 7: Evaluación exhaustiva del modelo

Para una evaluación más detallada que incluya varias métricas y gráficas, PyCaret ofrece evaluate_model.

```
from pycaret.classification import plot_model
plot_model(tuned_dt, plot = 'confusion_matrix')
```

Esto abrirá una interfaz de usuario en tu navegador o Jupyter notebook (dependiendo de tu entorno de trabajo) que te permitirá explorar diferentes aspectos del modelo seleccionado.



PyCaret: AutoML

Paso 8: Finalizar el modelo

Una vez que estés satisfecho con el rendimiento del modelo, puedes "finalizarlo". Esto entrena el modelo en el conjunto completo de datos (incluyendo el conjunto de prueba).

```
from pycaret.classification import finalize_model
final_dt = finalize_model(tuned_dt)
```

El modelo finalizado final_dt está listo para ser usado en producción o para hacer predicciones sobre nuevos datos.

Paso 9: Realizar predicciones

Con el modelo finalizado, puedes hacer predicciones sobre nuevos datos. Para demostración, vamos a predecir las etiquetas del mismo conjunto de datos de Iris:

```
from pycaret.classification import predict_model
predictions = predict_model(final_dt, data=dataset)
```

“predictions” incluirá las etiquetas predichas junto con los datos originales.

PyCaret: AutoML

Paso 10: Guardar y cargar modelos

Finalmente, puedes guardar tu modelo para uso futuro y luego cargarlo cuando sea necesario.

Guardar modelo

```
from pycaret.classification import save_model
save_model(final_dt, 'final_dt_model_iris')
```

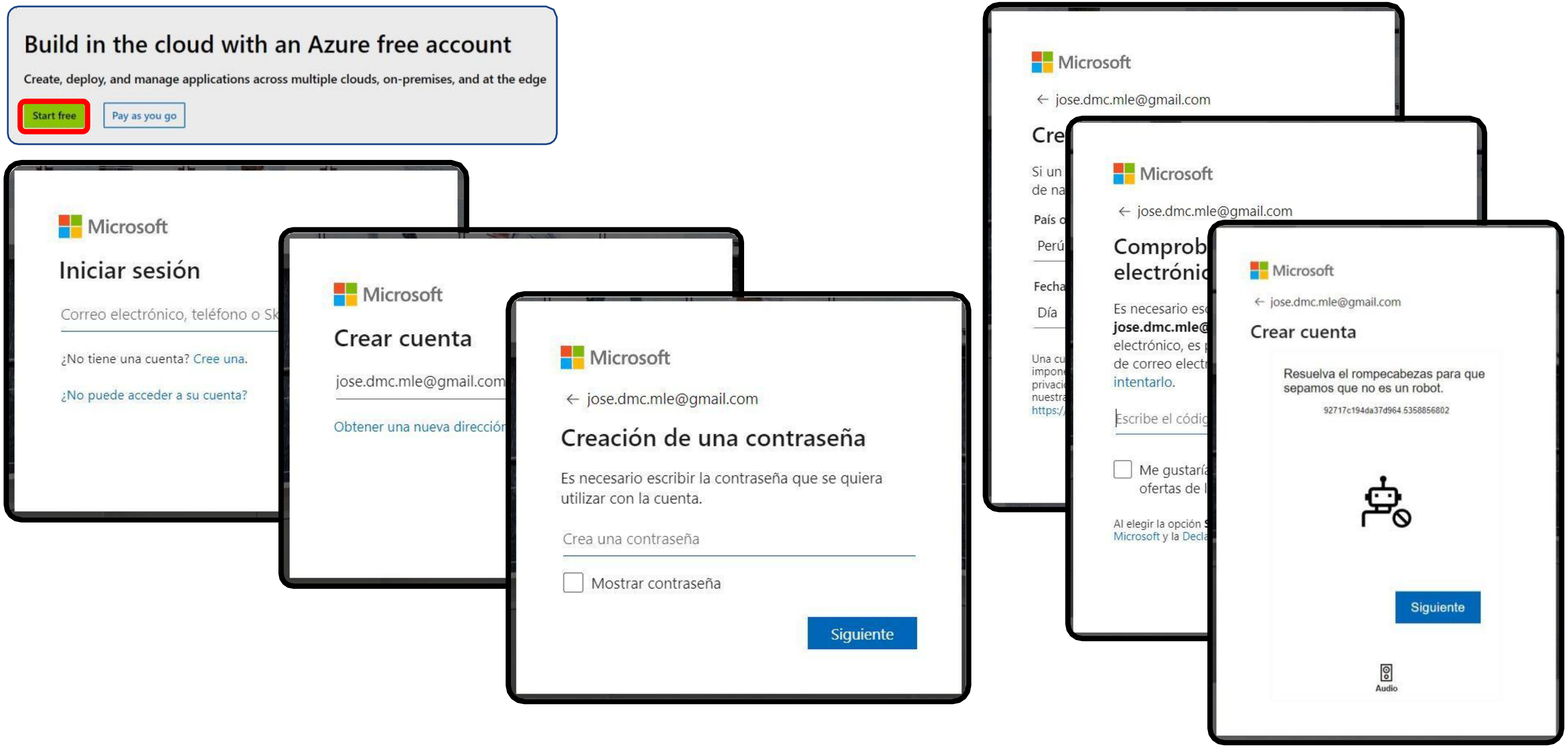
Cargar modelo

```
from pycaret.classification import load_model
loaded_model = load_model('final_dt_model_iris')
```

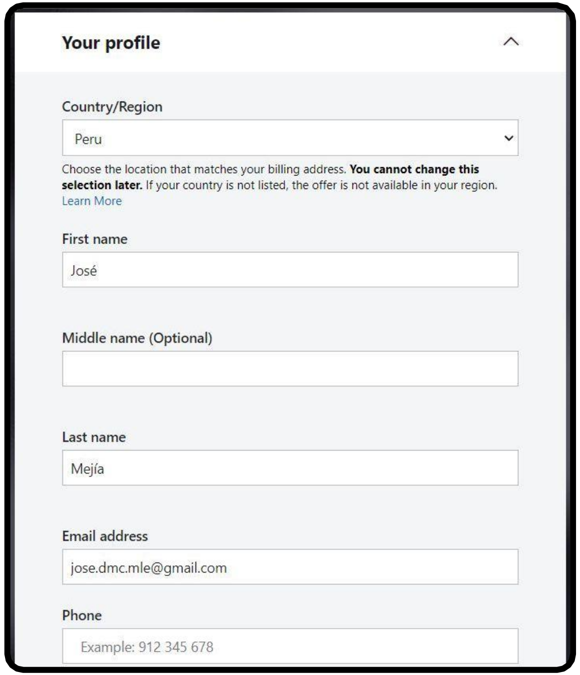
El modelo guardado final_dt_model_iris puede ser cargado en cualquier momento para hacer predicciones sobre nuevos conjuntos de datos sin necesidad de reentrenar el modelo.

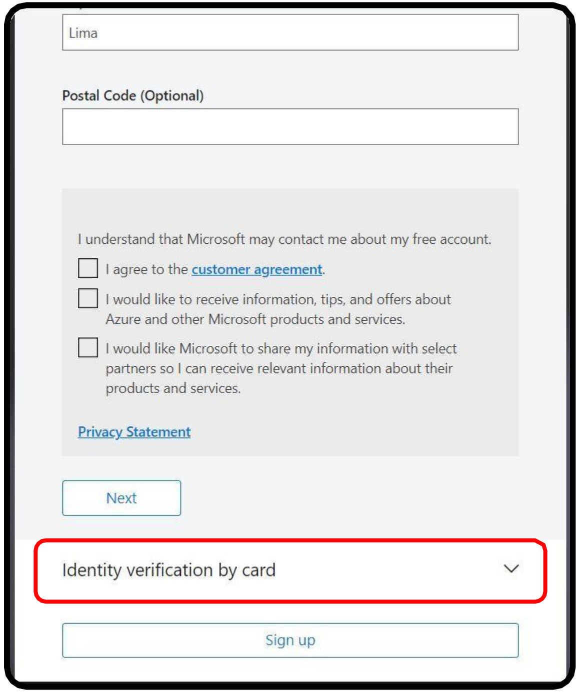
Elaborado por: José Enrique Mejía Gamarra

Azure: Crear cuenta en Azure



Azure: Crear cuenta de Azure





Azure

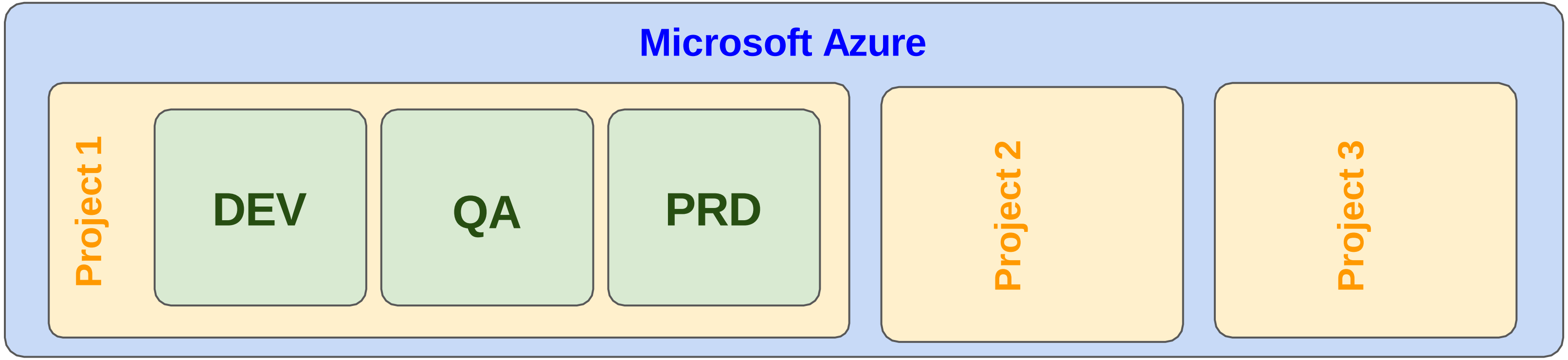
Azure: Conceptos

Subscription

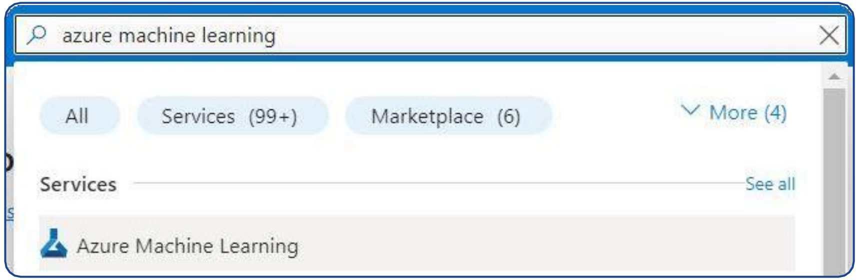
Una suscripción en Azure representa un **acuerdo con Microsoft** para utilizar sus servicios de nube. Esencialmente, actúa como un **contenedor en el que se alojan los recursos** de Azure utilizados. Al crear una suscripción, se **acuerda pagar por los recursos** que consume o por un nivel de servicio específico, según el tipo de suscripción que elija.

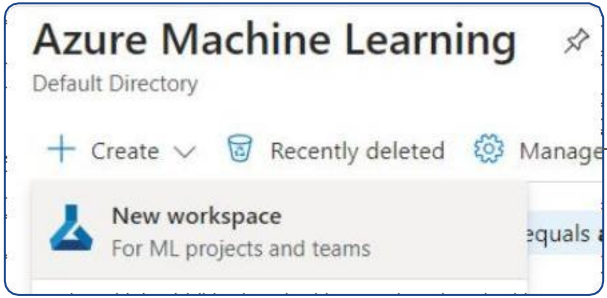
Resource Group

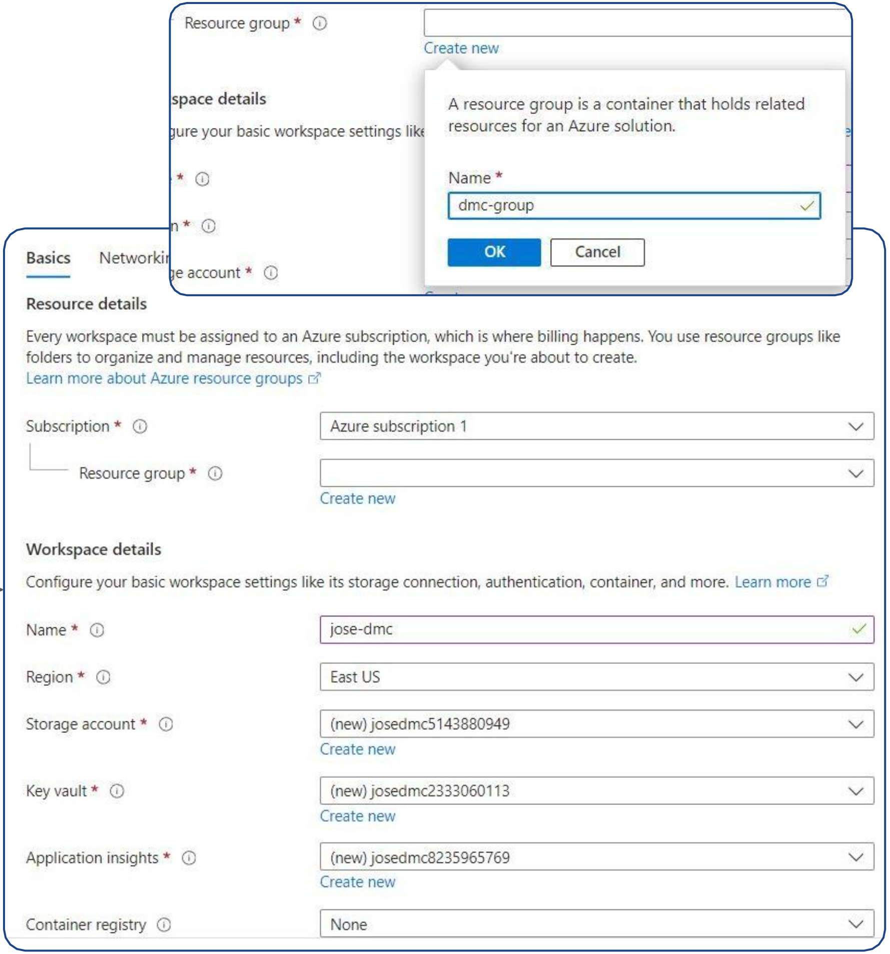
En Azure, un "Resource Group" o Grupo de Recursos es un **contenedor que alberga recursos relacionados para una solución de Azure.** Los recursos son elementos de Azure como aplicaciones web, bases de datos, cuentas de almacenamiento, redes virtuales, etc. Un Grupo de Recursos es una forma de organizar estos recursos en la arquitectura de Azure, facilitando su gestión, monitoreo y acceso al control.



Azure ML: Crear workspace







Azure ML: Crear workspace

Basics | **Networking** | Encryption | Identity | Tags | Review + create

Network isolation

Choose the type of network isolation you need for your workspace, from not isolated at all to an entirely separate virtual network managed by Azure Machine Learning. Learn more about managed network isolation

Public
- Workspace is accessed via public endpoint
- Compute can access public resources
- Outbound data movement is unrestricted

Private with Internet Outbound
- Workspace is accessed via private endpoint
- Compute can access private resources
- Outbound data movement is unrestricted

Private with Approved Outbound
- Workspace is accessed via private endpoint
- Compute can access allowlisted resources only
- Outbound data movement is restricted to approved targets

Basics | Networking | **Encryption** | Identity | Tags | Review + create

Data encryption

Azure Machine Learning service stores metrics and metadata in an Azure Cosmos DB instance where all data is encrypted at rest. By default, the data is encrypted with Microsoft-managed keys. You may choose to bring your own (customer-managed) keys.

Encryption type
- Microsoft-managed keys
- Customer-managed keys

When using a customer-managed key, the costs for your subscription will be higher because of the additional resources in your subscription. To estimate the cost, use the Azure pricing calculator. To learn more, see Use customer-managed keys - Azure Machine Learning | Microsoft Docs

Basics | Networking | Encryption | **Identity** | Tags | Review + create

Managed identity

A managed identity enables Azure resources to authenticate to cloud services without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based access control. A workspace can be given either a system assigned identity or a user assigned identity.

Identity type
- System assigned identity
- User assigned identity

The managed user assigned identity option is only supported if an existing storage account, key vault, and container registry are used.

Storage account access

Azure machine learning allows you to choose between credential-based or identity-based access when connecting to the default storage account. When using identity-based authentication, the Storage Blob Data Contributor role must be granted to the workspace managed identity on the storage account. Learn more

Storage account access type
- Credential-based access
- Identity-based access

Data impact

If your workspace contains sensitive data, you can specify a high business impact workspace. This will control the amount of data Microsoft collects for diagnostic purposes and enables additional encryption in Microsoft managed environments.

High business impact workspace

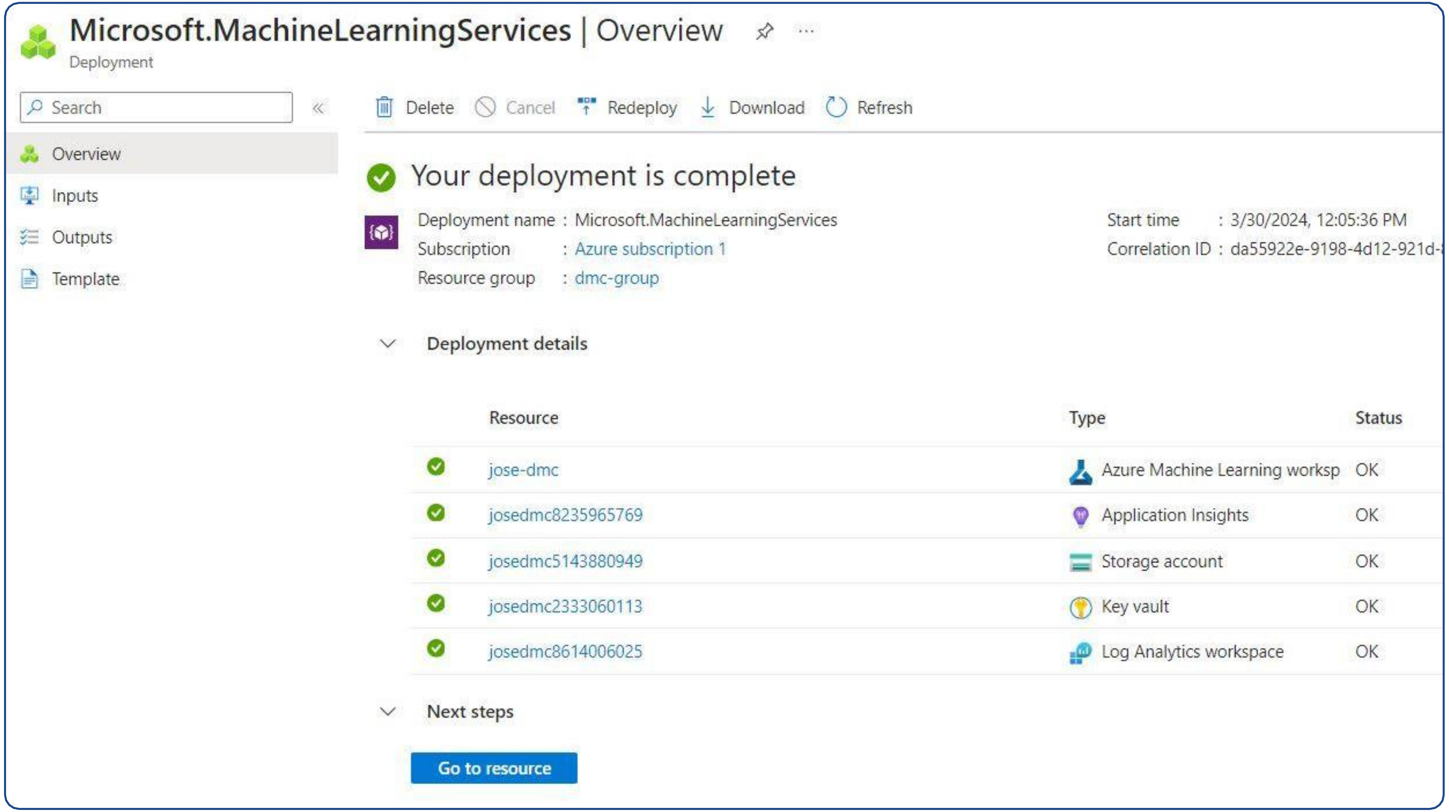
Basics | Networking | Encryption | Identity | **Tags** | Review + create

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. Learn more about tags

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

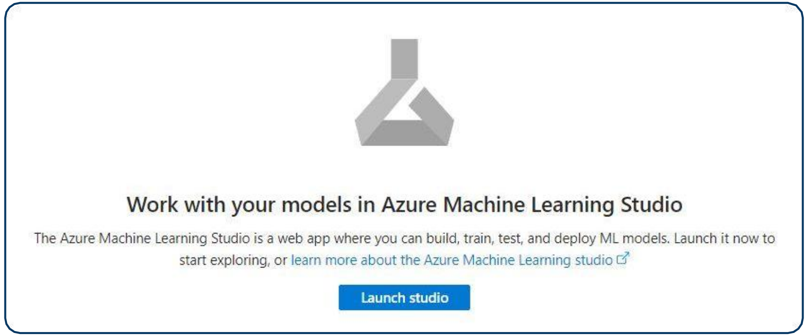
Name		Value
	:	

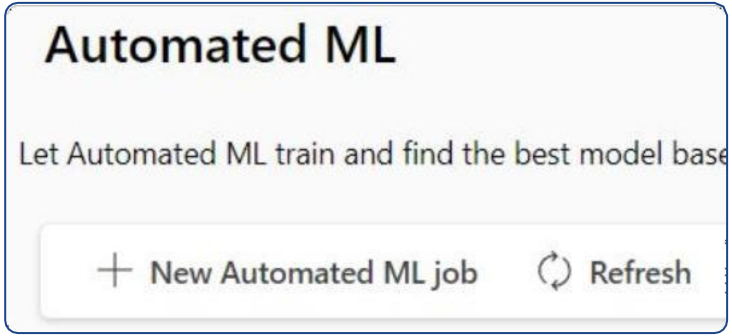
Azure ML: Crear workspace

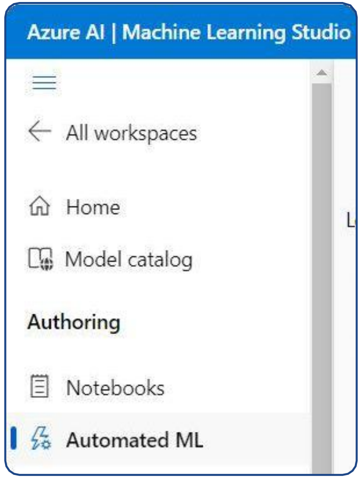


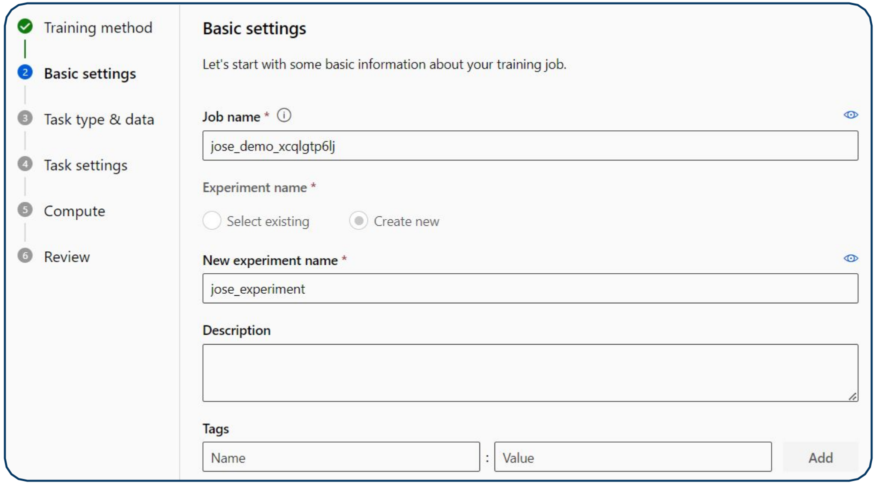
Elaborado por: José Enrique Mejía Gamarra

Azure ML: Launch Studio

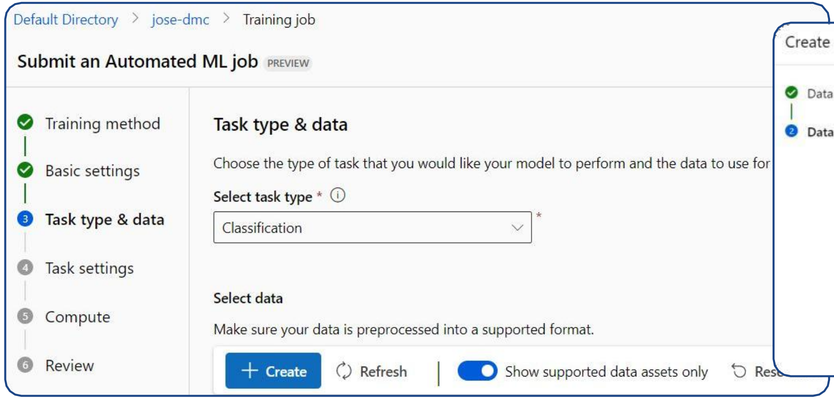


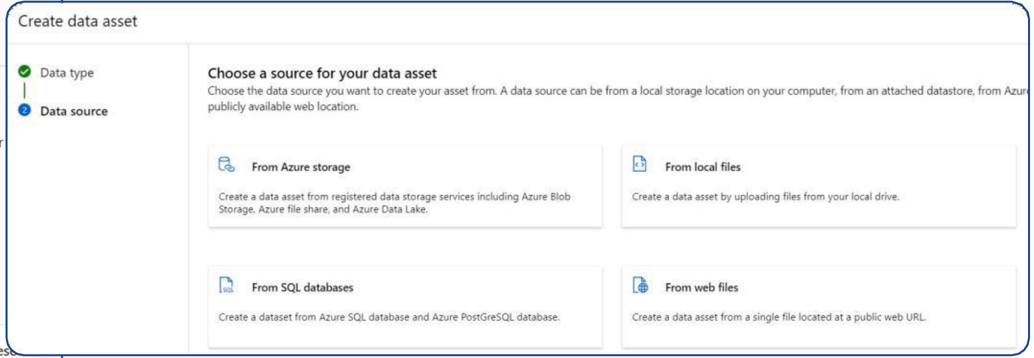


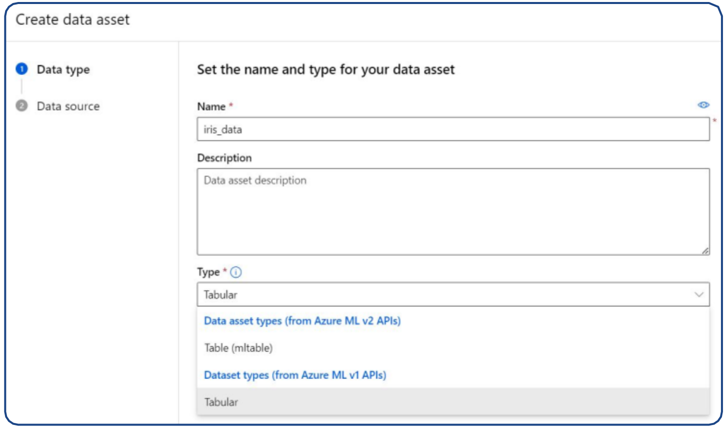


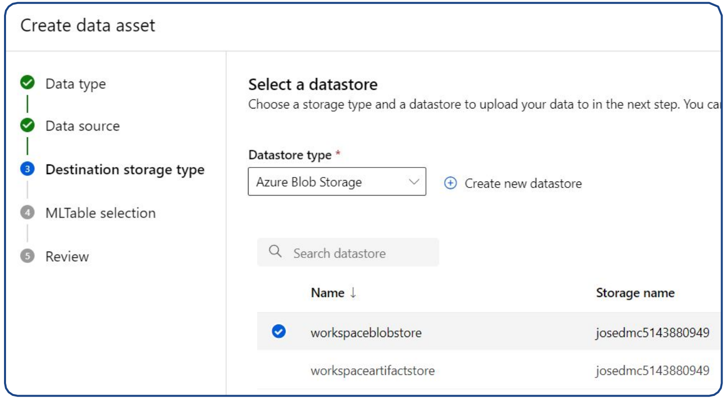


Azure ML: Launch Studio



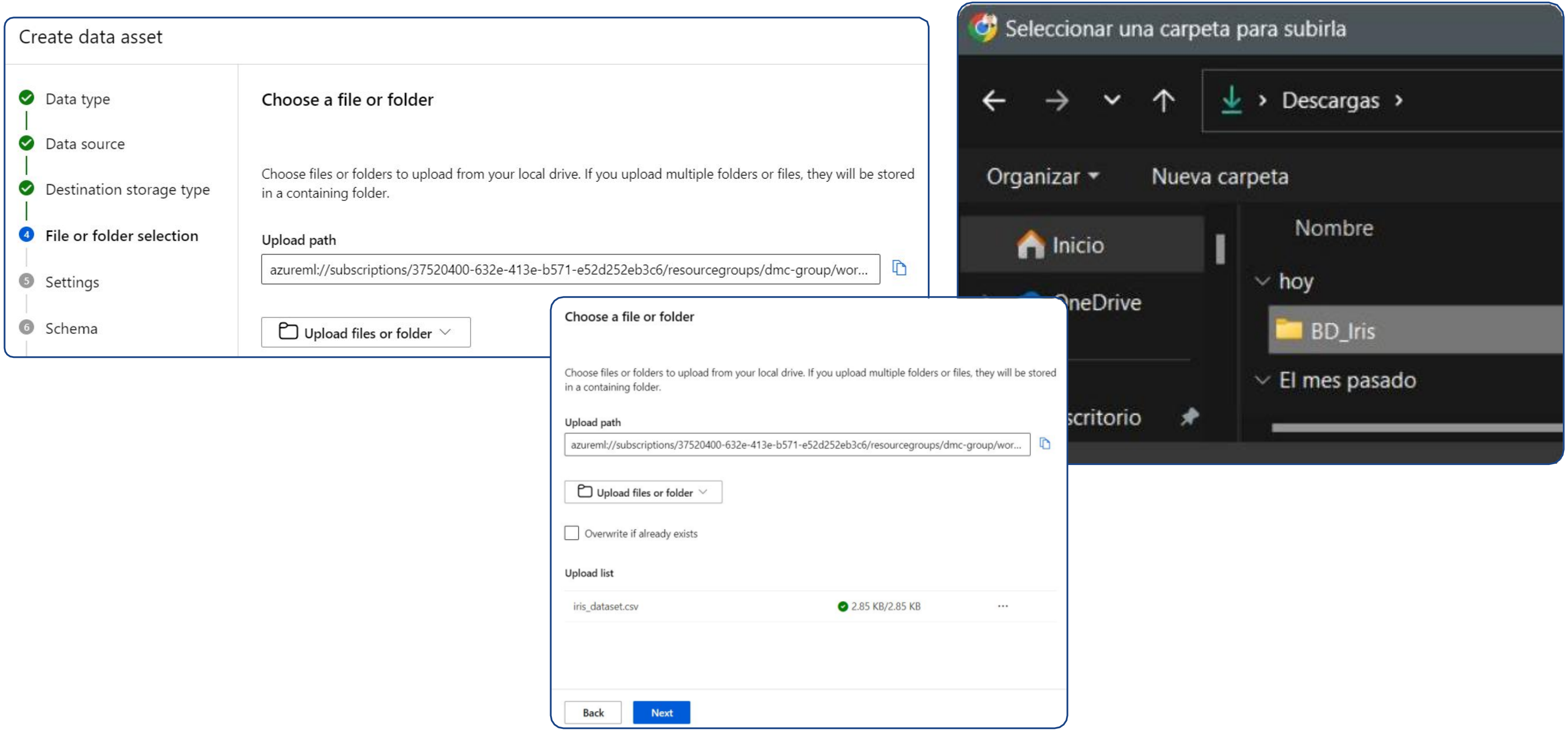






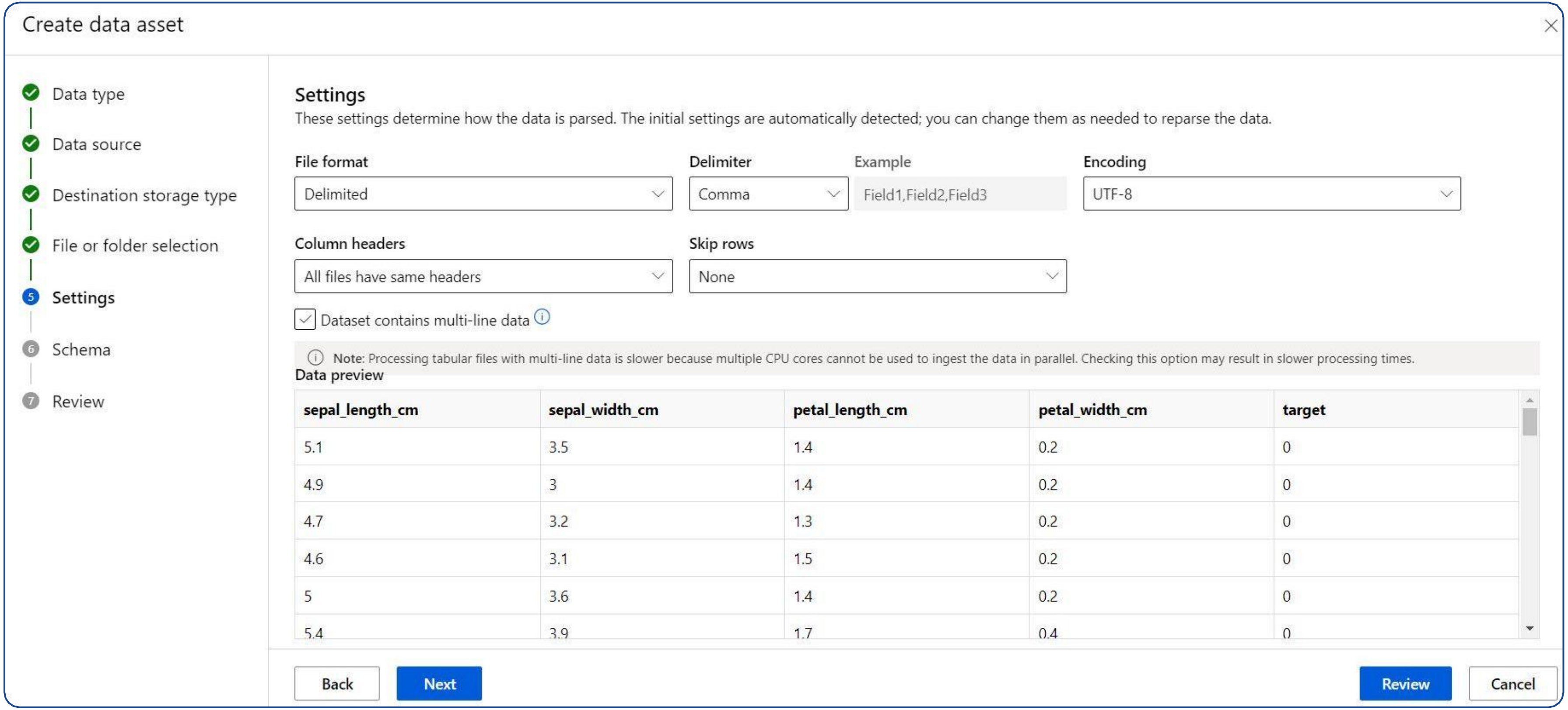
Elaborado por: José Enrique Mejía Gamarra

Azure ML: Launch Studio

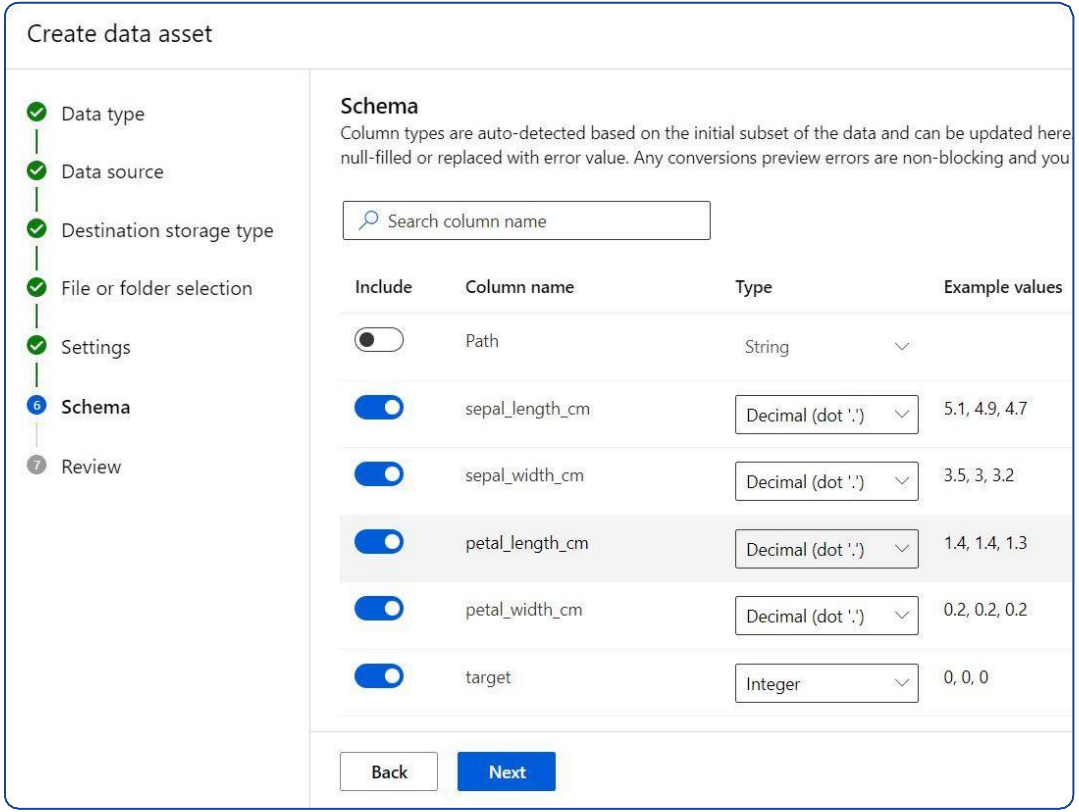


Elaborado por: José Enrique Mejía Gamarra

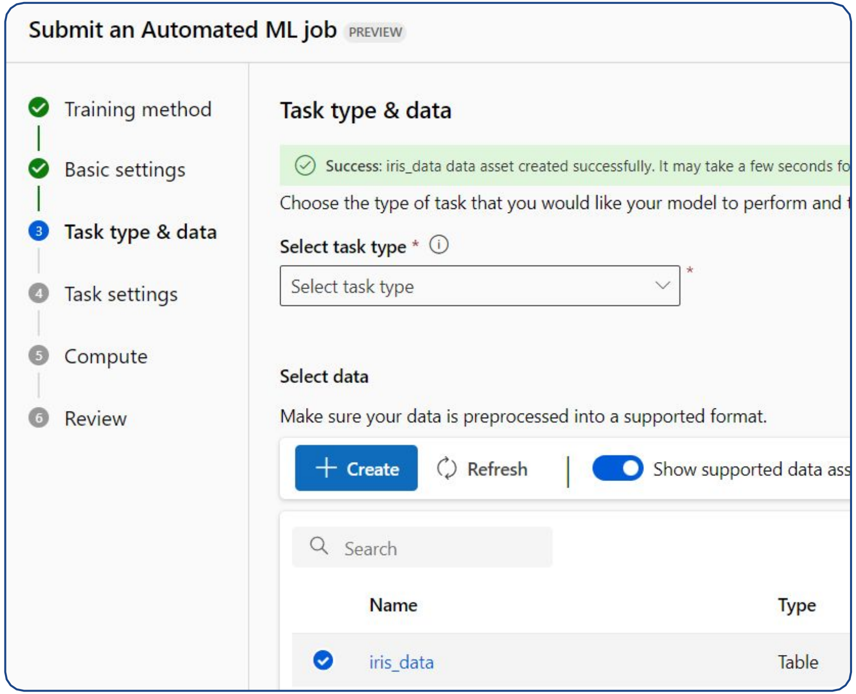
Azure ML: Launch Studio

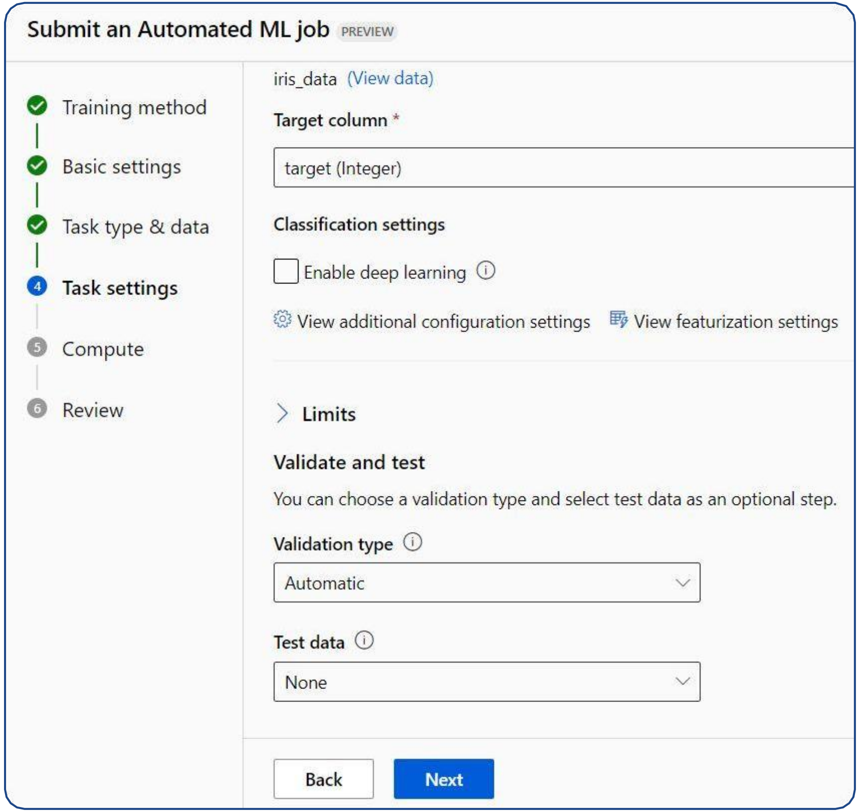


Azure ML: Launch Studio

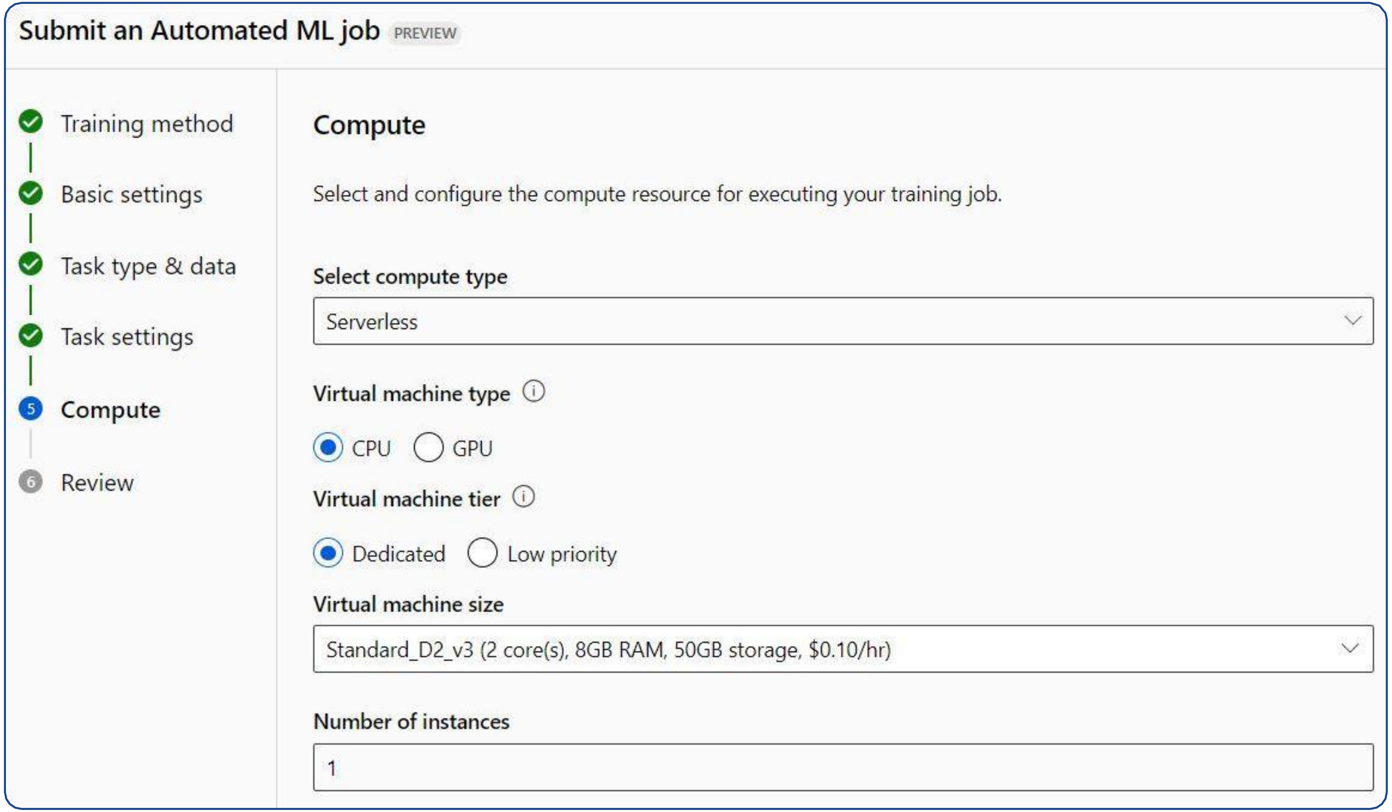


Azure ML: Launch Studio

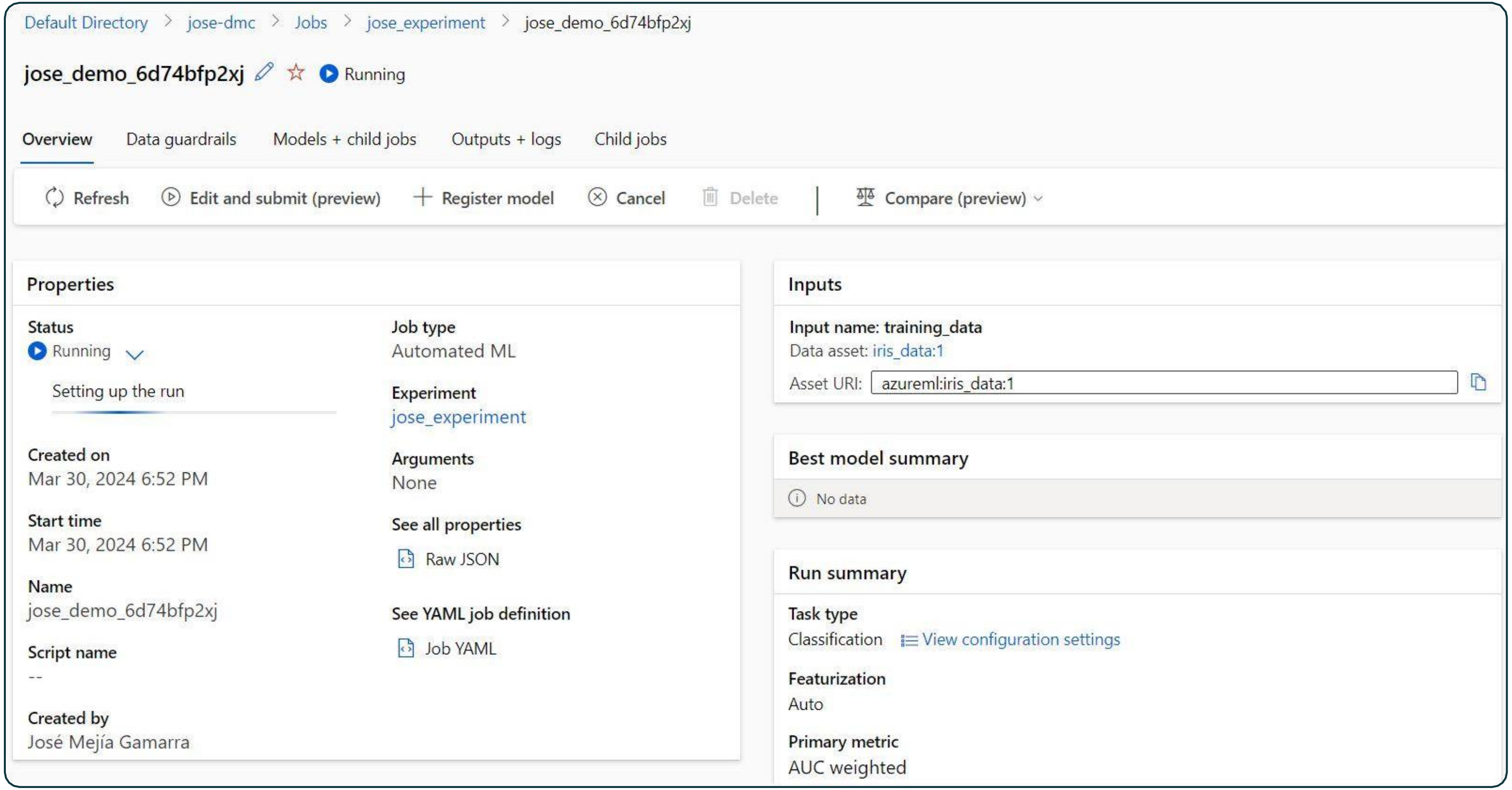




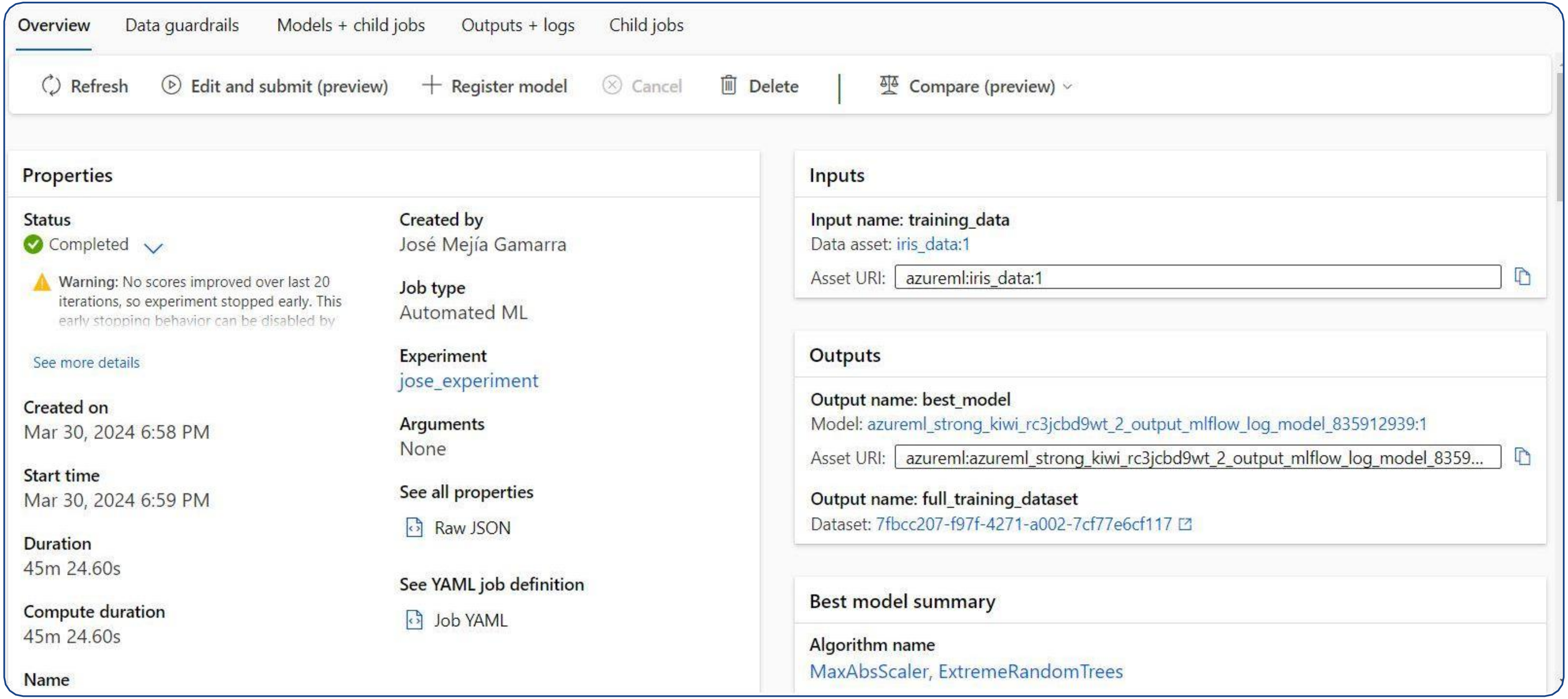
Azure ML: Launch Studio



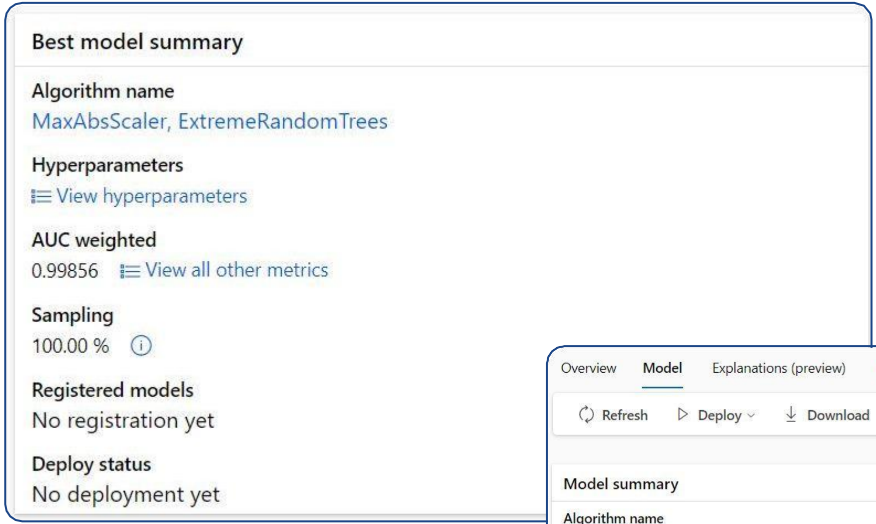
Azure ML: Launch Studio

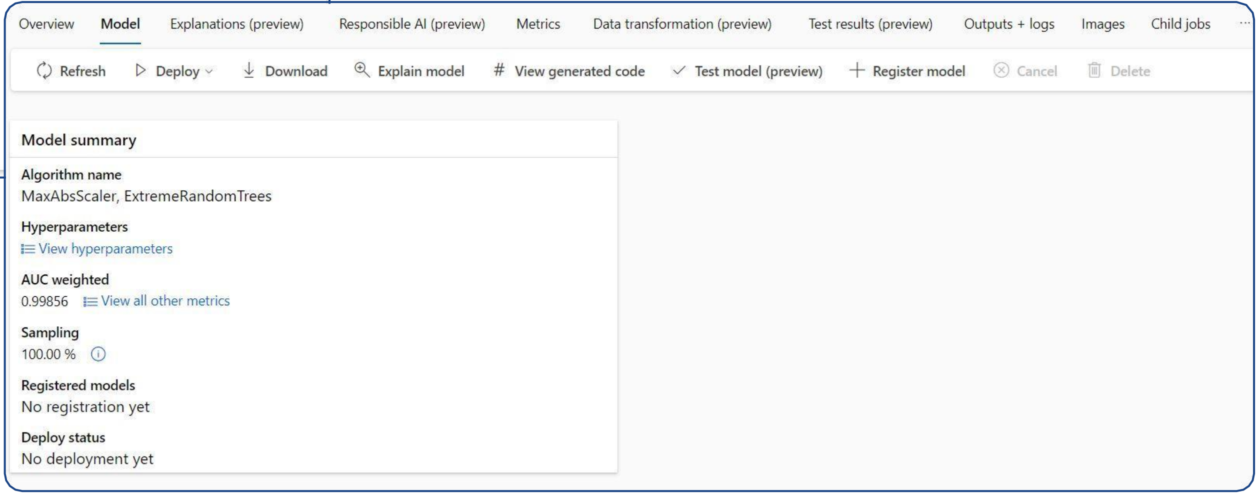


Azure ML: Launch Studio



Azure ML: Launch Studio

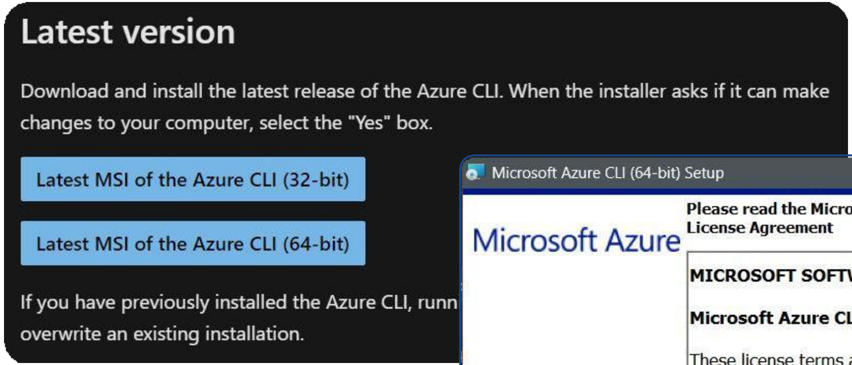


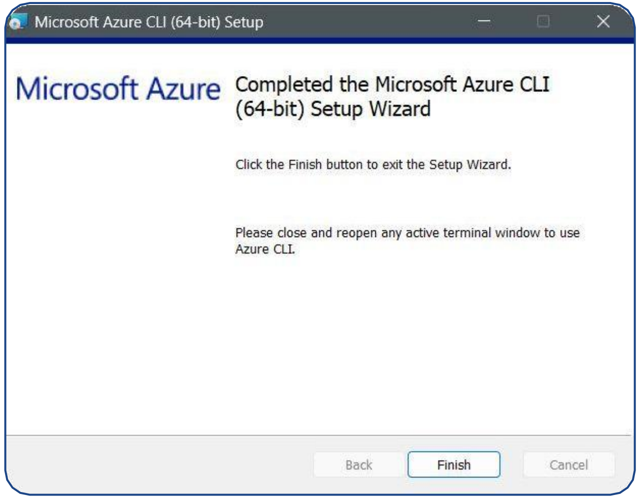


Elaborado por: José Enrique Mejía Gamarra

Azure: Instalar CLI

https://learn.microsoft.com/en-us/cli/azure/install-azure-cli





Elaborado por: José Enrique Mejía Gamarra

Azure: AutoML with Python SDK

Step 0: Install dependencies

```
# !pip install azure-identity==1.15.0
# !pip install azureml-fsspec==1.3.1
# !pip install azureml-sdk==1.55.0
# !pip install azure-ai-ml==1.15.0
# !pip install mltable==1.6.1
```

Step 1: Set up your workspace

```python
from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient

credential = DefaultAzureCredential()
ml_client = None
try:
    # Enter details of your Azure Machine Learning workspace
    subscription_id = "suscription-id-413e-b571-e52d252eb3c6"
    resource_group = "dmc-group"
    workspace = "jose-dmc"
    ml_client = MLClient(credential, subscription_id, resource_group, workspace)
except Exception as ex:
    print(ex)
```

Step 2: Data source and format

```python
import mltable

paths = [
    {'file': './train_data/iris_dataset.csv'}
]

train_table = mltable.from_delimited_files(paths)
train_table.save('./train_data')
```

✓ 1.9s Python

```
paths:
- file: file://d:\DMC\AutoML_Azure\train_data\iris_dataset.csv
transformations:
- read_delimited:
    delimiter: ','
    empty_as_string: false
    encoding: utf8
    header: all_files_same_headers
    include_path_column: false
    infer_column_types: true
    partition_size: 20971520
    path_column: Path
    support_multi_line: false
type: mltable
```

Elaborado por: José Enrique Mejía Gamarra

Azure: AutoML with Python SDK

Step 3: Create a Compute Instance (If not exist)

```python
# #Sign into Azure with Azure CLI
# !az login
```

✓ 0.0s Python

```python
ci_basic_name = "jose-basic-ci"
from azure.ai.ml.entities import ComputeInstance, AmlCompute
ci_basic = ComputeInstance(name=ci_basic_name, size="Standard_DS2_v2")
ml_client.begin_create_or_update(ci_basic).result()
```

✓ 1m 37.5s Python

```
ComputeInstance({'state': 'Running', 'last_operation': {'operation_name': 'Create', '
```

Azure: AutoML with Python SDK

Step 4: Configure your experiment settings

```
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import automl, Input

# make an Input object for the training data
my_training_data_input = Input(
    type=AssetTypes.MLTABLE, path="./train_data"
)

# configure the classification job
classification_job = automl.classification(
    compute                 = ci_basic_name,
    experiment_name         = "jose_experiment_python_3",
    training_data           = my_training_data_input,
    target_column_name      = "target",
    primary_metric          = "accuracy",
    n_cross_validations = 5,
    enable_model_explainability = True,
    tags                    = {"dmc": "demo"}
)

# # Limits are all optional
# classification_job.set_limits(
#     timeout_minutes=600,
#     trial_timeout_minutes=20,
#     max_trials=2,
#     enable_early_termination=True,
# )

# # Training properties are optional
# classification_job.set_training(
#     blocked_training_algorithms=["logistic_regression"],
#     enable_onnx_compatible_models=True
# )
```

Azure: AutoML with Python SDK

Step 5: Run experiment

```
# Submit the AutoML job
returned_job = ml_client.jobs.create_or_update(classification_job)

# Get the URL to monitor the job in Azure Machine Learning studio
print(f"Monitor your job at {returned_job.studio_url}")
```

✓ 10.2s

```
Monitor your job at https://ml.azure.com/runs/quirky_star_nktb75xkkp?wsi
```

Azure: AutoML with Python SDK