

HOW-TO GUIDE

[Run training with CLI, SDK, or REST API](#)

[Tune hyperparameters for model training](#)

[Build pipelines from reusable components](#)

[Use automated ML in studio](#)

[Use designer \(drag-n-drop\) in studio](#)

[Train with R](#)

Deploy models

DEPLOY

[Streamline model deployment with endpoints](#)

[Real-time scoring with online endpoints](#)

[Batch scoring with batch endpoints](#)

[Deploy R models](#)

Manage the ML lifecycle (MLOps)

HOW-TO GUIDE

[Track, monitor, analyze training runs](#)

[Model management, deployment & monitoring](#)

Security for ML projects

HOW-TO GUIDE

[Create a secure workspace](#)

[Connect to data sources](#)

[Enterprise security & governance](#)

Reference docs



[Python SDK \(v2\)](#)

[CLI \(v2\)](#)

[REST API](#)

[Algorithm & component reference](#)

Resources



[Upgrade to v2](#)

[Python SDK \(v2\) code examples ↗](#)

[CLI \(v2\) code examples ↗](#)

[ML Studio \(classic\) documentation](#)

Azure Machine Learning SDK & CLI (v1)

Article • 02/24/2023 • 2 minutes to read

APPLIES TO:  Azure CLI ml extension v1  Python SDK azureml v1

All articles in this section document the use of the first version of Azure Machine Learning Python SDK (v1) or Azure CLI ml extension (v1).

SDK v1

The Azure SDK examples in articles in this section require the `azureml-core`, or Python SDK v1 for Azure Machine Learning. The Python SDK v2 is now available.

The v1 and v2 Python SDK packages are incompatible, and v2 style of coding will not work for articles in this directory. However, machine learning workspaces and all underlying resources can be interacted with from either, meaning one user can create a workspace with the SDK v1 and another can submit jobs to the same workspace with the SDK v2.

We recommend not to install both versions of the SDK on the same environment, since it can cause clashes and confusion in the code.

How do I know which SDK version I have?

- To find out whether you have Azure Machine Learning Python SDK v1, run `pip show azureml-core`. (Or, in a Jupyter notebook, use `%pip show azureml-core`)
- To find out whether you have Azure Machine Learning Python SDK v2, run `pip show azure-ai-ml`. (Or, in a Jupyter notebook, use `%pip show azure-ai-ml`)

Based on the results of `pip show` you can determine which version of SDK you have.

CLI v1

The Azure CLI commands in articles in this section require the `azure-cli-ml`, or v1, extension for Azure Machine Learning. The enhanced v2 CLI using the `ml` extension is now available and recommended.

The extensions are incompatible, so v2 CLI commands will not work for articles in this directory. However, machine learning workspaces and all underlying resources can be

interacted with from either, meaning one user can create a workspace with the v1 CLI and another can submit jobs to the same workspace with the v2 CLI.

How do I know which CLI extension I have?

To find which extensions you have installed, use `az extension list`.

- If the list of **Extensions** contains `azure-cli-ml`, you have the v1 extension.
- If the list contains `ml`, you have the v2 extension.

Next steps

For more information on installing and using the different extensions, see the following articles:

- `azure-cli-ml` - [Install, set up, and use the CLI \(v1\)](#)
- `ml` - [Install and set up the CLI \(v2\)](#)

For more information on installing and using the different SDK versions:

- `azureml-core` - [Install the Azure Machine Learning SDK \(v1\) for Python](#)
- `azure-ai-ml` - [Install the Azure Machine Learning SDK \(v2\) for Python ↗](#)

Additional resources

Documentation

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[Hyperparameter tuning a model \(v1\) - Azure Machine Learning](#)

Automate hyperparameter tuning for deep learning and machine learning models using Azure Machine Learning.(v1)

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Customize outputs in batch deployments - Azure Machine Learning](#)

Learn how create deployments that generate custom outputs and files.

[Troubleshooting local model deployment - Azure Machine Learning](#)

Try a local model deployment as a first step in troubleshooting model deployment errors.

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[Author scoring scripts for batch deployments - Azure Machine Learning](#)

In this article, learn how to author scoring scripts to perform batch inference in batch deployments.

[Show 5 more](#)

[Training](#)

Learning paths and modules

[Train models in Azure Machine Learning with the CLI \(v2\) - Training](#)

Train models in Azure Machine Learning with the CLI (v2)

What is Azure Machine Learning?

Article • 02/24/2023 • 7 minutes to read

Azure Machine Learning is a cloud service for accelerating and managing the machine learning project lifecycle. Machine learning professionals, data scientists, and engineers can use it in their day-to-day workflows: Train and deploy models, and manage MLOps.

You can create a model in Azure Machine Learning or use a model built from an open-source platform, such as Pytorch, TensorFlow, or scikit-learn. MLOps tools help you monitor, retrain, and redeploy models.

💡 Tip

Free trial! If you don't have an Azure subscription, create a free account before you begin. [Try the free or paid version of Azure Machine Learning ↗](#). You get credits to spend on Azure services. After they're used up, you can keep the account and use [free Azure services ↗](#). Your credit card is never charged unless you explicitly change your settings and ask to be charged.

Who is Azure Machine Learning for?

Azure Machine Learning is for individuals and teams implementing MLOps within their organization to bring machine learning models into production in a secure and auditable production environment.

Data scientists and ML engineers will find tools to accelerate and automate their day-to-day workflows. Application developers will find tools for integrating models into applications or services. Platform developers will find a robust set of tools, backed by durable Azure Resource Manager APIs, for building advanced ML tooling.

Enterprises working in the Microsoft Azure cloud will find familiar security and role-based access control (RBAC) for infrastructure. You can set up a project to deny access to protected data and select operations.

Productivity for everyone on the team

Machine learning projects often require a team with varied skill set to build and maintain. Azure Machine Learning has tools that help enable you to:

- Collaborate with your team via shared notebooks, compute resources, data, and environments
- Develop models for fairness and explainability, tracking and auditability to fulfill lineage and audit compliance requirements
- Deploy ML models quickly and easily at scale, and manage and govern them efficiently with MLOps
- Run machine learning workloads anywhere with built-in governance, security, and compliance

Cross-compatible platform tools that meet your needs

Anyone on an ML team can use their preferred tools to get the job done. Whether you're running rapid experiments, hyperparameter-tuning, building pipelines, or managing inferences, you can use familiar interfaces including:

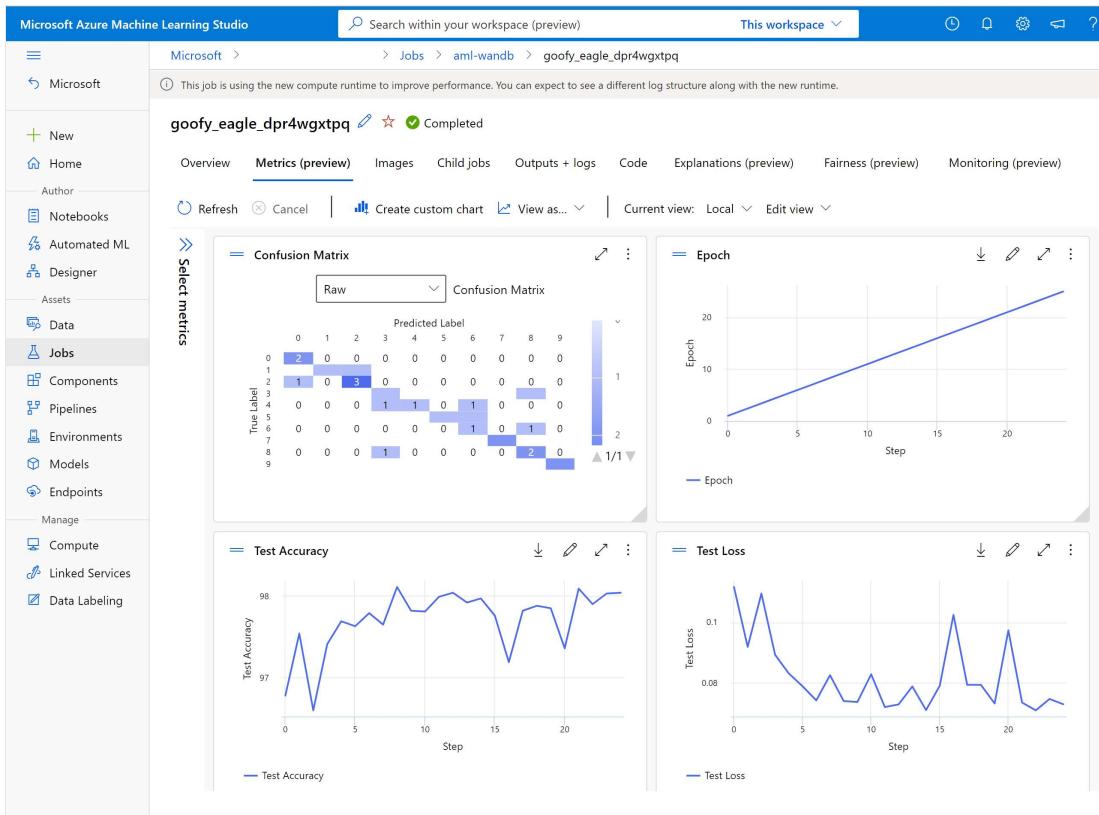
- [Azure Machine Learning studio](#)
- [Python SDK \(v2\)](#)
- [CLI \(v2\)](#)
- [Azure Resource Manager REST APIs](#)

As you're refining the model and collaborating with others throughout the rest of Machine Learning development cycle, you can share and find assets, resources, and metrics for your projects on the Azure Machine Learning studio UI.

Studio

The [Azure Machine Learning studio](#) offers multiple authoring experiences depending on the type of project and the level of your past ML experience, without having to install anything.

- Notebooks: write and run your own code in managed Jupyter Notebook servers that are directly integrated in the studio.
- Visualize run metrics: analyze and optimize your experiments with visualization.



- Azure Machine Learning designer: use the designer to train and deploy machine learning models without writing any code. Drag and drop datasets and components to create ML pipelines. Try out the [designer tutorial](#).
- Automated machine learning UI: Learn how to create [automated ML experiments](#) with an easy-to-use interface.
- Data labeling: Use Azure Machine Learning data labeling to efficiently coordinate [image labeling](#) or [text labeling](#) projects.

Enterprise-readiness and security

Azure Machine Learning integrates with the Azure cloud platform to add security to ML projects.

Security integrations include:

- Azure Virtual Networks (VNets) with network security groups
- Azure Key Vault where you can save security secrets, such as access information for storage accounts
- Azure Container Registry set up behind a VNet

See [Tutorial: Set up a secure workspace](#).

Azure integrations for complete solutions

Other integrations with Azure services support a machine learning project from end-to-end. They include:

- Azure Synapse Analytics to process and stream data with Spark
- Azure Arc, where you can run Azure services in a Kubernetes environment
- Storage and database options, such as Azure SQL Database, Azure Storage Blobs, and so on
- Azure App Service allowing you to deploy and manage ML-powered apps

ⓘ Important

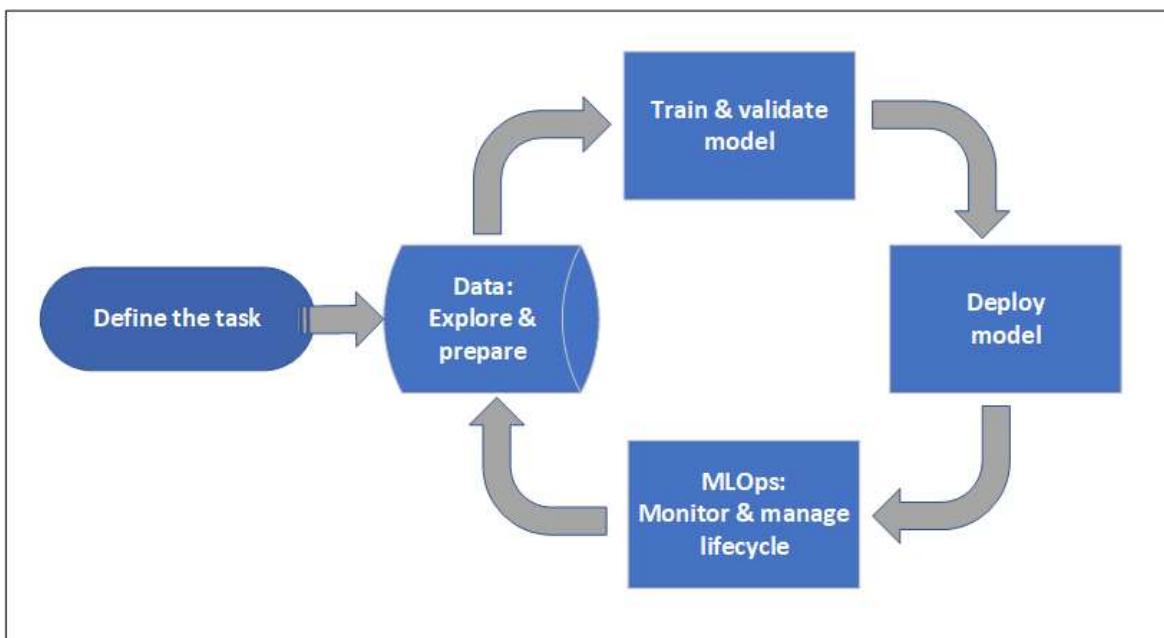
Azure Machine Learning doesn't store or process your data outside of the region where you deploy.

Machine learning project workflow

Typically models are developed as part of a project with an objective and goals. Projects often involve more than one person. When experimenting with data, algorithms, and models, development is iterative.

Project lifecycle

While the project lifecycle can vary by project, it will often look like this:



A workspace organizes a project and allows for collaboration for many users all working toward a common objective. Users in a workspace can easily share the results of their runs from experimentation in the studio user interface or use versioned assets for jobs like environments and storage references.

For more information, see [Manage Azure Machine Learning workspaces](#).

When a project is ready for operationalization, users' work can be automated in a machine learning pipeline and triggered on a schedule or HTTPS request.

Models can be deployed to the managed inferencing solution, for both real-time and batch deployments, abstracting away the infrastructure management typically required for deploying models.

Train models

In Azure Machine Learning, you can run your training script in the cloud or build a model from scratch. Customers often bring models they've built and trained in open-source frameworks, so they can operationalize them in the cloud.

Open and interoperable

Data scientists can use models in Azure Machine Learning that they've created in common Python frameworks, such as:

- PyTorch
- TensorFlow
- scikit-learn
- XGBoost
- LightGBM

Other languages and frameworks are supported as well, including:

- R
- .NET

See [Open-source integration with Azure Machine Learning](#).

Automated featurization and algorithm selection (AutoML)

In a repetitive, time-consuming process, in classical machine learning data scientists use prior experience and intuition to select the right data featurization and algorithm for training. Automated ML (AutoML) speeds this process and can be used through the studio UI or Python SDK.

See [What is automated machine learning?](#)

Hyperparameter optimization

Hyperparameter optimization, or hyperparameter tuning, can be a tedious task. Azure Machine Learning can automate this task for arbitrary parameterized commands with little modification to your job definition. Results are visualized in the studio.

See [How to tune hyperparameters](#).

Multinode distributed training

Efficiency of training for deep learning and sometimes classical machine learning training jobs can be drastically improved via multinode distributed training. Azure Machine Learning compute clusters offer the latest GPU options.

Supported via Azure Machine Learning Kubernetes and Azure Machine Learning compute clusters:

- PyTorch
- TensorFlow
- MPI

The MPI distribution can be used for Horovod or custom multinode logic. Additionally, Apache Spark is supported via Azure Synapse Analytics Spark clusters (preview).

See [Distributed training with Azure Machine Learning](#).

Embarrassingly parallel training

Scaling a machine learning project may require scaling embarrassingly parallel model training. This pattern is common for scenarios like forecasting demand, where a model may be trained for many stores.

Deploy models

To bring a model into production, it's deployed. Azure Machine Learning's managed endpoints abstract the required infrastructure for both batch or real-time (online) model scoring (inferencing).

Real-time and batch scoring (inferencing)

Batch scoring, or *batch inferencing*, involves invoking an endpoint with a reference to data. The batch endpoint runs jobs asynchronously to process data in parallel on compute clusters and store the data for further analysis.

Real-time scoring, or *online inferencing*, involves invoking an endpoint with one or more model deployments and receiving a response in near-real-time via HTTPs. Traffic can be split across multiple deployments, allowing for testing new model versions by diverting some amount of traffic initially and increasing once confidence in the new model is established.

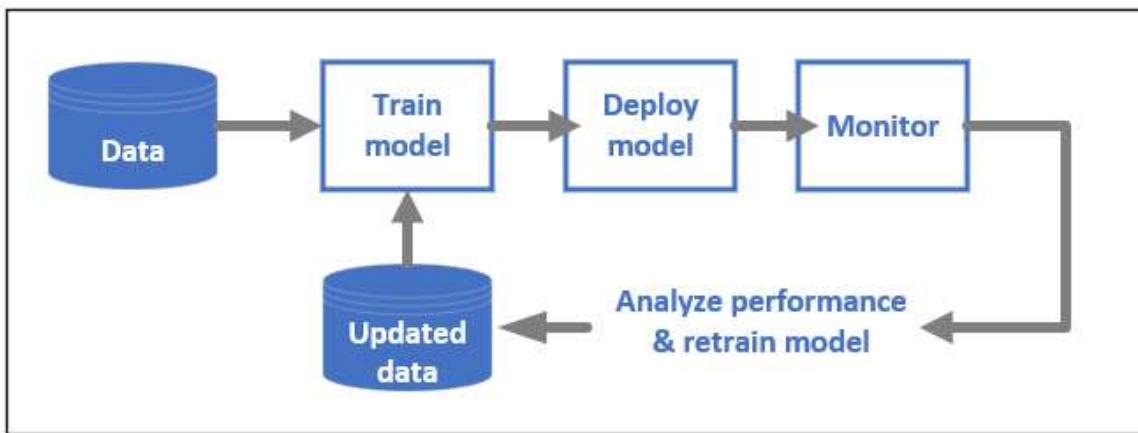
See:

- [Deploy a model with a real-time managed endpoint](#)
- [Use batch endpoints for scoring](#)

MLOps: DevOps for machine learning

DevOps for machine learning models, often called MLOps, is a process for developing models for production. A model's lifecycle from training to deployment must be auditable if not reproducible.

ML model lifecycle



Learn more about [MLOps in Azure Machine Learning](#).

Integrations enabling MLOps

Azure Machine Learning is built with the model lifecycle in mind. You can audit the model lifecycle down to a specific commit and environment.

Some key features enabling MLOps include:

- `git` integration
- MLflow integration
- Machine learning pipeline scheduling
- Azure Event Grid integration for custom triggers
- Easy to use with CI/CD tools like GitHub Actions or Azure DevOps

Also, Azure Machine Learning includes features for monitoring and auditing:

- Job artifacts, such as code snapshots, logs, and other outputs
- Lineage between jobs and assets, such as containers, data, and compute resources

Next steps

Start using Azure Machine Learning:

- [Set up an Azure Machine Learning workspace](#)
 - [Tutorial: Build a first machine learning project](#)
 - [How to run training jobs](#)
-

Additional resources

Documentation

[Azure Machine Learning examples - Code Samples](#)

Top-level directory for official Azure Machine Learning sample code and examples.

[Quickstart: Create workspace resources - Azure Machine Learning](#)

Create an Azure Machine Learning workspace and cloud resources that can be used to train machine learning models.

[ML Studio \(classic\): Migrate to Azure Machine Learning - Azure Machine Learning](#)

Migrate from Studio (classic) to Azure Machine Learning for a modernized data science platform.

[Plan to manage costs - Azure Machine Learning](#)

Plan and manage costs for Azure Machine Learning with cost analysis in Azure portal. Learn further cost-saving tips to lower your cost when building ML models.

[MLOps with Azure Machine Learning - Cloud Adoption Framework](#)

Learn about MLOps principles and practices that increase workflow efficiencies like continuous integration, delivery, and deployment.

[MLOps: Machine learning model management - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage, and monitor your models to continuously improve them.

[Tutorial: AutoML- train no-code classification models - Azure Machine Learning](#)

Train a classification model without writing a single line of code using Azure Machine Learning automated ML in the studio UI.

[Tutorial: Azure ML in a day - Azure Machine Learning](#)

Use Azure Machine Learning to train and deploy a model in a cloud-based Python Jupyter Notebook.

[Show 5 more](#)

[Training](#)

Learning paths and modules

[Build and operate machine learning solutions with Azure Machine Learning - Training](#)

Build and operate machine learning solutions with Azure Machine Learning

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Azure Machine Learning glossary

Article • 02/24/2023 • 3 minutes to read

The Azure Machine Learning glossary is a short dictionary of terminology for the Azure Machine Learning platform. For the general Azure terminology, see also:

- [Microsoft Azure glossary: A dictionary of cloud terminology on the Azure platform](#)
- [Cloud computing terms ↗](#) - General industry cloud terms.
- [Azure fundamental concepts](#) - Microsoft Cloud Adoption Framework for Azure.

Component

An Azure Machine Learning [component](#) is a self-contained piece of code that does one step in a machine learning pipeline. Components are the building blocks of advanced machine learning pipelines. Components can do tasks such as data processing, model training, model scoring, and so on. A component is analogous to a function - it has a name, parameters, expects input, and returns output.

Compute

A compute is a designated compute resource where you run your job or host your endpoint. Azure Machine learning supports the following types of compute:

- **Compute cluster** - a managed-compute infrastructure that allows you to easily create a cluster of CPU or GPU compute nodes in the cloud.
- **Compute instance** - a fully configured and managed development environment in the cloud. You can use the instance as a training or inference compute for development and testing. It's similar to a virtual machine on the cloud.
- **Kubernetes cluster** - used to deploy trained machine learning models to Azure Kubernetes Service. You can create an Azure Kubernetes Service (AKS) cluster from your Azure Machine Learning workspace, or attach an existing AKS cluster.
- **Attached compute** - You can attach your own compute resources to your workspace and use them for training and inference.

Data

Azure Machine Learning allows you to work with different types of data:

- URIs (a location in local/cloud storage)
 - `uri_folder`

- `uri_file`
- Tables (a tabular data abstraction)
 - `mltable`
- Primitives
 - `string`
 - `boolean`
 - `number`

For most scenarios, you'll use URIs (`uri_folder` and `uri_file`) - a location in storage that can be easily mapped to the filesystem of a compute node in a job by either mounting or downloading the storage to the node.

`mltable` is an abstraction for tabular data that is to be used for AutoML Jobs, Parallel Jobs, and some advanced scenarios. If you're just starting to use Azure Machine Learning and aren't using AutoML, we strongly encourage you to begin with URIs.

Datastore

Azure Machine Learning datastores securely keep the connection information to your data storage on Azure, so you don't have to code it in your scripts. You can register and create a datastore to easily connect to your storage account, and access the data in your underlying storage service. The CLI v2 and SDK v2 support the following types of cloud-based storage services:

- Azure Blob Container
- Azure File Share
- Azure Data Lake
- Azure Data Lake Gen2

Environment

Azure Machine Learning environments are an encapsulation of the environment where your machine learning task happens. They specify the software packages, environment variables, and software settings around your training and scoring scripts. The environments are managed and versioned entities within your Machine Learning workspace. Environments enable reproducible, auditable, and portable machine learning workflows across various computes.

Types of environment

Azure Machine Learning supports two types of environments: curated and custom.

Curated environments are provided by Azure Machine Learning and are available in your workspace by default. Intended to be used as is, they contain collections of Python packages and settings to help you get started with various machine learning frameworks. These pre-created environments also allow for faster deployment time. For a full list, see the [curated environments article](#).

In custom environments, you're responsible for setting up your environment. Make sure to install the packages and any other dependencies that your training or scoring script needs on the compute. Azure Machine Learning allows you to create your own environment using

- A docker image
- A base docker image with a conda YAML to customize further
- A docker build context

Model

Azure machine learning models consist of the binary file(s) that represent a machine learning model and any corresponding metadata. Models can be created from a local or remote file or directory. For remote locations `https`, `wasbs` and `azureml` locations are supported. The created model will be tracked in the workspace under the specified name and version. Azure Machine Learning supports three types of storage format for models:

- `custom_model`
- `mlflow_model`
- `triton_model`

Workspace

The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. The workspace keeps a history of all jobs, including logs, metrics, output, and a snapshot of your scripts. The workspace stores references to resources like datastores and compute. It also holds all assets like models, environments, components and data asset.

Next steps

Additional resources

Documentation

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Share Responsible AI insights and make data-driven decisions with Azure Machine Learning Responsible AI scorecard - Azure Machine Learning](#)

Learn about how to use the Responsible AI scorecard to share responsible AI insights from your machine learning models and make data-driven decisions with non-technical and technical stakeholders.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Set up AutoML with Python - Azure Machine Learning](#)

Learn how to set up an AutoML training run with the Azure Machine Learning Python SDK using Azure Machine Learning automated ML.

[Export Data: Component Reference - Azure Machine Learning](#)

Use the Export Data component in Azure Machine Learning designer to save results and intermediate data outside of Azure Machine Learning.

[Overview of forecasting methods in AutoML - Azure Machine Learning](#)

Learn how Azure Machine Learning's AutoML uses machine learning to build forecasting models

[Prepare data for computer vision tasks - Azure Machine Learning](#)

Image data preparation for Azure Machine Learning automated ML to train computer vision models on classification, object detection, and segmentation

[Profile model memory and CPU usage \(v1\) - Azure Machine Learning](#)

Use CLI (v1) or SDK (v1) to profile your model before deployment. Profiling determines the memory and CPU usage of your model.

[Show 5 more](#)

Training

Learning paths and modules

[Build and operate machine learning solutions with Azure Machine Learning - Training](#)

Build and operate machine learning solutions with Azure Machine Learning

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Upgrade to v2

Article • 02/24/2023 • 10 minutes to read

Azure Machine Learning's v2 REST APIs, Azure CLI extension, and Python SDK introduce consistency and a set of new features to accelerate the production machine learning lifecycle. This article provides an overview of upgrading to v2 with recommendations to help you decide on v1, v2, or both.

Prerequisites

- General familiarity with Azure Machine Learning and the v1 Python SDK.
- Understand [what is v2?](#)

Should I use v2?

You should use v2 if you're starting a new machine learning project or workflow. You should use v2 if you want to use the new features offered in v2. The features include:

- Managed Inferencing
- Reusable components in pipelines
- Improved scheduling of pipelines
- Responsible AI dashboard
- Registry of assets

A new v2 project can reuse existing resources like workspaces and compute and existing assets like models and environments created using v1.

Some feature gaps in v2 include:

- Spark support in jobs - this is currently in preview in v2.
- Publishing jobs (pipelines in v1) as endpoints. You can however, schedule pipelines without publishing.
- Support for SQL/database datastores.
- Ability to use classic prebuilt components in the designer with v2.

You should then ensure the features you need in v2 meet your organization's requirements, such as being generally available.

Important

New features in Azure Machine Learning will only be launched in v2.

Should I upgrade existing code to v2

You can reuse your existing assets in your v2 workflows. For instance a model created in v1 can be used to perform Managed Inferencing in v2.

Optionally, if you want to upgrade specific parts of your existing code to v2, please refer to the comparison links provided in the details of each resource or asset in the rest of this document.

Which v2 API should I use?

In v2 interfaces via REST API, CLI, and Python SDK are available. The interface you should use depends on your scenario and preferences.

API	Notes
REST	Fewest dependencies and overhead. Use for building applications on Azure Machine Learning as a platform, directly in programming languages without an SDK provided, or per personal preference.
CLI	Recommended for automation with CI/CD or per personal preference. Allows quick iteration with YAML files and straightforward separation between Azure Machine Learning and ML model code.
Python SDK	Recommended for complicated scripting (for example, programmatically generating large pipeline jobs) or per personal preference. Allows quick iteration with YAML files or development solely in Python.

Can I use v1 and v2 together?

v1 and v2 can co-exist in a workspace. You can reuse your existing assets in your v2 workflows. For instance a model created in v1 can be used to perform Managed Inferencing in v2. Resources like workspace, compute, and datastore work across v1 and v2, with exceptions. A user can call the v1 Python SDK to change a workspace's description, then using the v2 CLI extension change it again. Jobs (experiments/runs/pipelines in v1) can be submitted to the same workspace from the v1 or v2 Python SDK. A workspace can have both v1 and v2 model deployment endpoints.

Using v1 and v2 code together

We do not recommend using the v1 and v2 SDKs together in the same code. It is technically possible to use v1 and v2 in the same code because they use different Azure namespaces. However, there are many classes with the same name across these namespaces (like Workspace, Model) which can cause confusion and make code readability and debuggability challenging.

ⓘ Important

If your workspace uses a private endpoint, it will automatically have the `v1_legacy_mode` flag enabled, preventing usage of v2 APIs. See [how to configure network isolation with v2](#) for details.

Resources and assets in v1 and v2

This section gives an overview of specific resources and assets in Azure Machine Learning. See the concept article for each entity for details on their usage in v2.

Workspace

Workspaces don't need to be upgraded with v2. You can use the same workspace, regardless of whether you're using v1 or v2.

If you create workspaces using automation, do consider upgrading the code for creating a workspace to v2. Typically Azure resources are managed via Azure Resource Manager (and Bicep) or similar resource provisioning tools. Alternatively, you can use the [CLI \(v2\)](#) and [YAML files](#).

For a comparison of SDK v1 and v2 code, see [Workspace management in SDK v1 and SDK v2](#).

ⓘ Important

If your workspace uses a private endpoint, it will automatically have the `v1_legacy_mode` flag enabled, preventing usage of v2 APIs. See [how to configure network isolation with v2](#) for details.

Connection (workspace connection in v1)

Workspace connections from v1 are persisted on the workspace, and fully available with v2.

For a comparison of SDK v1 and v2 code, see [Workspace management in SDK v1 and SDK v2](#).

Datastore

Object storage datastore types created with v1 are fully available for use in v2. Database datastores are not supported; export to object storage (usually Azure Blob) is the recommended migration path.

For a comparison of SDK v1 and v2 code, see [Datastore management in SDK v1 and SDK v2](#).

Compute

Compute of type `AmlCompute` and `ComputeInstance` are fully available for use in v2.

For a comparison of SDK v1 and v2 code, see [Compute management in SDK v1 and SDK v2](#).

Endpoint and deployment (endpoint and web service in v1)

With SDK/CLI v1, you can deploy models on ACI or AKS as web services. Your existing v1 model deployments and web services will continue to function as they are, but Using SDK/CLI v1 to deploy models on ACI or AKS as web services is now considered as **legacy**. For new model deployments, we recommend upgrading to v2. In v2, we offer [managed endpoints](#) or [Kubernetes endpoints](#). The following table guides our recommendation:

Endpoint type in v2	Upgrade from	Notes
Local	ACI	Quick test of model deployment locally; not for production.
Managed online endpoint	ACI, AKS	Enterprise-grade managed model deployment infrastructure with near real-time responses and massive scaling for production.
Managed batch endpoint	ParallelRunStep in a pipeline for batch scoring	Enterprise-grade managed model deployment infrastructure with massively parallel batch processing for production.

Endpoint type in v2	Upgrade from	Notes
Azure Kubernetes Service (AKS)	ACI, AKS	Manage your own AKS cluster(s) for model deployment, giving flexibility and granular control at the cost of IT overhead.
Azure Arc Kubernetes	N/A	Manage your own Kubernetes cluster(s) in other clouds or on-premises, giving flexibility and granular control at the cost of IT overhead.

For a comparison of SDK v1 and v2 code, see [Upgrade deployment endpoints to SDK v2](#).

For migration steps from your existing ACI web services to managed online endpoints, see our [upgrade guide article](#) and [blog ↗](#).

Jobs (experiments, runs, pipelines in v1)

In v2, "experiments", "runs", and "pipelines" are consolidated into jobs. A job has a type. Most jobs are `command` jobs that run a command, like `python main.py`. What runs in a job is agnostic to any programming language, so you can run `bash` scripts, invoke `python` interpreters, run a bunch of `curl` commands, or anything else. Another common type of job is `pipeline`, which defines child jobs that may have input/output relationships, forming a directed acyclic graph (DAG).

For a comparison of SDK v1 and v2 code, see

- [Run a script](#)
- [Local runs](#)
- [Hyperparameter tuning](#)
- [Parallel Run](#)
- [Pipelines](#)
- [AutoML](#)

Designer

You can use designer to build pipelines using your own v2 custom components and the new prebuilt components from registry. In this situation, you can use v1 or v2 data assets in your pipeline.

You can continue to use designer to build pipelines using classic prebuilt components and v1 dataset types (tabular, file). You cannot use existing designer classic prebuilt components with v2 data asset.

You cannot build a pipeline using both existing designer classic prebuilt components and v2 custom components.

Data (datasets in v1)

Datasets are renamed to data assets. *Backwards compatibility* is provided, which means you can use V1 Datasets in V2. When you consume a V1 Dataset in a V2 job you will notice they are automatically mapped into V2 types as follows:

- V1 FileDataset = V2 Folder (`uri_folder`)
- V1 TabularDataset = V2 Table (`mltable`)

It should be noted that *forwards compatibility* is **not** provided, which means you **cannot** use V2 data assets in V1.

This article talks more about handling data in v2 - [Read and write data in a job](#)

For a comparison of SDK v1 and v2 code, see [Data assets in SDK v1 and v2](#).

Model

Models created from v1 can be used in v2.

For a comparison of SDK v1 and v2 code, see [Model management in SDK v1 and SDK v2](#)

Environment

Environments created from v1 can be used in v2. In v2, environments have new features like creation from a local Docker context.

Managing secrets

The management of Key Vault secrets differs significantly in V2 compared to V1. The V1 `set_secret` and `get_secret` SDK methods are not available in V2. Instead, direct access using Key Vault client libraries should be used.

For details about Key Vault, see [Use authentication credential secrets in Azure Machine Learning training jobs](#).

Scenarios across the machine learning lifecycle

There are a few scenarios that are common across the machine learning lifecycle using Azure Machine Learning. We'll look at a few and give general recommendations for upgrading to v2.

Azure setup

Azure recommends Azure Resource Manager templates (often via Bicep for ease of use) to create resources. The same is a good approach for creating Azure Machine Learning resources as well.

If your team is only using Azure Machine Learning, you may consider provisioning the workspace and any other resources via YAML files and CLI instead.

Prototyping models

We recommend v2 for prototyping models. You may consider using the CLI for an interactive use of Azure Machine Learning, while your model training code is Python or any other programming language. Alternatively, you may adopt a full-stack approach with Python solely using the Azure Machine Learning SDK or a mixed approach with the Azure Machine Learning Python SDK and YAML files.

Production model training

We recommend v2 for production model training. Jobs consolidate the terminology and provide a set of consistency that allows for easier transition between types (for example, `command` to `sweep`) and a GitOps-friendly process for serializing jobs into YAML files.

With v2, you should separate your machine learning code from the control plane code. This separation allows for easier iteration and allows for easier transition between local and cloud. We also recommend using MLflow for tracking and model logging. See the [MLflow concept article](#) for details.

Production model deployment

We recommend v2 for production model deployment. Managed endpoints abstract the IT overhead and provide a performant solution for deploying and scoring models, both for online (near real-time) and batch (massively parallel) scenarios.

Kubernetes deployments are supported in v2 through AKS or Azure Arc, enabling Azure cloud and on-premises deployments managed by your organization.

Machine learning operations (MLOps)

A MLOps workflow typically involves CI/CD through an external tool. Typically a CLI is used in CI/CD, though you can alternatively invoke Python or directly use REST.

The solution accelerator for MLOps with v2 is being developed at <https://github.com/Azure/mlops-v2> and can be used as reference or adopted for setup and automation of the machine learning lifecycle.

A note on GitOps with v2

A key paradigm with v2 is serializing machine learning entities as YAML files for source control with `git`, enabling better GitOps approaches than were possible with v1. For instance, you could enforce policy by which only a service principal used in CI/CD pipelines can create/update/delete some or all entities, ensuring changes go through a governed process like pull requests with required reviewers. Since the files in source control are YAML, they're easy to diff and track changes over time. You and your team may consider shifting to this paradigm as you upgrade to v2.

You can obtain a YAML representation of any entity with the CLI via `az ml <entity> show --output yaml`. Note that this output will have system-generated properties, which can be ignored or deleted.

Next steps

- [Get started with the CLI \(v2\)](#)
 - [Get started with the Python SDK \(v2\)](#)
-

Additional resources

Documentation

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[CLI \(v2\) mltable YAML schema - Azure Machine Learning](#)

Reference documentation for the CLI (v2) MLTable YAML schema.

[Customize outputs in batch deployments - Azure Machine Learning](#)

Learn how to create deployments that generate custom outputs and files.

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[az ml batch-endpoint](#)

[Upgrade deployment endpoints to SDK v2 - Azure Machine Learning](#)

Upgrade deployment endpoints from v1 to v2 of Azure Machine Learning SDK

[Create compute clusters CLI v1 - Azure Machine Learning](#)

Learn how to create compute clusters in your Azure Machine Learning workspace with CLI v1. Use the compute cluster as a compute target for training or inference.

[How to administrate data authentication - Azure Machine Learning](#)

Learn how to manage data access and how to authenticate in Azure Machine Learning

[Show 5 more](#)

[Training](#)

Learning paths and modules

[Train models in Azure Machine Learning with the CLI \(v2\) - Training](#)

Train models in Azure Machine Learning with the CLI (v2)

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Upgrade workspace management to SDK v2

Article • 09/29/2022 • 2 minutes to read

The workspace functionally remains unchanged with the V2 development platform. However, there are network-related changes to be aware of. For details, see [Network Isolation Change with Our New API Platform on Azure Resource Manager](#)

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Create a workspace

- SDK v1

```
Python

from azureml.core import Workspace

ws = Workspace.create(
    name='my_workspace',
    location='eastus',
    subscription_id = '<SUBSCRIPTION_ID>'
    resource_group = '<RESOURCE_GROUP>'
)
```

- SDK v2

```
Python

from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

# specify the details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>

# get a handle to the subscription
ml_client = MLClient(DefaultAzureCredential(), subscription_id,
resource_group)

# specify the workspace details
ws = Workspace(
    name="my_workspace",
    location="eastus",
    display_name="My workspace",
```

```
        description="This example shows how to create a workspace",
        tags=dict(purpose="demo"),
    )

    ml_client.workspaces.begin_create(ws)
```

Create a workspace for use with Azure Private Link endpoints

- SDK v1

Python

```
from azureml.core import Workspace

ws = Workspace.create(
    name='my_workspace',
    location='eastus',
    subscription_id = '<SUBSCRIPTION_ID>',
    resource_group = '<RESOURCE_GROUP>'
)

ple = PrivateEndPointConfig(
    name='my_private_link_endpoint',
    vnet_name='<VNET_NAME>',
    vnet_subnet_name='<VNET_SUBNET_NAME>',
    vnet_subscription_id='<SUBSCRIPTION_ID>',
    vnet_resource_group='<RESOURCE_GROUP>'
)

ws.add_private_endpoint(ple, private_endpoint_auto_approval=True)
```

- SDK v2

Python

```
from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

# specify the details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"

# get a handle to the subscription
ml_client = MLClient(DefaultAzureCredential(), subscription_id,
resource_group)

ws = Workspace(
```

```

        name="private_link_endpoint_workspace",
        location="eastus",
        display_name="Private Link endpoint workspace",
        description="When using private link, you must set the
image_build_compute property to a cluster name to use for Docker image
environment building. You can also specify whether the workspace should
be accessible over the internet.",
        image_build_compute="cpu-compute",
        public_network_access="Disabled",
        tags=dict(purpose="demonstration"),
    )

ml_client.workspaces.begin_create(ws)

```

Load/connect to workspace using parameters

- SDK v1

Python

```

from azureml.core import Workspace
ws = Workspace.from_config()

# specify the details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>

# get handle on the workspace
ws = Workspace.get(
    subscription_id='<SUBSCRIPTION_ID>',
    resource_group='<RESOURCE_GROUP>',
    name='my_workspace',
)

```

- SDK v2

Python

```

from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

# specify the details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>

# get handle on the workspace
ws = MLClient(
    DefaultAzureCredential(),
    subscription_id='<SUBSCRIPTION_ID>',

```

```
        resource_group_name='<RESOURCE_GROUP>',
        workspace_name='my_workspace'
    )
```

Load/connect to workspace using config file

- SDK v1

Python

```
from azureml.core import Workspace

ws = Workspace.from_config()
ws.get_details()
```

- SDK v2

Python

```
from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

ws = MLClient.from_config(
    DefaultAzureCredential()
)
```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
Method/API in SDK v1 (use links to ref docs)	Method/API in SDK v2 (use links to ref docs)

Related documents

For more information, see:

- [What is a workspace?](#)

Upgrade compute management to v2

Article • 02/14/2023 • 2 minutes to read

The compute management functionally remains unchanged with the v2 development platform.

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Create compute instance

- SDK v1

Python

```
import datetime
import time

from azureml.core.compute import ComputeTarget, ComputeInstance
from azureml.core.compute_target import ComputeTargetException

# Compute Instances need to have a unique name across the region.
# Here, we create a unique name with current datetime
ci_basic_name = "basic-ci" +
datetime.datetime.now().strftime("%Y%m%d%H%M")

compute_config = ComputeInstance.provisioning_configuration(
    vm_size='STANDARD_DS3_V2'
)
instance = ComputeInstance.create(ws, ci_basic_name ,
compute_config)
instance.wait_for_completion(show_output=True)
```

- SDK v2

Python

```
# Compute Instances need to have a unique name across the region.
# Here, we create a unique name with current datetime
from azure.ai.ml.entities import ComputeInstance, AmlCompute
import datetime

ci_basic_name = "basic-ci" +
datetime.datetime.now().strftime("%Y%m%d%H%M")
ci_basic = ComputeInstance(name=ci_basic_name, size="STANDARD_DS3_V2",
idle_time_before_shutdown_minutes="30")
ml_client.begin_create_or_update(ci_basic)
```

Create compute cluster

- SDK v1

Python

```
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

# Choose a name for your CPU cluster
cpu_cluster_name = "cpucluster"
compute_config =
    AmlCompute.provisioning_configuration(vm_size='STANDARD_DS3_V2',
                                           max_nodes=4)
cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name,
                                   compute_config)
cpu_cluster.wait_for_completion(show_output=True)
```

- SDK v2

Python

```
from azure.ai.ml.entities import AmlCompute
cpu_cluster_name = "cpucluster"
cluster_basic = AmlCompute(
    name=cpu_cluster_name,
    type="amlcompute",
    size="STANDARD_DS3_V2",
    max_instances=4
)
ml_client.begin_create_or_update(cluster_basic)
```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
Method/API in SDK v1 (use links to ref docs)	Method/API in SDK v2 (use links to ref docs)

Next steps

- [Create and manage an Azure Machine Learning compute instance](#)
- [Create an Azure Machine Learning compute cluster](#)

Additional resources

Documentation

[azure.ai.ml.operations.DatastoreOperations class](#)

Represents a client for performing operations on Datastores. You should not instantiate this class directly. Instead, you should create MLClient and use this client via the property MLClient.datastores

[CLI \(v2\) sweep job YAML schema - Azure Machine Learning](#)

Reference documentation for the CLI (v2) sweep job YAML schema.

[Customize outputs in batch deployments - Azure Machine Learning](#)

Learn how create deployments that generate custom outputs and files.

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.ManagedOnlineDeployment class](#)

Managed Online endpoint deployment entity.

[azure.ai.ml.entities.Datastore class](#)

Datastore of an Azure ML workspace, abstract class.

[Upgrade deployment endpoints to SDK v2 - Azure Machine Learning](#)

Upgrade deployment endpoints from v1 to v2 of Azure Machine Learning SDK

[CLI \(v2\) data YAML schema - Azure Machine Learning](#)

Reference documentation for the CLI (v2) data YAML schema.

[Show 5 more](#)

Upgrade datastore management to SDK v2

Article • 02/24/2023 • 2 minutes to read

Azure Machine Learning Datastores securely keep the connection information to your data storage on Azure, so you don't have to code it in your scripts. V2 Datastore concept remains mostly unchanged compared with V1. The difference is we won't support SQL-like data sources via Azure Machine Learning Datastores. We'll support SQL-like data sources via Azure Machine Learning data import&export functionalities.

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Create a datastore from an Azure Blob container via account_key

- SDK v1

Python

```
blob_datastore_name='azblobsdk' # Name of the datastore to workspace
container_name=os.getenv("BLOB_CONTAINER", "<my-container-name>") #
Name of Azure blob container
account_name=os.getenv("BLOB_ACCOUNTNAME", "<my-account-name>") #
Storage account name
account_key=os.getenv("BLOB_ACCOUNT_KEY", "<my-account-key>") # Storage
account access key

blob_datastore = Datastore.register_azure_blob_container(workspace=ws,
datastore_name=blob_datastore_name,
container_name=container_name,
account_name=account_name,
account_key=account_key)
```

- SDK v2

Python

```
from azure.ai.ml.entities import AzureBlobDatastore
from azure.ai.ml import MLClient
```

```

ml_client = MLClient.from_config()

store = AzureBlobDatastore(
    name="blob-protocol-example",
    description="Datastore pointing to a blob container using wasbs
protocol.",
    account_name="mytestblobstore",
    container_name="data-container",
    protocol="wasbs",
    credentials={
        "account_key":
"XXXXXXXXXXXXXXxXXXXXXXXXXXXXXxXXXXXXXXXXXXXXxXXXXXXXXXXXXXXxXXXXXXXXXXXXXX
XXXXXXXXXXXXXXxXXXXXXXXXXXXXX"
    },
)

ml_client.create_or_update(store)

```

Create a datastore from an Azure Blob container via sas_token

- SDK v1

Python

```

blob_datastore_name='azblobsdk' # Name of the datastore to workspace
container_name=os.getenv("BLOB_CONTAINER", "<my-container-name>") #
Name of Azure blob container
sas_token=os.getenv("BLOB_SAS_TOKEN", "<my-sas-token>") # Sas token

blob_datastore = Datastore.register_azure_blob_container(workspace=ws,
datastore_name=blob_datastore_name,
container_name=container_name,
sas_token=sas_token)

```

- SDK v2

Python

```

from azure.ai.ml.entities import AzureBlobDatastore
from azure.ai.ml import MLClient

ml_client = MLClient.from_config()

store = AzureBlobDatastore(
    name="blob-sas-example",

```

```

        description="Datastore pointing to a blob container using SAS
token.",
        account_name="mytestblobstore",
        container_name="data-container",
        credentials=SasTokenCredentials(
            sas_token= "?xx=XXXX-XX-
XX&xx=xxxx&xxx=xx=xxxxxx&xx=XXXX-XX-XXXXX:XX:XXX&xx=XXXX-XX-
XXXXX:XX:XXX&xxx=xxxxx&xxx=XXXxxxxxxxxxxxxxxxxxxxxxxXXXXxxXXXXxXXXXxXXXXxXXXX"
            ),
        )
    )

ml_client.create_or_update(store)

```

Create a datastore from an Azure Blob container via identity-based authentication

- SDK v1

Python

```

blob_datastore = Datastore.register_azure_blob_container(workspace=ws,
datastore_name='credentialless_blob',
container_name='my_container_name',
account_name='my_account_name')

```

- SDK v2

Python

```

from azure.ai.ml.entities import AzureBlobDatastore
from azure.ai.ml import MLClient

ml_client = MLClient.from_config()

store = AzureBlobDatastore(
    name="",
    description="",
    account_name="",
    container_name=""
)

ml_client.create_or_update(store)

```

Get datastores from your workspace

- SDK v1

```
Python
```

```
# Get a named datastore from the current workspace
datastore = Datastore.get(ws, datastore_name='your datastore name')
```

```
Python
```

```
# List all datastores registered in the current workspace
datastores = ws.datastores
for name, datastore in datastores.items():
    print(name, datastore.datastore_type)
```

- SDK v2

```
Python
```

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

#Enter details of your Azure Machine Learning workspace
subscription_id = '<SUBSCRIPTION_ID>'
resource_group = '<RESOURCE_GROUP>'
workspace_name = '<AZUREML_WORKSPACE_NAME>'

ml_client = MLClient(credential=DefaultAzureCredential(),
                     subscription_id=subscription_id,
                     resource_group_name=resource_group)

datastore = ml_client.datastores.get(name='your datastore name')
```

Mapping of key functionality in SDK v1 and SDK v2

Storage types in SDK v1	Storage types in SDK v2
azureml_blob_datastore	azureml_blob_datastore
azureml_data_lake_gen1_datastore	azureml_data_lake_gen1_datastore
azureml_data_lake_gen2_datastore	azureml_data_lake_gen2_datastore
azureml_sql_database_datastore	Will be supported via import & export functionalities

Storage types in SDK v1	Storage types in SDK v2
<code>azureml_my_sql_datastore</code>	Will be supported via import & export functionalities
<code>azureml_postgre_sql_datastore</code>	Will be supported via import & export functionalities

Next steps

For more information, see:

- [Create datastores](#)
- [Read and write data in a job](#)
- [V2 datastore operations](#)

Additional resources

Documentation

[Upgrade script run to SDK v2 - Azure Machine Learning](#)

Upgrade how to run a script from SDK v1 to SDK v2

[Upgrade data management to SDK v2 - Azure Machine Learning](#)

Upgrade data management from v1 to v2 of Azure Machine Learning SDK

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.Data class](#)

Data for training and scoring.

[azureml.data.abstract_dataset.AbstractDataset class - Azure Machine Learning Python](#)

Base class of datasets in Azure Machine Learning. Please reference TabularDatasetFactory class and FileDatasetFactory class to create instances of dataset.

[azure.ai.ml.entities.ManagedOnlineDeployment class](#)

Managed Online endpoint deployment entity.

[azureml-pipeline-core package - Azure Machine Learning Python](#)

[Show 5 more](#)

Upgrade data management to SDK v2

Article • 02/24/2023 • 2 minutes to read

In V1, an Azure Machine Learning dataset can either be a `Filedataset` or a `Tabulardataset`. In V2, an Azure Machine Learning data asset can be a `uri_folder`, `uri_file` or `mltable`. You can conceptually map `Filedataset` to `uri_folder` and `uri_file`, `Tabulardataset` to `mltable`.

- URIs (`uri_folder`, `uri_file`) - a Uniform Resource Identifier that is a reference to a storage location on your local computer or in the cloud, that makes it easy to access data in your jobs.
- MLTable - a method to abstract the tabular data schema definition, to make it easier for consumers of that data to materialize the table into a Pandas/Dask/Spark dataframe.

This article compares data scenario(s) in SDK v1 and SDK v2.

Create a `filedataset` / `uri` type of data asset

- SDK v1 - Create a `Filedataset`

Python

```
from azureml.core import Workspace, Datastore, Dataset

# create a FileDataset pointing to files in 'animals' folder and its
# subfolders recursively
datastore_paths = [(datastore, 'animals')]
animal_ds = Dataset.File.from_files(path=datastore_paths)

# create a FileDataset from image and label files behind public web
# urls
web_paths =
['https://azureopendatastorage.blob.core.windows.net/mnist/train-
images-idx3-ubyte.gz',
 'https://azureopendatastorage.blob.core.windows.net/mnist/train-labels-
idx1-ubyte.gz']
mnist_ds = Dataset.File.from_files(path=web_paths)
```

- SDK v2

- Create a `URI_FOLDER` type data asset

Python

```
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

# Supported paths include:
# local: './<path>'
# blob:
'https://<account_name>.blob.core.windows.net/<container_name>/<path>'
# ADLS gen2:
'abfss://<file_system>@<account_name>.dfs.core.windows.net/<path>/'
# Datastore: 'azureml://datastores/<data_store_name>/paths/<path>'

my_path = '<path>'

my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FOLDER,
    description=<description>,
    name=<name>,
    version='<version>'
)

ml_client.data.create_or_update(my_data)
```

- Create a `URI_FILE` type data asset.

Python

```
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

# Supported paths include:
# local: './<path>/<file>'
# blob:
'https://<account_name>.blob.core.windows.net/<container_name>/<path>/<file>'
# ADLS gen2:
'abfss://<file_system>@<account_name>.dfs.core.windows.net/<path>/<file>'
# Datastore:
'azureml://datastores/<data_store_name>/paths/<path>/<file>'

my_path = '<path>'

my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FILE,
    description=<description>,
    name=<name>,
    version=<version>
)
```

```
ml_client.data.create_or_update(my_data)
```

Create a tabular dataset/data asset

- SDK v1

Python

```
from azureml.core import Workspace, Datastore, Dataset

datastore_name = 'your datastore name'

# get existing workspace
workspace = Workspace.from_config()

# retrieve an existing datastore in the workspace by name
datastore = Datastore.get(workspace, datastore_name)

# create a TabularDataset from 3 file paths in datastore
datastore_paths = [(datastore, 'weather/2018/11.csv'),
                    (datastore, 'weather/2018/12.csv'),
                    (datastore, 'weather/2019/*.csv')]

weather_ds = Dataset.Tabular.from_delimited_files(path=datastore_paths)
```

- SDK v2 - Create `mltable` data asset via yaml definition

YAML

```
type: mltable

paths:
  - pattern: ./*.txt
transformations:
  - read_delimited:
      delimiter: ,
      encoding: ascii
      header: all_files_same_headers
```

Python

```
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

# my_path must point to folder containing MLTable artifact (MLTable
file + data
# Supported paths include:
```

```

# local: './<path>'
# blob:
'https://<account_name>.blob.core.windows.net/<container_name>/<path>'
# ADLS gen2:
'abfss://<file_system>@<account_name>.dfs.core.windows.net/<path>/'
# Datastore: 'azureml://datastores/<data_store_name>/paths/<path>'

my_path = '<path>'

my_data = Data(
    path=my_path,
    type=AssetTypes.MLTABLE,
    description=<description>,
    name=<name>,
    version='<version>'
)

ml_client.data.create_or_update(my_data)

```

Use data in an experiment/job

- SDK v1

Python

```

from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory=script_folder,
                      script='train_titanic.py',
                      # pass dataset as an input with friendly name
'start'
                      arguments=[ '--input-data',
titanic_ds.as_named_input('titanic')],
                      compute_target=compute_target,
                      environment=myenv)

# Submit the run configuration for your training run
run = experiment.submit(src)
run.wait_for_completion(show_output=True)

```

- SDK v2

Python

```

from azure.ai.ml import command
from azure.ai.ml.entities import Data
from azure.ai.ml import Input, Output
from azure.ai.ml.constants import AssetTypes

# Possible Asset Types for Data:

```

```

# AssetTypes.URI_FILE
# AssetTypes.URI_FOLDER
# AssetTypes.MLTABLE

# Possible Paths for Data:
# Blob:
https://<account_name>.blob.core.windows.net/<container_name>/<folder>/
<file>
# Datastore: azureml://datastores/paths/<folder>/<file>
# Data Asset: azureml:<my_data>:<version>

my_job_inputs = {
    "raw_data": Input(type=AssetTypes.URI_FOLDER, path="<path>")
}

my_job_outputs = {
    "prep_data": Output(type=AssetTypes.URI_FOLDER, path="<path>")
}

job = command(
    code=".src", # local path where the code is stored
    command="python process_data.py --raw_data ${inputs.raw_data} --"
    prep_data ${outputs.prep_data}",
    inputs=my_job_inputs,
    outputs=my_job_outputs,
    environment="<environment_name>:<version>",
    compute="cpu-cluster",
)

# submit the command
returned_job = ml_client.create_or_update(job)
# get a URL for the status of the job
returned_job.services["Studio"].endpoint

```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
Method/API in SDK v1	Method/API in SDK v2

Next steps

For more information, see the documentation here:

- [Data in Azure Machine Learning](#)
- [Create data_assets](#)
- [Read and write data in a job](#)

- V2 datastore operations
-

Additional resources

Documentation

[Upgrade script run to SDK v2 - Azure Machine Learning](#)

Upgrade how to run a script from SDK v1 to SDK v2

[Upgrade datastore management to SDK v2 - Azure Machine Learning](#)

Upgrade datastore management from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.Data class](#)

Data for training and scoring.

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[azureml-pipeline-core package - Azure Machine Learning Python](#)

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.ManagedOnlineDeployment class](#)

Managed Online endpoint deployment entity.

[azure.ai.ml.entities.Model class](#)

Model for training and scoring.

[Show 5 more](#)

Upgrade model management to SDK v2

Article • 12/02/2022 • 2 minutes to read

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Create model

- SDK v1

Python

```
import urllib.request
from azureml.core.model import Model

# Register model
model = Model.register(ws, model_name="local-file-example",
model_path="mlflow-model/model.pkl")
```

- SDK v2

Python

```
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import AssetTypes

file_model = Model(
    path="mlflow-model/model.pkl",
    type=AssetTypes.CUSTOM_MODEL,
    name="local-file-example",
    description="Model created from local file."
)
ml_client.models.create_or_update(file_model)
```

Use model in an experiment/job

- SDK v1

Python

```
model = run.register_model(model_name='run-model-example',
model_path='outputs/model/')
print(model.name, model.id, model.version, sep='\t')
```

- SDK v2

Python

```
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import AssetTypes

run_model = Model(
    path="azureml://jobs/$RUN_ID/outputs/artifacts/paths/model/",
    name="run-model-example",
    description="Model created from run.",
    type=AssetTypes.CUSTOM_MODEL
)

ml_client.models.create_or_update(run_model)
```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
Model.register	ml_client.models.create_or_update
run.register_model	ml_client.models.create_or_update
Model.deploy	ml_client.begin_create_or_update(blue_deployment)

Next steps

For more information, see the documentation here:

- [Create a model in v1](#)
- [Deploy a model in v1](#)
- [Create a model in v2](#)
- [Deploy a model in v2](#)

Upgrade script run to SDK v2

Article • 02/24/2023 • 2 minutes to read

In SDK v2, "experiments" and "runs" are consolidated into jobs.

A job has a type. Most jobs are command jobs that run a `command`, like `python main.py`.

What runs in a job is agnostic to any programming language, so you can run `bash`

scripts, invoke `python` interpreters, run a bunch of `curl` commands, or anything else.

To upgrade, you'll need to change your code for submitting jobs to SDK v2. What you run *within* the job doesn't need to be upgraded to SDK v2. However, it's recommended to remove any code specific to Azure Machine Learning from your model training scripts. This separation allows for an easier transition between local and cloud and is considered best practice for mature MLOps. In practice, this means removing `azureml.*` lines of code. Model logging and tracking code should be replaced with MLflow. For more details, see [how to use MLflow in v2](#).

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Submit a script run

- SDK v1

Python

```
from azureml.core import Workspace, Experiment, Environment,
ScriptRunConfig

# connect to the workspace
ws = Workspace.from_config()

# define and configure the experiment
experiment = Experiment(workspace=ws, name='day1-experiment-train')
config = ScriptRunConfig(source_directory='./src',
                        script='train.py',
                        compute_target='cpu-cluster')

# set up pytorch environment
env = Environment.from_conda_specification(
    name='pytorch-env',
    file_path='pytorch-env.yml')
config.run_config.environment = env

run = experiment.submit(config)
```

```
aml_url = run.get_portal_url()
print(aml_url)
```

- SDK v2

Python

```
#import required libraries
from azure.ai.ml import MLClient, command
from azure.ai.ml.entities import Environment
from azure.identity import DefaultAzureCredential

#connect to the workspace
ml_client = MLClient.from_config(DefaultAzureCredential())

# set up pytorch environment
env = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    conda_file="pytorch-env.yml",
    name="pytorch-env"
)

# define the command
command_job = command(
    code="./src",
    command="train.py",
    environment=env,
    compute="cpu-cluster",
)

returned_job = ml_client.jobs.create_or_update(command_job)
returned_job
```

Mapping of key functionality in v1 and v2

Functionality in SDK v1	Rough mapping in SDK v2
experiment.submit	MLClient.jobs.create_or_update
ScriptRunConfig()	command()

Next steps

For more information, see:

- [V1 - Experiment](#)
- [V2 - Command Job](#)

- Train models with the Azure Machine Learning Python SDK v2
-

Additional resources

Documentation

[Upgrade data management to SDK v2 - Azure Machine Learning](#)

Upgrade data management from v1 to v2 of Azure Machine Learning SDK

[Upgrade datastore management to SDK v2 - Azure Machine Learning](#)

Upgrade datastore management from v1 to v2 of Azure Machine Learning SDK

[azureml-pipeline-core package - Azure Machine Learning Python](#)

[azure.ai.ml.entities.Data class](#)

Data for training and scoring.

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[azureml.data.abstract_dataset.AbstractDataset class - Azure Machine Learning Python](#)

Base class of datasets in Azure Machine Learning. Please reference TabularDatasetFactory class and FileDatasetFactory class to create instances of dataset.

[azure.ai.ml.entities.Datastore class](#)

Datastore of an Azure ML workspace, abstract class.

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[Show 5 more](#)

Upgrade local runs to SDK v2

Article • 02/02/2023 • 2 minutes to read

Local runs are similar in both V1 and V2. Use the "local" string when setting the compute target in either version.

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Submit a local run

- SDK v1

Python

```
from azureml.core import Workspace, Experiment, Environment,
ScriptRunConfig

# connect to the workspace
ws = Workspace.from_config()

# define and configure the experiment
experiment = Experiment(workspace=ws, name='day1-experiment-train')
config = ScriptRunConfig(source_directory='./src',
                        script='train.py',
                        compute_target='local')

# set up pytorch environment
env = Environment.from_conda_specification(
    name='pytorch-env',
    file_path='pytorch-env.yml')
config.run_config.environment = env

run = experiment.submit(config)

aml_url = run.get_portal_url()
print(aml_url)
```

- SDK v2

Python

```
#import required libraries
from azure.ai.ml import MLClient, command
from azure.ai.ml.entities import Environment
from azure.identity import DefaultAzureCredential

#connect to the workspace
ml_client = MLClient.from_config(DefaultAzureCredential())
```

```

# set up pytorch environment
env = Environment(
    image='mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04',
    conda_file='pytorch-env.yml',
    name='pytorch-env'
)

# define the command
command_job = command(
    code='./src',
    command='train.py',
    environment=env,
    compute='local',
)

returned_job = ml_client.jobs.create_or_update(command_job)
returned_job

```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
<code>experiment.submit</code>	<code>MLClient.jobs.create_or_update</code>

Next steps

- Train models with Azure Machine Learning
-

Additional resources

Documentation

[Upgrade script run to SDK v2 - Azure Machine Learning](#)

Upgrade how to run a script from SDK v1 to SDK v2

[azure.ai.ml.entities.Datastore class](#)

Datastore of an Azure ML workspace, abstract class.

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[azure.ai.ml.Output class](#)

Define an output of a Component or Job.

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.Environment class](#)

Environment for training.

[CLI \(v2\) core YAML syntax - Azure Machine Learning](#)

Overview CLI (v2) core YAML syntax.

[Show 5 more](#)

Upgrade AutoML to SDK v2

Article • 02/10/2023 • 2 minutes to read

In SDK v2, "experiments" and "runs" are consolidated into jobs.

In SDK v1, AutoML was primarily configured and run using the `AutoMLConfig` class. In SDK v2, this class has been converted to an `AutoML` job. Although there are some differences in the configuration options, by and large, naming & functionality has been preserved in V2.

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Submit AutoML run

- SDK v1: Below is a sample AutoML classification task. For the entire code, check out our [examples repo ↗](#).

Python

```
# Imports

import azureml.core
from azureml.core.experiment import Experiment
from azureml.core.workspace import Workspace
from azureml.core.dataset import Dataset
from azureml.train.automl import AutoMLConfig
from azureml.train.automl.run import AutoMLRun

# Load tabular dataset
data = "<url_to_data>"
dataset = Dataset.Tabular.from_delimited_files(data)
training_data, validation_data = dataset.random_split(percentage=0.8,
seed=223)
label_column_name = "Class"

# Configure Auto ML settings
automl_settings = {
    "n_cross_validations": 3,
    "primary_metric": "average_precision_score_weighted",
    "enable_early_stopping": True,
    "max_concurrent_iterations": 2,
    "experiment_timeout_hours": 0.25,
    "verbosity": logging.INFO,
}

# Put together an AutoML job constructor
automl_config = AutoMLConfig(
    task="classification",
```

```

        debug_log="automl_errors.log",
        compute_target=compute_target,
        training_data=training_data,
        label_column_name=label_column_name,
        **automl_settings,
    )

    # Submit run
    remote_run = experiment.submit(automl_config, show_output=False)
    azureml_url = remote_run.get_portal_url()
    print(azureml_url)

```

- SDK v2: Below is a sample AutoML classification task. For the entire code, check out our [examples repo ↗](#).

Python

```

# Imports
from azure.ai.ml import automl, Input, MLClient

from azure.ai.ml.constants import AssetTypes
from azure.ai.ml.automl import (
    classification,
    ClassificationPrimaryMetrics,
    ClassificationModels,
)

# Create MLTables for training dataset
# Note that AutoML Job can also take in tabular data
my_training_data_input = Input(
    type=AssetTypes.MLTABLE, path="./data/training-mltable-folder"
)

# Create the AutoML classification job with the related factory-
# function.
classification_job = automl.classification(
    compute=<compute_name>,
    experiment_name=<exp_name?>,
    training_data=my_training_data_input,
    target_column_name=<name_of_target_column>,
    primary_metric="accuracy",
    n_cross_validations=5,
    enable_model_explainability=True,
    tags={"my_custom_tag": "My custom value"},
)

# Limits are all optional
classification_job.set_limits(
    timeout_minutes=600,
    trial_timeout_minutes=20,
    max_trials=5,
    max_concurrent_trials = 4,
)

```

```

        max_cores_per_trial= 1,
        enable_early_termination=True,
    )

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

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

```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
Method/API in SDK v1 (use links to ref docs)	Method/API in SDK v2 (use links to ref docs)

Next steps

For more information, see:

- [How to train an AutoML model with Python SDKv2](#)

Additional resources

Documentation

[azure.ai.ml.automl package](#)

Contains automated machine learning classes for Azure Machine Learning SDKv2. Main areas include managing AutoML tasks.

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.operations.DatastoreOperations class](#)

Represents a client for performing operations on Datastores. You should not instantiate this class directly. Instead, you should create MLClient and use this client via the property MLClient.datastores

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[Upgrade workspace management to SDK v2 - Azure Machine Learning](#)

Upgrade workspace management from v1 to v2 of Azure Machine Learning SDK

[Migrate logging from SDK v1 to MLflow - Azure Machine Learning](#)

Comparison of SDK v1 logging APIs and MLflow tracking

[azure.ai.ml.entities.Model class](#)

Model for training and scoring.

[Show 5 more](#)

Upgrade hyperparameter tuning to SDK v2

Article • 02/24/2023 • 3 minutes to read

In SDK v2, tuning hyperparameters are consolidated into jobs.

A job has a type. Most jobs are command jobs that run a `command`, like `python main.py`.

What runs in a job is agnostic to any programming language, so you can run `bash` scripts, invoke `python` interpreters, run a bunch of `curl` commands, or anything else.

A sweep job is another type of job, which defines sweep settings and can be initiated by calling the `sweep` method of `command`.

To upgrade, you'll need to change your code for defining and submitting your hyperparameter tuning experiment to SDK v2. What you run *within* the job doesn't need to be upgraded to SDK v2. However, it's recommended to remove any code specific to Azure Machine Learning from your model training scripts. This separation allows for an easier transition between local and cloud and is considered best practice for mature MLOps. In practice, this means removing `azureml.*` lines of code. Model logging and tracking code should be replaced with MLflow. For more information, see [how to use MLflow in v2](#).

This article gives a comparison of scenario(s) in SDK v1 and SDK v2.

Run hyperparameter tuning in an experiment

- SDK v1

Python

```
from azureml.core import ScriptRunConfig, Experiment, Workspace
from azureml.train.hyperdrive import RandomParameterSampling,
BanditPolicy, HyperDriveConfig, PrimaryMetricGoal
from azureml.train.hyperdrive import choice, loguniform

dataset = Dataset.get_by_name(ws, 'mnist-dataset')

# list the files referenced by mnist dataset
dataset.to_path()

#define the search space for your hyperparameters
param_sampling = RandomParameterSampling(
{
    '--batch-size': choice(25, 50, 100),
    '--epoch-count': choice(1, 2, 3),
    '--learning-rate': loguniform(0.001, 0.01)
})
```

```

        '--first-layer-neurons': choice(10, 50, 200, 300, 500),
        '--second-layer-neurons': choice(10, 50, 200, 500),
        '--learning-rate': loguniform(-6, -1)
    }
)

args = ['--data-folder', dataset.as_named_input('mnist').as_mount()]

#Set up your script run
src = ScriptRunConfig(source_directory=script_folder,
                      script='keras_mnist.py',
                      arguments=args,
                      compute_target=compute_target,
                      environment=keras_env)

# Set early stopping on this one
early_termination_policy = BanditPolicy(evaluation_interval=2,
                                         slack_factor=0.1)

# Define the configurations for your hyperparameter tuning experiment
hyperdrive_config = HyperDriveConfig(run_config=src,
                                      hyperparameter_sampling=param_sampling,
                                      policy=early_termination_policy,
                                      primary_metric_name='Accuracy',
                                      primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                                      max_total_runs=20,
                                      max_concurrent_runs=4)

# Specify your experiment details
experiment = Experiment(workspace, experiment_name)

hyperdrive_run = experiment.submit(hyperdrive_config)

#Find the best model
best_run = hyperdrive_run.get_best_run_by_primary_metric()

```

- SDK v2

Python

```

from azure.ai.ml import MLClient
from azure.ai.ml import command, Input
from azure.ai.ml.sweep import Choice, Uniform, MedianStoppingPolicy
from azure.identity import DefaultAzureCredential

# Create your command
command_job_for_sweep = command(
    code=".src",
    command="python main.py --iris-csv ${{inputs.iris_csv}} --learning-
rate ${{inputs.learning_rate}} --boosting ${{inputs.boosting}}",
    environment="AzureML-lightgbm-3.2-ubuntu18.04-py37-cpu@latest",
    inputs={

```

```

    "iris_csv": Input(
        type="uri_file",
    ),
    #define the search space for your hyperparameters
    "learning_rate": Uniform(min_value=0.01, max_value=0.9),
    "boosting": Choice(values=[ "gbdt", "dart"]),
},
compute="cpu-cluster",
)

# Call sweep() on your command job to sweep over your parameter
expressions
sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm="random",
    primary_metric="test-multi_logloss",
    goal="Minimize",
)

# Define the limits for this sweep
sweep_job.set_limits(max_total_trials=20, max_concurrent_trials=10,
timeout=7200)

# Set early stopping on this one
sweep_job.early_termination = MedianStoppingPolicy(delay_evaluation=5,
evaluation_interval=2)

# Specify your experiment details
sweep_job.display_name = "lightgbm-iris-sweep-example"
sweep_job.experiment_name = "lightgbm-iris-sweep-example"
sweep_job.description = "Run a hyperparameter sweep job for LightGBM on
Iris dataset."

# submit the sweep
returned_sweep_job = ml_client.create_or_update(sweep_job)

# get a URL for the status of the job
returned_sweep_job.services["Studio"].endpoint

# Download best trial model output
ml_client.jobs.download(returned_sweep_job.name, output_name="model")

```

Run hyperparameter tuning in a pipeline

- SDK v1

Python

```
tf_env = Environment.get(ws, name='AzureML-TensorFlow-2.0-GPU')
```

```

data_folder = dataset.as_mount()
src = ScriptRunConfig(source_directory=script_folder,
                      script='tf_mnist.py',
                      arguments=['--data-folder', data_folder],
                      compute_target=compute_target,
                      environment=tf_env)

#Define HyperDrive configs
ps = RandomParameterSampling(
    {
        '--batch-size': choice(25, 50, 100),
        '--first-layer-neurons': choice(10, 50, 200, 300, 500),
        '--second-layer-neurons': choice(10, 50, 200, 500),
        '--learning-rate': loguniform(-6, -1)
    }
)

early_termination_policy = BanditPolicy(evaluation_interval=2,
                                         slack_factor=0.1)

hd_config = HyperDriveConfig(run_config=src,
                             hyperparameter_sampling=ps,
                             policy=early_termination_policy,
                             primary_metric_name='validation_acc',

primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
max_total_runs=4,
max_concurrent_runs=4)

metrics_output_name = 'metrics_output'
metrics_data = PipelineData(name='metrics_data',
                           datastore=datastore,
                           pipeline_output_name=metrics_output_name,
                           training_output=TrainingOutput("Metrics"))

model_output_name = 'model_output'
saved_model = PipelineData(name='saved_model',
                           datastore=datastore,
                           pipeline_output_name=model_output_name,
                           training_output=TrainingOutput("Model",

model_file="outputs/model/saved_model.pb"))
#Create HyperDriveStep
hd_step_name='hd_step01'
hd_step = HyperDriveStep(
    name=hd_step_name,
    hyperdrive_config=hd_config,
    inputs=[data_folder],
    outputs=[metrics_data, saved_model])

#Find and register best model
conda_dep = CondaDependencies()
conda_dep.add_pip_package("azureml-sdk")

rcfg = RunConfiguration(conda_dependencies=conda_dep)

```

```

register_model_step = PythonScriptStep(script_name='register_model.py',
                                       name="register_model_step01",
                                       inputs=[saved_model],
                                       compute_target=cpu_cluster,
                                       arguments=[ "--saved-model",
                                       saved_model],
                                       allow_reuse=True,
                                       runconfig=rcfg)

register_model_step.run_after(hd_step)

#Run the pipeline
pipeline = Pipeline(workspace=ws, steps=[hd_step, register_model_step])
pipeline_run = exp.submit(pipeline)

```

- SDK v2

Python

```

train_component_func = load_component(path="../train.yml")
score_component_func = load_component(path="../predict.yml")

# define a pipeline
@pipeline()
def pipeline_with_hyperparameter_sweep():
    """Tune hyperparameters using sample components."""
    train_model = train_component_func(
        data=Input(
            type="uri_file",
            path="wasbs://datasets@azuremlexamples.blob.core.windows.net/iris.csv",
        ),
        c_value=Uniform(min_value=0.5, max_value=0.9),
        kernel=Choice(["rbf", "linear", "poly"]),
        coef0=Uniform(min_value=0.1, max_value=1),
        degree=3,
        gamma="scale",
        shrinking=False,
        probability=False,
        tol=0.001,
        cache_size=1024,
        verbose=False,
        max_iter=-1,
        decision_function_shape="ovr",
        break_ties=False,
        random_state=42,
    )
    sweep_step = train_model.sweep(
        primary_metric="training_f1_score",
        goal="minimize",
        sampling_algorithm="random",
    )

```

```

        compute="cpu-cluster",
    )
    sweep_step.set_limits(max_total_trials=20,
    max_concurrent_trials=10, timeout=7200)

    score_data = score_component_func(
        model=sweep_step.outputs.model_output,
    test_data=sweep_step.outputs.test_data
    )

pipeline_job = pipeline_with_hyperparameter_sweep()

# set pipeline level compute
pipeline_job.settings.default_compute = "cpu-cluster"

# submit job to workspace
pipeline_job = ml_client.jobs.create_or_update(
    pipeline_job, experiment_name="pipeline_samples"
)
pipeline_job

```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
HyperDriveRunConfig()	SweepJob()
hyperdrive Package	sweep Package

Next steps

For more information, see:

- [SDK v1 - Tune Hyperparameters](#)
- [SDK v2 - Tune Hyperparameters](#)
- [SDK v2 - Sweep in Pipeline](#)

Additional resources

 [Documentation](#)

[azure.ai.ml.entities.Datastore class](#)

Datastore of an Azure ML workspace, abstract class.

[Troubleshooting local model deployment - Azure Machine Learning](#)

Try a local model deployment as a first step in troubleshooting model deployment errors.

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[mltable package - Azure Machine Learning Python](#)

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[azureml.pipeline.core.PipelineParameter class - Azure Machine Learning Python](#)

Defines a parameter in a pipeline execution. Use PipelineParameters to construct versatile Pipelines which can be resubmitted later with varying parameter values.

[azureml.pipeline.steps.ParallelRunConfig class - Azure Machine Learning Python](#)

Defines configuration for a ParallelRunStep object. For an example of using ParallelRunStep, see the notebook <https://aka.ms/batch-inference-notebooks>. For troubleshooting guide, see <https://aka.ms/prstsg>. You can find more references there.

[azure.ai.ml.entities.Command class](#)

Base class for command node, used for command component version consumption. You should not instantiate this class directly. Instead, you should create from builder function: command.

[Show 5 more](#)

Training

Learning paths and modules

[Tune hyperparameters with Azure Databricks - Training](#)

Tune hyperparameters with Azure Databricks

Upgrade parallel run step to SDK v2

Article • 02/24/2023 • 5 minutes to read

In SDK v2, "Parallel run step" is consolidated into job concept as `parallel job`. Parallel job keeps the same target to empower users to accelerate their job execution by distributing repeated tasks on powerful multi-nodes compute clusters. On top of parallel run step, v2 parallel job provides extra benefits:

- Flexible interface, which allows user to define multiple custom inputs and outputs for your parallel job. You can connect them with other steps to consume or manage their content in your entry script
- Simplify input schema, which replaces `Dataset` as input by using v2 `data asset` concept. You can easily use your local files or blob directory URI as the inputs to parallel job.
- More powerful features are under developed in v2 parallel job only. For example, resume the failed/canceled parallel job to continue process the failed or unprocessed mini-batches by reusing the successful result to save duplicate effort.

To upgrade your current sdk v1 parallel run step to v2, you'll need to

- Use `parallel_run_function` to create parallel job by replacing `ParallelRunConfig` and `ParallelRunStep` in v1.
- Upgrade your v1 pipeline to v2. Then invoke your v2 parallel job as a step in your v2 pipeline. See [how to upgrade pipeline from v1 to v2](#) for the details about pipeline upgrade.

Note: User `entry script` is compatible between v1 parallel run step and v2 parallel job. So you can keep using the same `entry_script.py` when you upgrade your parallel run job.

This article gives a comparison of scenario(s) in SDK v1 and SDK v2. In the following examples, we'll build a parallel job to predict input data in a pipelines job. You'll see how to build a parallel job, and how to use it in a pipeline job for both SDK v1 and SDK v2.

Prerequisites

- Prepare your SDK v2 environment: [Install the Azure Machine Learning SDK v2 for Python](#)
- Understand the basis of SDK v2 pipeline: [How to create Azure Machine Learning pipeline with Python SDK v2](#)

Create parallel step

- SDK v1

```
Python

# Create the configuration to wrap the inference script
from azureml.pipeline.steps import ParallelRunStep, ParallelRunConfig

parallel_run_config = ParallelRunConfig(
    source_directory=scripts_folder,
    entry_script=script_file,
    mini_batch_size=PipelineParameter(name="batch_size_param", default_value="5"),
    error_threshold=10,
    output_action="append_row",
```

```

        append_row_file_name="mnist_outputs.txt",
        environment=batch_env,
        compute_target=compute_target,
        process_count_per_node=PipelineParameter(name="process_count_param",
        default_value=2),
        node_count=2
    )

    # Create the Parallel run step
    parallelrun_step = ParallelRunStep(
        name="predict-digits-mnist",
        parallel_run_config=parallel_run_config,
        inputs=[ input_mnist_ds_consumption ],
        output=output_dir,
        allow_reuse=False
)

```

- SDK v2

Python

```

# parallel job to process file data
file_batch_inference = parallel_run_function(
    name="file_batch_score",
    display_name="Batch Score with File Dataset",
    description="parallel component for batch score",
    inputs=dict(
        job_data_path=Input(
            type=AssetTypes.MLTABLE,
            description="The data to be split and scored in parallel",
        )
    ),
    outputs=dict(job_output_path=Output(type=AssetTypes.MLTABLE)),
    input_data="${{inputs.job_data_path}}",
    instance_count=2,
    mini_batch_size="1",
    mini_batch_error_threshold=1,
    max_concurrency_per_instance=1,
    task=RunFunction(
        code=".src",
        entry_script="file_batch_inference.py",
        program_arguments="--job_output_path ${{outputs.job_output_path}}",
        environment="azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:1",
    ),
)

```

Use parallel step in pipeline

- SDK v1

Python

```

# Run pipeline with parallel run step
from azureml.core import Experiment

pipeline = Pipeline(workspace=ws, steps=[parallelrun_step])
experiment = Experiment(ws, 'digit_identification')
pipeline_run = experiment.submit(pipeline)
pipeline_run.wait_for_completion(show_output=True)

```

- SDK v2

Python

```
@pipeline()
def parallel_in_pipeline(pipeline_job_data_path, pipeline_score_model):

    prepare_file_tabular_data = prepare_data(input_data=pipeline_job_data_path)
    # output of file & tabular data should be type MLTable
    prepare_file_tabular_data.outputs.file_output_data.type = AssetTypes.MLTABLE
    prepare_file_tabular_data.outputs.tabular_output_data.type = AssetTypes.MLTABLE

    batch_inference_with_file_data = file_batch_inference(
        job_data_path=prepare_file_tabular_data.outputs.file_output_data
    )
    # use eval_mount mode to handle file data
    batch_inference_with_file_data.inputs.job_data_path.mode = (
        InputOutputModes.EVAL_MOUNT
    )
    batch_inference_with_file_data.outputs.job_output_path.type = AssetTypes.MLTABLE

    batch_inference_with_tabular_data = tabular_batch_inference(
        job_data_path=prepare_file_tabular_data.outputs.tabular_output_data,
        score_model=pipeline_score_model,
    )
    # use direct mode to handle tabular data
    batch_inference_with_tabular_data.inputs.job_data_path.mode = (
        InputOutputModes.DIRECT
    )

    return {
        "pipeline_job_out_file": batch_inference_with_file_data.outputs.job_output_path,
        "pipeline_job_out_tabular":
            batch_inference_with_tabular_data.outputs.job_output_path,
    }

    pipeline_job_data_path = Input(
        path=".dataset/", type=AssetTypes.MLTABLE, mode=InputOutputModes.RO_MOUNT
    )
    pipeline_score_model = Input(
        path=".model/", type=AssetTypes.URI_FOLDER, mode=InputOutputModes.DOWNLOAD
    )
    # create a pipeline
    pipeline_job = parallel_in_pipeline(
        pipeline_job_data_path=pipeline_job_data_path,
        pipeline_score_model=pipeline_score_model,
    )
    pipeline_job.outputs.pipeline_job_out_tabular.type = AssetTypes.URI_FILE

    # set pipeline level compute
    pipeline_job.settings.default_compute = "cpu-cluster"

    # run pipeline job
    pipeline_job = ml_client.jobs.create_or_update(
        pipeline_job, experiment_name="pipeline_samples"
    )
```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
-------------------------	-------------------------

Functionality in SDK v1	Rough mapping in SDK v2
<code>azureml.pipeline.steps.parallelrunconfig</code> <code>azureml.pipeline.steps.parallelrunstep</code>	<code>azure.ai.ml.parallel</code>
<code>OutputDatasetConfig</code>	<code>Output</code>
<code>dataset as _mount</code>	<code>Input</code>

Parallel job configurations and settings mapping

SDK v1	SDK v2	Description
<code>ParallelRunConfig.environment</code>	<code>parallel_run_function.task.environment</code>	Environment that training job will run in.
<code>ParallelRunConfig.entry_script</code>	<code>parallel_run_function.task.entry_script</code>	User script that will be run in parallel on multiple nodes.
<code>ParallelRunConfig.error_threshold</code>	<code>parallel_run_function.error_threshold</code>	The number of failed mini batches that could be ignored in this parallel job. If the count of failed mini-batch is higher than this threshold, the parallel job will be marked as failed. "-1" is the default number, which means to ignore all failed mini-batch during parallel job.
<code>ParallelRunConfig.output_action</code>	<code>parallel_run_function.append_row_to</code>	Aggregate all returns from each run of mini-batch and output it into this file. May reference to one of the outputs of parallel job by using the expression <code> \${outputs.<output_name>} </code>
<code>ParallelRunConfig.node_count</code>	<code>parallel_run_function.instance_count</code>	Optional number of instances or nodes used by the compute target. Defaults to 1.
<code>ParallelRunConfig.process_count_per_node</code>	<code>parallel_run_function.max_concurrency_per_instance</code>	The max parallelism that each compute instance has.

SDK v1	SDK v2	Description
ParallelRunConfig.mini_batch_size	parallel_run_function.mini_batch_size	Define the size of each mini-batch to split the input.
		If the input_data is a folder or set of files, this number defines the file count for each mini-batch. For example, 10, 100.
		If the input_data is tabular data from <code>mtable</code> , this number defines the proximate physical size for each mini-batch. The default unit is Byte and the value could accept string like 100 kb, 100 mb.
ParallelRunConfig.source_directory	parallel_run_function.task.code	A local or remote path pointing at source code.
ParallelRunConfig.description	parallel_run_function.description	A friendly description of the parallel
ParallelRunConfig.logging_level	parallel_run_function.logging_level	A string of the logging level name, which is defined in 'logging'. Possible values are 'WARNING', 'INFO', and 'DEBUG'. (optional, default value is 'INFO'.) This value could be set through PipelineParameter.
ParallelRunConfig.run_invocation_timeout	parallel_run_function.retry_settings.timeout	The timeout in seconds for executing custom run() function. If the execution time is higher than this threshold, the mini-batch will be aborted, and marked as a failed mini-batch to trigger retry.
ParallelRunConfig.run_max_try	parallel_run_function.retry_settings.max_retries	The number of retries when mini-batch is failed or timeout. If all retries are failed, the mini-batch will be marked as failed to be counted by <code>mini_batch_error_threshold</code> calculation.
ParallelRunConfig.append_row_file_name	parallel_run_function.append_row_to	Combined with <code>append_row_to</code> setting.

SDK v1	SDK v2	Description
ParallelRunConfig.allowed_failed_count	parallel_run_function.mini_batch_error_threshold	The number of failed mini batches that could be ignored in this parallel job. If the count of failed mini-batch is higher than this threshold, the parallel job will be marked as failed. "-1" is the default number, which means to ignore all failed mini-batch during parallel job.
ParallelRunConfig.allowed_failed_percent	parallel_run_function.task.program_arguments set --allowed_failed_percent	Similar to "allowed_failed_count" but this setting uses the percent of failed mini-batches instead of the mini-batch failure count. The range of this setting is [0, 100]. "100" is the default number, which means to ignore all failed mini-batch during parallel job.
ParallelRunConfig.partition_keys	<i>Under development.</i>	
ParallelRunConfig.environment_variables	parallel_run_function.environment_variables	A dictionary of environment variables names and values. These environment variables are set on the process where user script is being executed.
ParallelRunStep.name	parallel_run_function.name	Name of the parallel job or component created.
ParallelRunStep.inputs	parallel_run_function.inputs	A dict of inputs used by this parallel.
--	parallel_run_function.input_data	Declare the data to be split and processed with parallel
ParallelRunStep.output	parallel_run_function.outputs	The outputs of this parallel job.
ParallelRunStep.side_inputs	parallel_run_function.inputs	Defined together with inputs.
ParallelRunStep.arguments	parallel_run_function.task.program_arguments	The arguments of the parallel task.

SDK v1	SDK v2	Description
ParallelRunStep.allow_reuse	parallel_run_function.is_deterministic	Specify whether the parallel will return same output given same input.

Next steps

For more information, see the documentation here:

- [Parallel run step SDK v1 examples ↗](#)
 - [Parallel job SDK v2 examples ↗](#)
-

Additional resources

Documentation

[Upgrade hyperparameter tuning to SDK v2 - Azure Machine Learning](#)

Upgrade hyperparameter tuning from v1 to v2 of Azure Machine Learning SDK

[Troubleshooting local model deployment - Azure Machine Learning](#)

Try a local model deployment as a first step in troubleshooting model deployment errors.

[mltable package - Azure Machine Learning Python](#)

[Upgrade script run to SDK v2 - Azure Machine Learning](#)

Upgrade how to run a script from SDK v1 to SDK v2

[azureml.pipeline.core.PipelineRun class - Azure Machine Learning Python](#)

Represents a run of a Pipeline. This class can be used to manage, check status, and retrieve run details once a pipeline run is submitted. Use get_steps to retrieve the StepRun objects which are created by the pipeline run. Other uses include retrieving the Graph object associated with the pipeline run, fetching the status of the pipeline run, and waiting for run...

[azureml.pipeline.core.PipelineParameter class - Azure Machine Learning Python](#)

Defines a parameter in a pipeline execution. Use PipelineParameters to construct versatile Pipelines which can be resubmitted later with varying parameter values.

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[Show 5 more](#)

Upgrade pipelines to SDK v2

Article • 02/24/2023 • 4 minutes to read

In SDK v2, "pipelines" are consolidated into jobs.

A job has a type. Most jobs are command jobs that run a `command`, like `python main.py`.

What runs in a job is agnostic to any programming language, so you can run `bash`

scripts, invoke `python` interpreters, run a bunch of `curl` commands, or anything else.

A `pipeline` is another type of job, which defines child jobs that may have input/output relationships, forming a directed acyclic graph (DAG).

To upgrade, you'll need to change your code for defining and submitting the pipelines to SDK v2. What you run *within* the child job doesn't need to be upgraded to SDK v2. However, it's recommended to remove any code specific to Azure Machine Learning from your model training scripts. This separation allows for an easier transition between local and cloud and is considered best practice for mature MLOps. In practice, this means removing `azureml.*` lines of code. Model logging and tracking code should be replaced with MLflow. For more information, see [how to use MLflow in v2](#).

This article gives a comparison of scenario(s) in SDK v1 and SDK v2. In the following examples, we'll build three steps (train, score and evaluate) into a dummy pipeline job. This demonstrates how to build pipeline jobs using SDK v1 and SDK v2, and how to consume data and transfer data between steps.

Run a pipeline

- SDK v1

```
Python

# import required libraries
import os
import azureml.core
from azureml.core import (
    Workspace,
    Dataset,
    Datastore,
    ComputeTarget,
    Experiment,
    ScriptRunConfig,
)
from azureml.pipeline.steps import PythonScriptStep
from azureml.pipeline.core import Pipeline
```

```

# check core SDK version number
print("Azure Machine Learning SDK Version: ", azureml.core.VERSION)

# load workspace
workspace = Workspace.from_config()
print(
    "Workspace name: " + workspace.name,
    "Azure region: " + workspace.location,
    "Subscription id: " + workspace.subscription_id,
    "Resource group: " + workspace.resource_group,
    sep="\n",
)

# create an ML experiment
experiment = Experiment(workspace=workspace,
name="train_score_eval_pipeline")

# create a directory
script_folder = "./src"

# create compute
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

# Choose a name for your CPU cluster
amlcompute_cluster_name = "cpu-cluster"

# Verify that cluster does not exist already
try:
    aml_compute = ComputeTarget(workspace=workspace,
name=amlcompute_cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config =
AmlCompute.provisioning_configuration(vm_size='STANDARD_DS12_V2',
                                         max_nodes=4)
    aml_compute = ComputeTarget.create(ws, amlcompute_cluster_name,
compute_config)

aml_compute.wait_for_completion(show_output=True)

# define data set
data_urls =
["wasbs://demo@dprepdata.blob.core.windows.net/Titanic.csv"]
input_ds = Dataset.File.from_files(data_urls)

# define steps in pipeline
from azureml.data import OutputFileDatasetConfig
model_output = OutputFileDatasetConfig('model_output')
train_step = PythonScriptStep(
    name="train step",
    script_name="train.py",
    arguments=['--training_data',
    input_ds.as_named_input('training_data').as_mount() , '--max_epochs' , 5,

```

```

'--learning_rate', 0.1,'--model_output', model_output],
    source_directory=script_folder,
    compute_target=aml_compute,
    allow_reuse=True,
)

score_output = OutputFileDatasetConfig('score_output')
score_step = PythonScriptStep(
    name="score step",
    script_name="score.py",
    arguments=['--model_input',model_output.as_input('model_input'), '--test_data', input_ds.as_named_input('test_data').as_mount(), '--score_output', score_output],
    source_directory=script_folder,
    compute_target=aml_compute,
    allow_reuse=True,
)

eval_output = OutputFileDatasetConfig('eval_output')
eval_step = PythonScriptStep(
    name="eval step",
    script_name="eval.py",
    arguments=[ '--scoring_result',score_output.as_input('scoring_result'), '--eval_output', eval_output],
    source_directory=script_folder,
    compute_target=aml_compute,
    allow_reuse=True,
)

# built pipeline
from azureml.pipeline.core import Pipeline

pipeline_steps = [train_step, score_step, eval_step]

pipeline = Pipeline(workspace = workspace, steps=pipeline_steps)
print("Pipeline is built.")

pipeline_run = experiment.submit(pipeline, regenerate_outputs=False)

print("Pipeline submitted for execution.")

```

- SDK v2. [Full sample link ↗](#)

Python

```

# import required libraries
from azure.identity import DefaultAzureCredential,
InteractiveBrowserCredential

from azure.ai.ml import MLClient, Input
from azure.ai.ml.dsl import pipeline

```

```
try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case
    DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()

# Get a handle to workspace
ml_client = MLClient.from_config(credential=credential)

# Retrieve an already attached Azure Machine Learning Compute.
cluster_name = "cpu-cluster"
print(ml_client.compute.get(cluster_name))

# Import components that are defined with Python function
with open("src/components.py") as fin:
    print(fin.read())

# You need to install mldesigner package to use command_component
decorator.
# Option 1: install directly
# !pip install mldesigner

# Option 2: install as an extra dependency of azure-ai-ml
# !pip install azure-ai-ml[designer]

# import the components as functions
from src.components import train_model, score_data, eval_model

cluster_name = "cpu-cluster"
# define a pipeline with component
@pipeline(default_compute=cluster_name)
def pipeline_with_python_function_components(input_data, test_data,
learning_rate):
    """E2E dummy train-score-eval pipeline with components defined via
    Python function components"""

    # Call component obj as function: apply given inputs & parameters
    # to create a node in pipeline
    train_with_sample_data = train_model(
        training_data=input_data, max_epochs=5,
        learning_rate=learning_rate
    )

    score_with_sample_data = score_data(
        model_input=train_with_sample_data.outputs.model_output,
        test_data=test_data
    )

    eval_with_sample_data = eval_model(
        scoring_result=score_with_sample_data.outputs.score_output
    )
```

```

    # Return: pipeline outputs
    return {
        "eval_output": eval_with_sample_data.outputs.eval_output,
        "model_output": train_with_sample_data.outputs.model_output,
    }

pipeline_job = pipeline_with_python_function_components(
    input_data=Input(
        path="wasbs://demo@dprepdata.blob.core.windows.net/Titanic.csv",
        type="uri_file"
    ),
    test_data=Input(
        path="wasbs://demo@dprepdata.blob.core.windows.net/Titanic.csv",
        type="uri_file"
    ),
    learning_rate=0.1,
)

# submit job to workspace
pipeline_job = ml_client.jobs.create_or_update(
    pipeline_job, experiment_name="train_score_eval_pipeline"
)

```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
azureml.pipeline.core.Pipeline	azure.ai.ml.dsl.pipeline
OutputDatasetConfig	Output
dataset as_mount	Input

Step and job/component type mapping

step in SDK v1	job type in SDK v2	component type in SDK v2
adla_step	None	None
automl_step	automl job	automl component
azurebatch_step	None	None

step in SDK v1	job type in SDK v2	component type in SDK v2
command_step	command job	command component
data_transfer_step	coming soon	coming soon
databricks_step	coming soon	coming soon
estimator_step	command job	command component
hyper_drive_step	sweep job	sweep component
kusto_step	None	None
module_step	None	command component
mpi_step	command job	command component
parallel_run_step	Parallel job	Parallel component
python_script_step	command job	command component
r_script_step	command job	command component
synapse_spark_step	coming soon	coming soon

Related documents

For more information, see the documentation here:

- [steps in SDK v1](#)
- [Create and run machine learning pipelines using components with the Azure Machine Learning SDK v2](#)
- [Build a simple ML pipeline for image classification \(SDK v1\)](#) ↗
- [OutputDatasetConfig](#)
- [mldesigner](#) ↗

Additional resources

Documentation

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.Data class](#)

Data for training and scoring.

[azure.ai.ml.operations.DatastoreOperations class](#)

Represents a client for performing operations on Datastores. You should not instantiate this class directly. Instead, you should create MLClient and use this client via the property MLClient.datastores

[Upgrade data management to SDK v2 - Azure Machine Learning](#)

Upgrade data management from v1 to v2 of Azure Machine Learning SDK

[Upgrade script run to SDK v2 - Azure Machine Learning](#)

Upgrade how to run a script from SDK v1 to SDK v2

[azure.ai.ml.entities.ManagedOnlineDeployment class](#)

Managed Online endpoint deployment entity.

[azure.ai.ml.entities.Job class](#)

Base class for job, can't be instantiated directly.

[Upgrade datastore management to SDK v2 - Azure Machine Learning](#)

Upgrade datastore management from v1 to v2 of Azure Machine Learning SDK

[Show 5 more](#)

Upgrade deployment endpoints to SDK v2

Article • 01/26/2023 • 2 minutes to read

We newly introduced [online endpoints](#) and batch endpoints as v2 concepts. There are several deployment funnels such as managed online endpoints, [kubernetes online endpoints](#) (including Azure Kubernetes Services and Arc-enabled Kubernetes) in v2, and Azure Container Instances (ACI) and Kubernetes Services (AKS) webservices in v1. In this article, we'll focus on the comparison of deploying to ACI webservices (v1) and managed online endpoints (v2).

Examples in this article show how to:

- Deploy your model to Azure
- Score using the endpoint
- Delete the webservice/endpoint

Create inference resources

- SDK v1

1. Configure a model, an environment, and a scoring script:

Python

```
# configure a model. example for registering a model
from azureml.core.model import Model
model = Model.register(ws, model_name="bidaf_onnx",
model_path="./model.onnx")

# configure an environment
from azureml.core import Environment
env = Environment(name='myenv')
python_packages = ['nltk', 'numpy', 'onnxruntime']
for package in python_packages:
    env.python.conda_dependencies.add_pip_package(package)

# configure an inference configuration with a scoring script
from azureml.core.model import InferenceConfig
inference_config = InferenceConfig(
    environment=env,
    source_directory="./source_dir",
    entry_script="./score.py",
)
```

2. Configure and deploy an ACI webservice:

```
Python

from azureml.core.webservice import AciWebservice

# defince compute resources for ACI
deployment_config = AciWebservice.deploy_configuration(
    cpu_cores=0.5, memory_gb=1, auth_enabled=True
)

# define an ACI webservice
service = Model.deploy(
    ws,
    "myservice",
    [model],
    inference_config,
    deployment_config,
    overwrite=True,
)

# create the service
service.wait_for_deployment(show_output=True)
```

For more information on registering models, see [Register a model from a local file](#).

- SDK v2

1. Configure a model, an environment, and a scoring script:

```
Python

from azure.ai.ml.entities import Model
# configure a model
model = Model(path="../model-1/model/sklearn_regression_model.pkl")

# configure an environment
from azure.ai.ml.entities import Environment
env = Environment(
    conda_file="../model-1/environment/conda.yml",
    image="mcr.microsoft.com/azureml/openmpi3.1.2-
ubuntu18.04:20210727.v1",
)

# configure an inference configuration with a scoring script
from azure.ai.ml.entities import CodeConfiguration
code_config = CodeConfiguration(
    code="../model-1/onlinescoring", scoring_script="score.py"
)
```

2. Configure and create an **online endpoint**:

```
Python

import datetime
from azure.ai.ml.entities import ManagedOnlineEndpoint

# create a unique endpoint name with current datetime to avoid
# conflicts
online_endpoint_name = "endpoint-" +
datetime.datetime.now().strftime("%m%d%H%M%f")

# define an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is a sample online endpoint",
    auth_mode="key",
    tags={"foo": "bar"},
)

# create the endpoint:
ml_client.begin_create_or_update(endpoint)
```

3. Configure and create an **online deployment**:

```
Python

from azure.ai.ml.entities import ManagedOnlineDeployment

# define a deployment
blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    environment=env,
    code_configuration=code_config,
    instance_type="Standard_F2s_v2",
    instance_count=1,
)

# create the deployment:
ml_client.begin_create_or_update(blue_deployment)

# blue deployment takes 100 traffic
endpoint.traffic = {"blue": 100}
ml_client.begin_create_or_update(endpoint)
```

For more information on concepts for endpoints and deployments, see [What are online endpoints?](#)

Submit a request

- SDK v1

```
Python
```

```
import json
data = {
    "query": "What color is the fox",
    "context": "The quick brown fox jumped over the lazy dog.",
}
data = json.dumps(data)
predictions = service.run(input_data=data)
print(predictions)
```

- SDK v2

```
Python
```

```
# test the endpoint (the request will route to blue deployment as set
# above)
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    request_file="../model-1/sample-request.json",
)

# test the specific (blue) deployment
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    deployment_name="blue",
    request_file="../model-1/sample-request.json",
)
```

Delete resources

- SDK v1

```
Python
```

```
service.delete()
```

- SDK v2

```
Python
```

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Mapping of key functionality in SDK v1 and SDK v2

Functionality in SDK v1	Rough mapping in SDK v2
<code>azureml.core.model.Model</code> class	<code>azure.ai.ml.entities.Model</code> class
<code>azureml.core.Environment</code> class	<code>azure.ai.ml.entities.Environment</code> class
<code>azureml.core.model.InferenceConfig</code> class	<code>azure.ai.ml.entities.CodeConfiguration</code> class
<code>azureml.core.webservice.AciWebservice</code> class	<code>azure.ai.ml.entities.OnlineDeployment</code> class (and <code>azure.ai.ml.entities.ManagedOnlineEndpoint</code> class)
<code>Model.deploy</code> or <code>Webservice.deploy</code>	<code>ml_client.begin_create_or_update(online_deployment)</code>
<code>Webservice.run</code>	<code>ml_client.online_endpoints.invoke</code>
<code>Webservice.delete</code>	<code>ml_client.online_endpoints.delete</code>

Related documents

For more information, see

v2 docs:

- [What are endpoints?](#)
- [Deploy machine learning models to managed online endpoint using Python SDK v2](#)

v1 docs:

- [MLOps: ML model management v1](#)
- [Deploy machine learning models](#)

Additional resources

Documentation

[azure.ai.ml.entities.Command class](#)

Base class for command node, used for command component version consumption. You should not instantiate this class directly. Instead, you should create from builder function: `command`.

[azure.ai.ml.entities.Datastore class](#)

Datastore of an Azure ML workspace, abstract class.

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.Model class](#)

Model for training and scoring.

[azure.ai.ml.Output class](#)

Define an output of a Component or Job.

[azure.ai.ml.entities.Job class](#)

Base class for job, can't be instantiated directly.

[azure.ai.ml.operations.ModelOperations class](#)

ModelOperations. You should not instantiate this class directly. Instead, you should create an MLClient instance that instantiates it for you and attaches it as an attribute.

[Upgrade script run to SDK v2 - Azure Machine Learning](#)

Upgrade how to run a script from SDK v1 to SDK v2

[Show 5 more](#)

Upgrade steps for Azure Container Instances web services to managed online endpoints

Article • 01/26/2023 • 4 minutes to read

[Managed online endpoints](#) help to deploy your ML models in a turnkey manner.

Managed online endpoints work with powerful CPU and GPU machines in Azure in a scalable, fully managed way. Managed online endpoints take care of serving, scaling, securing, and monitoring your models, freeing you from the overhead of setting up and managing the underlying infrastructure. Details can be found on [Deploy and score a machine learning model by using an online endpoint](#).

You can deploy directly to the new compute target with your previous models and environments, or use the [scripts](#) provided by us to export the current services and then deploy to the new compute without affecting your existing services. If you regularly create and delete Azure Container Instances (ACI) web services, we strongly recommend the deploying directly and not using the scripts.

Important

The scoring URL will be changed after upgrade. For example, the scoring url for ACI web service is like `http://aaaaaa-bbbb-1111.westus.azurecontainer.io/score`.

The scoring URI for a managed online endpoint is like `https://endpoint-name.westus.inference.ml.azure.com/score`.

Supported scenarios and differences

Auth mode

No auth isn't supported for managed online endpoint. If you use the upgrade scripts, it will convert it to key auth. For key auth, the original keys will be used. Token-based auth is also supported.

TLS

For ACI service secured with HTTPS, you don't need to provide your own certificates anymore, all the managed online endpoints are protected by TLS.

Custom DNS name **isn't** supported.

Resource requirements

[ContainerResourceRequirements](#) isn't supported, you can choose the proper [SKU](#) for your inferencing. The upgrade tool will map the CPU/Memory requirement to corresponding SKU. If you choose to redeploy manually through CLI/SDK V2, we also suggest the corresponding SKU for your new deployment.

CPU request	Memory request in GB	Suggested SKU
(0, 1]	(0, 1.2]	DS1 V2
(1, 2]	(1.2, 1.7]	F2s V2
(1, 2]	(1.7, 4.7]	DS2 V2
(1, 2]	(4.7, 13.7]	E2s V3
(2, 4]	(0, 5.7]	F4s V2
(2, 4]	(5.7, 11.7]	DS3 V2
(2, 4]	(11.7, 16]	E4s V3

"(" means greater than and ")" means less than or equal to. For example, "(0, 1]" means "greater than 0 and less than or equal to 1".

Important

When upgrading from ACI, there will be some changes in how you'll be charged. See [our blog](#) for a rough cost comparison to help you choose the right VM SKUs for your workload.

Network isolation

For private workspace and VNet scenarios, see [Use network isolation with managed online endpoints](#).

Important

As there are many settings for your workspace and VNet, we strongly suggest that redeploy through the Azure CLI extension v2 for machine learning instead of the script tool.

Not supported

- [EncryptionProperties](#) for ACI container isn't supported.
- ACI web services deployed through `deploy_from_model` and `deploy_from_image` isn't supported by the upgrade tool. Redeploy manually through CLI/SDK V2.

Upgrade steps

With our [CLI or SDK preview](#)

Redeploy manually with your model files and environment definition. You can find our examples on [azureml-examples](#). Specifically, this is the [SDK example for managed online endpoint](#).

With our [upgrade tool](#)

This tool will automatically create new managed online endpoint based on your existing web services. Your original services won't be affected. You can safely route the traffic to the new endpoint and then delete the old one.

Note

The upgrade script is a sample script and is provided without a service level agreement (SLA).

Use the following steps to run the scripts:

Tip

The new endpoint created by the scripts will be created under the same workspace.

1. Use a bash shell to run the scripts. For example, a terminal session on Linux or the Windows Subsystem for Linux (WSL).
2. Install [Python SDK V1](#) to run the Python script.
3. Install [Azure CLI](#).

4. Clone [the repository](#) to your local env. For example, `git clone`

```
https://github.com/Azure/azureml-examples.
```

5. Edit the following values in the `migrate-service.sh` file. Replace the values with ones that apply to your configuration.

- `<SUBSCRIPTION_ID>` - The subscription ID of your Azure subscription that contains your workspace.
- `<RESOURCEGROUP_NAME>` - The resource group that contains your workspace.
- `<WORKSPACE_NAME>` - The workspace name.
- `<SERVICE_NAME>` - The name of your existing ACI service.
- `<LOCAL_PATH>` - A local path where resources and templates used by the script are downloaded.
- `<NEW_ENDPOINT_NAME>` - The name of the new endpoint that will be created. We recommend that the new endpoint name is different from the previous service name. Otherwise, the original service won't be displayed if you check your endpoints on the portal.
- `<NEW_DEPLOYMENT_NAME>` - The name of the deployment to the new endpoint.

6. Run the bash script. For example, `./migrate-service.sh`. It will take about 5-10 minutes to finish the new deployment.

💡 Tip

If you receive an error that the script is not executable, or an editor opens when you try to run the script, use the following command to mark the script as executable:

Bash

```
chmod +x migrate-service.sh
```

7. After the deployment is completes successfully, you can verify the endpoint with the [az ml online-endpoint invoke](#) command.

Contact us

If you have any questions or feedback on the upgrade script, contact us at moeonboard@microsoft.com.

Next steps

- [What are Azure Machine Learning endpoints?](#)
 - [Deploy and score a model with an online endpoint](#)
-

Additional resources

Documentation

[azure.ai.ml.entities.Model class](#)

Model for training and scoring.

[azure.ai.ml.entities.Command class](#)

Base class for command node, used for command component version consumption. You should not instantiate this class directly. Instead, you should create from builder function: command.

[azureml.core.runconfig.DockerConfiguration class - Azure Machine Learning Python](#)

Represents Docker runtime configuration for jobs.

[azure.ai.ml.operations.DataOperations class](#)

[Upgrade local runs to SDK v2 - Azure Machine Learning](#)

Upgrade local runs from v1 to v2 of Azure Machine Learning SDK

[azure.ai.ml.entities.Job class](#)

Base class for job, can't be instantiated directly.

[azure.ai.ml.operations.ModelOperations class](#)

ModelOperations. You should not instantiate this class directly. Instead, you should create an MLClient instance that instantiates it for you and attaches it as an attribute.

[azure.ai.ml.Output class](#)

Define an output of a Component or Job.

[Show 5 more](#)

Training

Learning paths and modules

[Deploy an Azure Machine Learning model to a managed endpoint with CLI \(v2\) - Training](#)

Deploy an Azure Machine Learning model to a managed endpoint with CLI (v2)

Quickstart: Create workspace resources you need to get started with Azure Machine Learning

Article • 02/21/2023 • 5 minutes to read

In this quickstart, you'll create a workspace and then add compute resources to the workspace. You'll then have everything you need to get started with Azure Machine Learning.

The workspace is the top-level resource for your machine learning activities, providing a centralized place to view and manage the artifacts you create when you use Azure Machine Learning. The compute resources provide a pre-configured cloud-based environment you can use to train, deploy, automate, manage, and track machine learning models.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).

Create the workspace

If you already have a workspace, skip this section and continue to [Create a compute instance](#).

If you don't yet have a workspace, create one now:

1. Sign in to [Azure Machine Learning studio](#)
2. Select **Create workspace**
3. Provide the following information to configure your new workspace:

Field	Description
Workspace name	Enter a unique name that identifies your workspace. Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others. The workspace name is case-insensitive.
Subscription	Select the Azure subscription that you want to use.

Field	Description
Resource group	Use an existing resource group in your subscription or enter a name to create a new resource group. A resource group holds related resources for an Azure solution. You need <i>contributor</i> or <i>owner</i> role to use an existing resource group. For more information about access, see Manage access to an Azure Machine Learning workspace .
Region	Select the Azure region closest to your users and the data resources to create your workspace.

4. Select **Create** to create the workspace

ⓘ Note

This creates a workspace along with all required resources. If you would like to reuse resources, such as Storage Account, Azure Container Registry, Azure KeyVault, or Application Insights, use the [Azure portal](#) instead.

Create compute instance

You could install Azure Machine Learning on your own computer. But in this quickstart, you'll create an online compute resource that has a development environment already installed and ready to go. You'll use this online machine, a *compute instance*, for your development environment to write and run code in Python scripts and Jupyter notebooks.

Create a *compute instance* to use this development environment for the rest of the tutorials and quickstarts.

1. If you didn't just create a workspace in the previous section, sign in to [Azure Machine Learning studio](#) now, and select your workspace.
2. On the left side, select **Compute**.

3. Select **+New** to create a new compute instance.
4. Supply a name, Keep all the defaults on the first page.
5. Select **Create**.

In about two minutes, you'll see the **State** of the compute instance change from *Creating* to *Running*. It's now ready to go.

Create compute clusters

Next you'll create a compute cluster. You'll submit code to this cluster to distribute your training or batch inference processes across a cluster of CPU or GPU compute nodes in the cloud.

Create a compute cluster that will autoscale between zero and four nodes:

1. Still in the **Compute** section, in the top tab, select **Compute clusters**.
2. Select **+New** to create a new compute cluster.
3. Keep all the defaults on the first page, select **Next**. If you don't see any available compute, you'll need to request a quota increase. Learn more about [managing and increasing quotas](#).
4. Name the cluster **cpu-cluster**. If this name already exists, add your initials to the name to make it unique.

5. Leave the **Minimum number of nodes** at 0.
6. Change the **Maximum number of nodes** to 4 if possible. Depending on your settings, you may have a smaller limit.
7. Change the **Idle seconds before scale down** to 2400.
8. Leave the rest of the defaults, and select **Create**.

In less than a minute, the **State** of the cluster will change from *Creating* to *Succeeded*. The list shows the provisioned compute cluster, along with the number of idle nodes, busy nodes, and unprovisioned nodes. Since you haven't used the cluster yet, all the nodes are currently unprovisioned.

 **Note**

When the cluster is created, it will have 0 nodes provisioned. The cluster *does not* incur costs until you submit a job. This cluster will scale down when it has been idle for 2,400 seconds (40 minutes). This will give you time to use it in a few tutorials if you wish without waiting for it to scale back up.

Quick tour of the studio

The studio is your web portal for Azure Machine Learning. This portal combines no-code and code-first experiences for an inclusive data science platform.

Review the parts of the studio on the left-hand navigation bar:

- The **Authoring** section of the studio contains multiple ways to get started in creating machine learning models. You can:
 - **Notebooks** section allows you to create Jupyter Notebooks, copy sample notebooks, and run notebooks and Python scripts.
 - **Automated ML** steps you through creating a machine learning model without writing code.
 - **Designer** gives you a drag-and-drop way to build models using prebuilt components.
- The **Assets** section of the studio helps you keep track of the assets you create as you run your jobs. If you have a new workspace, there's nothing in any of these sections yet.
- You already used the **Manage** section of the studio to create your compute resources. This section also lets you create and manage data and external services you link to your workspace.

Workspace diagnostics

You can run diagnostics on your workspace from Azure Machine Learning studio or the Python SDK. After diagnostics run, a list of any detected problems is returned. This list includes links to possible solutions. For more information, see [How to use workspace diagnostics](#).

Clean up resources

If you plan to continue now to the next tutorial, skip to [Next steps](#).

Stop compute instance

If you're not going to use it now, stop the compute instance:

1. In the studio, on the left, select **Compute**.
2. In the top tabs, select **Compute instances**
3. Select the compute instance in the list.
4. On the top toolbar, select **Stop**.

Delete all resources

Important

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Favorites' section with items like All resources, Resource groups, App Services, SQL databases, etc. The main content area is titled 'Resource groups > my-rg'. The 'Overview' tab is selected. At the top right, there are buttons for Add, Edit columns, Delete resource group (which is highlighted with a red box), Refresh, and Move. Below that, it shows the Subscription (Visual Studio Ultimate with MSDN), Subscription ID (xxxxxx-xxx-xxxx-xxxx), and Tags (none). A table lists 14 items: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights). There are filters for 'Filter by name...', 'All types', and 'All locations'.

4. Enter the resource group name. Then select **Delete**.

Next steps

You now have an Azure Machine Learning workspace that contains:

- A compute instance to use for your development environment.
- A compute cluster to use for submitting training runs.

Use these resources to learn more about Azure Machine Learning and train a model with Python scripts.

[Quickstart: Run Jupyter notebook in Azure Machine Learning studio](#)

Additional resources

[Documentation](#)

[Tutorial: Azure ML in a day - Azure Machine Learning](#)

Use Azure Machine Learning to train and deploy a model in a cloud-based Python Jupyter Notebook.

[Machine learning inference during deployment - Cloud Adoption Framework](#)

Understand how your AI model makes predictions while it's being deployed in production.

[Tutorial: AutoML- train no-code classification models - Azure Machine Learning](#)

Train a classification model without writing a single line of code using Azure Machine Learning automated ML in the studio UI.

[Train deep learning PyTorch models \(SDK v2\) - Azure Machine Learning](#)

Learn how to run your PyTorch training scripts at enterprise scale using Azure Machine Learning SDK (v2).

[Quickstart: Run notebooks - Azure Machine Learning](#)

Learn to run Jupyter notebooks in studio, and find sample notebooks to learn more about Azure Machine Learning.

[Deploy machine learning models - Azure Machine Learning](#)

Learn how and where to deploy machine learning models. Deploy to Azure Container Instances, Azure Kubernetes Service, and FPGA.

[Build & train models - Azure Machine Learning](#)

Learn how to train models with Azure Machine Learning. Explore the different training methods and choose the right one for your project.

[Detect data drift on datasets \(preview\) - Azure Machine Learning](#)

Learn how to set up data drift detection in Azure Learning. Create datasets monitors (preview), monitor for data drift, and set up alerts.

[Show 5 more](#)

[Training](#)

Learning paths and modules

[Create workspace resources for getting started with Azure Machine Learning - Training](#)

Create workspace resources for getting started with Azure Machine Learning.

Quickstart: Interactive Data Wrangling with Apache Spark in Azure Machine Learning (preview)

Article • 02/24/2023 • 3 minutes to read

Important

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads.

Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

To handle interactive Azure Machine Learning notebook data wrangling, Azure Machine Learning integration, with Azure Synapse Analytics (preview), provides easy access to the Apache Spark framework. This access allows for Azure Machine Learning Notebook interactive data wrangling.

In this quickstart guide, you'll learn how to perform interactive data wrangling using Azure Machine Learning Managed (Automatic) Synapse Spark compute, Azure Data Lake Storage (ADLS) Gen 2 storage account, and user identity passthrough.

Prerequisites

- An Azure subscription; if you don't have an Azure subscription, [create a free account](#) before you begin.
- An Azure Machine Learning workspace. See [Create workspace resources](#).
- An Azure Data Lake Storage (ADLS) Gen 2 storage account. See [Create an Azure Data Lake Storage \(ADLS\) Gen 2 storage account](#).
- To enable this feature:
 1. Navigate to the Azure Machine Learning studio UI
 2. In the icon section at the top right of the screen, select **Manage preview features** (megaphone icon)
 3. In the **Managed preview feature** panel, toggle the **Run notebooks and jobs on managed Spark** feature to **on**

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a navigation sidebar with options like 'Contoso', 'New', 'Home', 'Author', 'Notebooks', 'Automated ML', 'Designer', 'Assets', 'Data', 'Jobs', 'Components', 'Pipelines', 'Environments', 'Models', 'Endpoints', 'Compute', 'Linked Services', and 'Data Labeling'. The main area displays a preview of a workspace named '<AML_WORKSPACE_NAME>'. It includes sections for 'Create new' (Notebooks), 'Recent resources' (Jobs, Compute, Models, Data), and a message stating 'No jobs to display'. On the right, there's a 'Manage preview features' panel with several toggle switches. One feature, 'Run notebooks and jobs on managed Spark', is highlighted with a red box and has its status set to 'Enabled'.

Add role assignments in Azure storage accounts

We must ensure that the input and output data paths are accessible, before we start interactive data wrangling. To enable read and write access, assign **Contributor** and **Storage Blob Data Contributor** roles to the user identity of the logged-in user.

To assign appropriate roles to the user identity:

1. Open the [Microsoft Azure portal](#).
2. Search and select the **Storage accounts** service.

The screenshot shows the Microsoft Azure portal homepage. A search bar at the top contains the text 'storage accounts'. Below the search bar, under the heading 'Azure services', there is a 'Create a resource' button and a 'Services' section. In the 'Services' section, the 'Storage accounts' item is highlighted with a red box. Other items listed include 'CloudTest Accounts', 'Genomics accounts', and 'Integration accounts'. The top navigation bar shows 'user@contoso.com' and 'CONTOSO CONTOSO.COM MICRO...'.

3. On the **Storage accounts** page, select the Azure Data Lake Storage (ADLS) Gen 2 storage account from the list. A page showing the storage account **Overview** will open.

The screenshot shows the 'Storage accounts' overview page in the Microsoft Azure portal. At the top, there are filter and search options. The main area displays a table with one record. The columns are 'Name' (with a red box around it), 'Type' (Storage account), 'Kind' (StorageV2, also with a red box), 'Resource group' (<RESOURCE_GROUP>), 'Location' (East US), and 'Subscription' (<SUBSCRIPTION_NAME>). The table has sorting arrows for each column. A large search icon is visible on the right side of the page.

4. Select Access Control (IAM) from the left panel

5. Select Add role assignment

The screenshot shows the Microsoft Azure Storage Account Access Control (IAM) interface. On the left sidebar, the 'Access Control (IAM)' option is selected and highlighted with a red box. In the main content area, there are three cards: 'Grant access to this resource', 'View access to this resource', and 'View deny assignments'. Each card has a 'View' and 'Learn more' button. Below these cards is a large red box highlighting the 'Add role assignments' button.

6. Find and select role Storage Blob Data Contributor

7. Select Next

The screenshot shows the 'Add role assignment' page. The 'Role' tab is selected, showing a list of roles. A search bar at the top is set to 'Storage Blob'. The 'Storage Blob Data Contributor' role is highlighted with a red box. At the bottom of the page, there are 'Review + assign', 'Previous', and 'Next' buttons. The 'Next' button is highlighted with a red box.

8. Select User, group, or service principal.

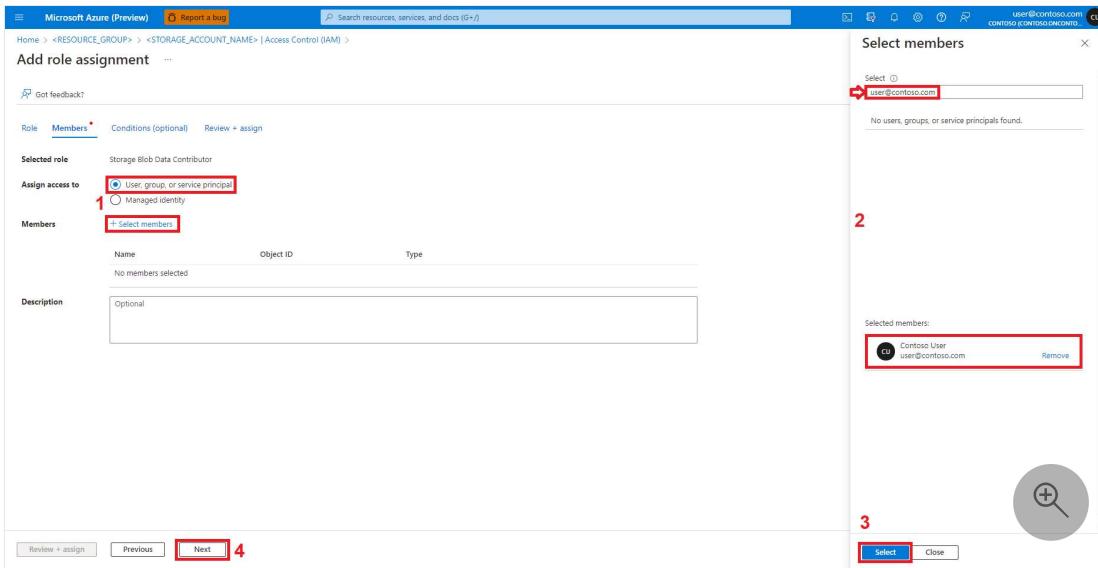
9. Select + Select members.

10. Search for the user identity below Select

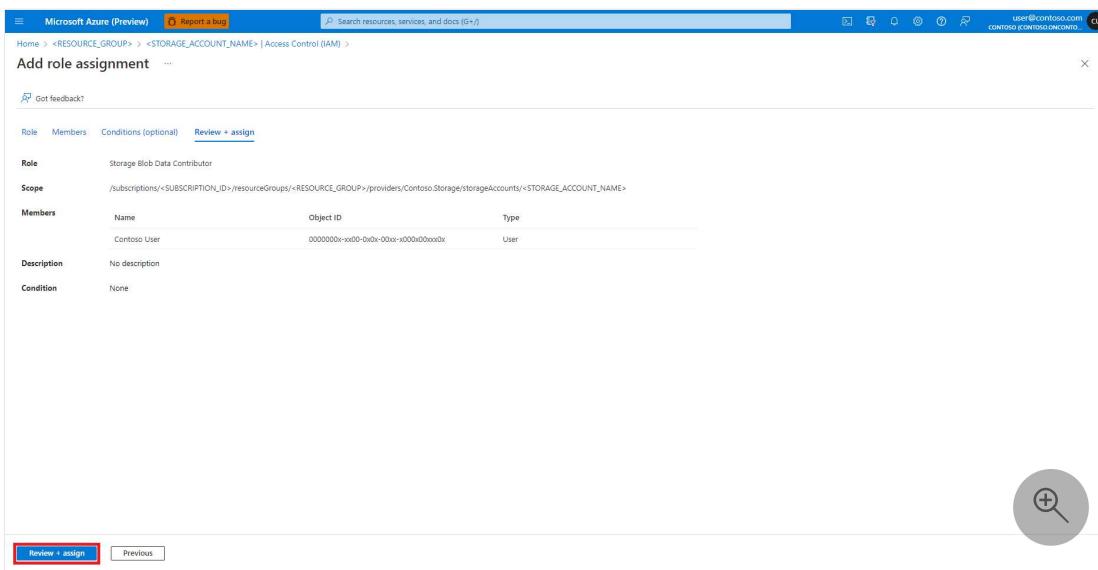
11. Select the user identity from the list, so that it shows under Selected members

12. Select the appropriate user identity

13. Select Next



14. Select Review + Assign

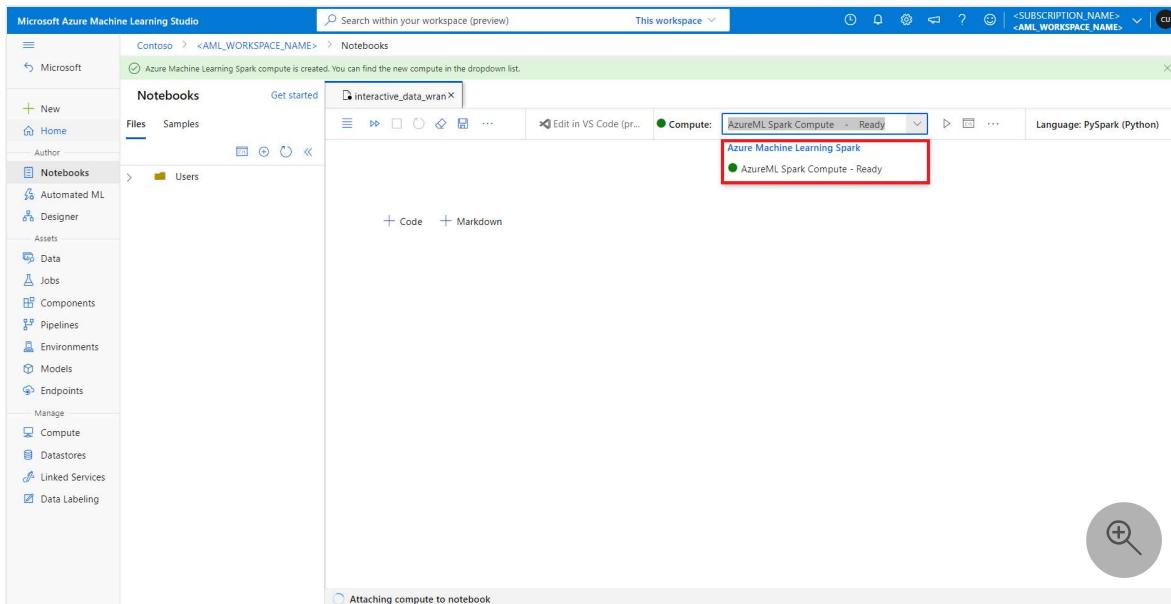


15. Repeat steps 2-13 for Contributor role assignment.

Once the user identity has the appropriate roles assigned, data in the Azure storage account should become accessible.

Managed (Automatic) Spark compute in Azure Machine Learning Notebooks

A Managed (Automatic) Spark compute is available in Azure Machine Learning Notebooks by default. To access it in a notebook, start in the **Compute** selection menu, and select **Azure Machine Learning Spark Compute** under **Azure Machine Learning Spark**.



Interactive data wrangling with Titanic data

💡 Tip

Data wrangling with a Managed (Automatic) Spark compute, and user identity passthrough for data access in an Azure Data Lake Storage (ADLS) Gen 2 storage account, both require the lowest number of configuration steps.

The data wrangling code shown here uses the `titanic.csv` file, available [here](#). Upload this file to a container created in the Azure Data Lake Storage (ADLS) Gen 2 storage account. This Python code snippet shows interactive data wrangling with an Azure Machine Learning Managed (Automatic) Spark compute, user identity passthrough, and an input/output data URI, in the

`abfss://<FILE_SYSTEM_NAME>@<STORAGE_ACCOUNT_NAME>.dfs.core.windows.net/<PATH_TO_DATA>` format. Here, `<FILE_SYSTEM_NAME>` matches the container name.

Python

```
import pyspark.pandas as pd
from pyspark.ml.feature import Imputer

df = pd.read_csv(
    "abfss://<FILE_SYSTEM_NAME>@<STORAGE_ACCOUNT_NAME>.dfs.core.windows.net/data/titanic.csv",
    index_col="PassengerId",
)
imputer = Imputer(inputCols=["Age"], outputCol="Age").setStrategy(
    "mean"
)
```

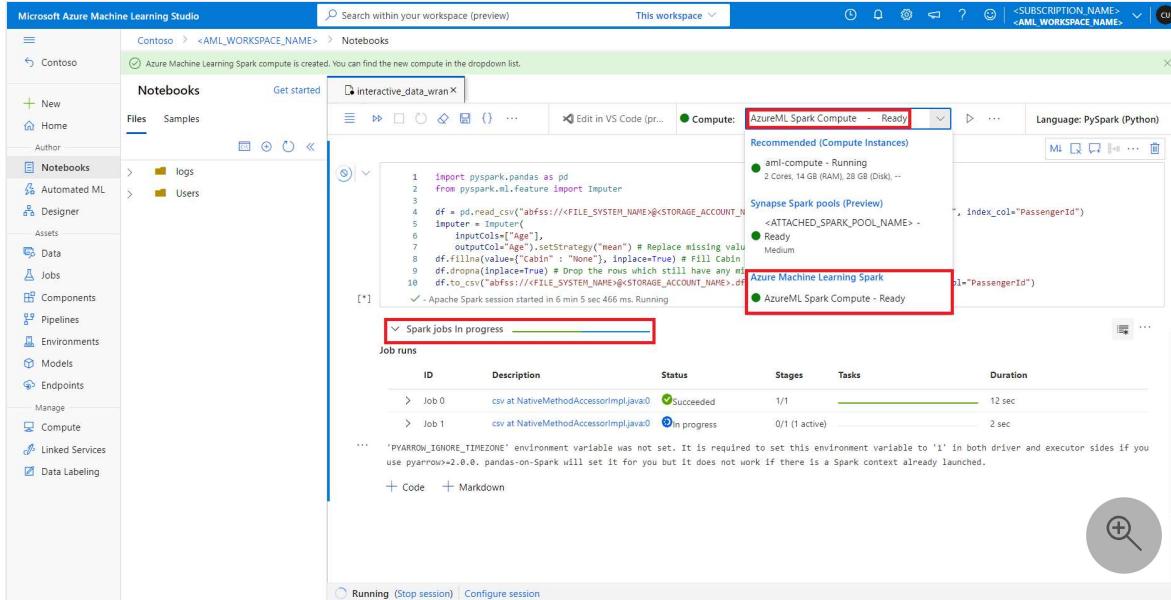
```

) # Replace missing values in Age column with the mean value
df.fillna(
    value={"Cabin": "None"}, inplace=True
) # Fill Cabin column with value "None" if missing
df.dropna(inplace=True) # Drop the rows which still have any missing value
df.to_csv(
    "abfss://<FILE_SYSTEM_NAME>@<STORAGE_ACCOUNT_NAME>.dfs.core.windows.net/data
    /wrangled",
    index_col="PassengerId",
)

```

⚠ Note

Only the Spark runtime version 3.2 supports `pyspark.pandas`, used in this Python code sample.



Next steps

- [Apache Spark in Azure Machine Learning \(preview\)](#)
- [Quickstart: Submit Apache Spark jobs in Azure Machine Learning \(preview\)](#)
- [Attach and manage a Synapse Spark pool in Azure Machine Learning \(preview\)](#)
- [Interactive Data Wrangling with Apache Spark in Azure Machine Learning \(preview\)](#)
- [Submit Spark jobs in Azure Machine Learning \(preview\)](#)
- [Code samples for Spark jobs using Azure Machine Learning CLI ↗](#)
- [Code samples for Spark jobs using Azure Machine Learning Python SDK ↗](#)

Additional resources

Documentation

[Azure Machine Learning glossary - Azure Machine Learning](#)

Glossary of terms for the Azure Machine Learning platform.

[Train deep learning Keras models \(SDK v2\) - Azure Machine Learning](#)

Learn how to train and register a Keras deep neural network classification model running on TensorFlow using Azure Machine Learning SDK (v2).

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Build & train models - Azure Machine Learning](#)

Learn how to train models with Azure Machine Learning. Explore the different training methods and choose the right one for your project.

[Experiment tracking and deploying models - Azure Data Science Virtual Machine](#)

Learn how to track and log experiments from the Data Science Virtual Machine with Azure Machine Learning and/or MLFlow.

[Export Data: Component Reference - Azure Machine Learning](#)

Use the Export Data component in Azure Machine Learning designer to save results and intermediate data outside of Azure Machine Learning.

[Tutorial: Train image classification model: VS Code \(preview\) - Azure Machine Learning](#)

Learn how to train an image classification model using TensorFlow and the Azure Machine Learning Visual Studio Code Extension

[Show 5 more](#)

Quickstart: Apache Spark jobs in Azure Machine Learning (preview)

Article • 02/22/2023 • 8 minutes to read

Important

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads.

Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

The Azure Machine Learning integration, with Azure Synapse Analytics (preview), provides easy access to distributed computing capability - backed by Azure Synapse - for scaling Apache Spark jobs on Azure Machine Learning.

In this quickstart guide, you learn how to submit a Spark job using Azure Machine Learning Managed (Automatic) Spark compute, Azure Data Lake Storage (ADLS) Gen 2 storage account, and user identity passthrough in a few simple steps.

For more information about [Apache Spark in Azure Machine Learning](#) concepts, see [this resource](#).

Prerequisites

CLI

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

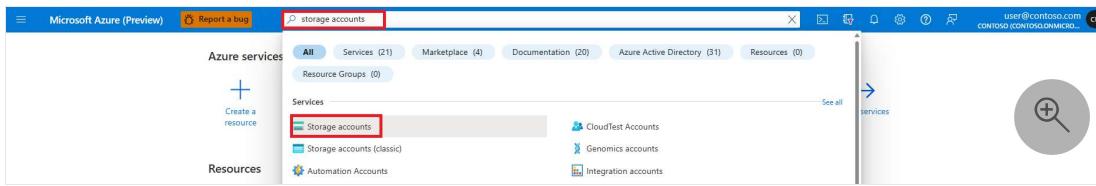
- An Azure subscription; if you don't have an Azure subscription, [create a free account](#) before you begin.
- An Azure Machine Learning workspace. See [Create workspace resources](#).
- An Azure Data Lake Storage (ADLS) Gen 2 storage account. See [Create an Azure Data Lake Storage \(ADLS\) Gen 2 storage account](#).
- [Create an Azure Machine Learning compute instance](#).
- [Install Azure Machine Learning CLI](#).

Add role assignments in Azure storage accounts

Before we submit an Apache Spark job, we must ensure that input, and output, data paths are accessible. Assign **Contributor** and **Storage Blob Data Contributor** roles to the user identity of the logged-in user to enable read and write access.

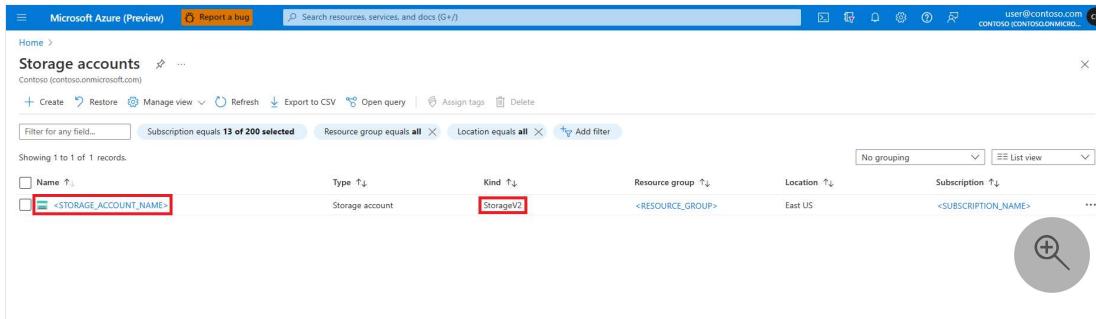
To assign appropriate roles to the user identity:

1. Open the [Microsoft Azure portal](#).
2. Search for, and select, the **Storage accounts** service.



The screenshot shows the Microsoft Azure (Preview) interface. At the top, there's a search bar with the text "storage accounts". Below the search bar, the navigation bar includes "All", "Services (21)", "Marketplace (4)", "Documentation (20)", "Azure Active Directory (31)", and "Resources (0)". Under the "Services" section, there's a "Create a resource" button and a list of services. The "Storage accounts" service is highlighted with a red box. Other services listed include "CloudTest Accounts", "Genomics accounts", and "Integration accounts".

3. On the **Storage accounts** page, select the Azure Data Lake Storage (ADLS) Gen 2 storage account from the list. A page showing **Overview** of the storage account opens.



The screenshot shows the "Storage accounts" overview page. At the top, there are buttons for "Create", "Restore", "Manage view", "Refresh", "Export to CSV", "Open query", "Assign tags", and "Delete". Below these are filters for "Subscription", "Resource group", and "Location". The main table lists one record: "Name": <STORAGE_ACCOUNT_NAME>, "Type": Storage account, "Kind": StorageV2, "Resource group": <RESOURCE_GROUP>, "Location": East US, and "Subscription": <SUBSCRIPTION_NAME>. The row for the storage account is highlighted with a red box.

4. Select **Access Control (IAM)** from the left panel.
5. Select **Add role assignment**.

The screenshot shows the Microsoft Azure Storage Account Access Control (IAM) page. The left sidebar lists various storage account management options, with 'Access Control (IAM)' highlighted by a red box. The main content area has tabs for 'Check access' (selected), 'Role assignments', 'Roles', 'Deny assignments', and 'Classic administrators'. Under 'Check access', there are three sections: 'Grant access to this resource' (with a 'Add role assignment' button highlighted by a red box), 'View access to this resource', and 'View deny assignments'.

6. Search for the role **Storage Blob Data Contributor**.

7. Select the role: **Storage Blob Data Contributor**.

8. Select **Next**.

The screenshot shows the 'Add role assignment' page. The search bar at the top contains 'Storage Blob' (highlighted by a red box). The table below lists roles, with 'Storage Blob Data Contributor' highlighted by a red box. The table columns are Role, Members, Conditions (optional), Review + assign, Description, Type, Category, and Details.

Role	Members	Conditions (optional)	Review + assign	Description	Type	Category	Details
Avere Cluster Runtime Operator				Avere Cluster runtime role used by Avere clusters running in subscriptions, for the purpose of failover IP addresses, accessing BLOB storage, etc	CustomRole	None	View
GenevaWarmPathStorageBlobContributor				Geneva Warm Path Storage Blob Contributor	CustomRole	None	View
Storage Blob Data Contributor				Allows for read, write and delete access to Azure Storage blob containers and data	BuiltinRole	Storage	View
Storage Blob Data Owner				Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.	BuiltinRole	Storage	View
Storage Blob Data Reader				Allows for read access to Azure Storage blob containers and data	BuiltinRole	Storage	View
Storage Blob Delegator				Allows for generation of a user delegation key which can be used to sign SAS tokens	BuiltinRole	Storage	View

9. Select **User, group, or service principal**.

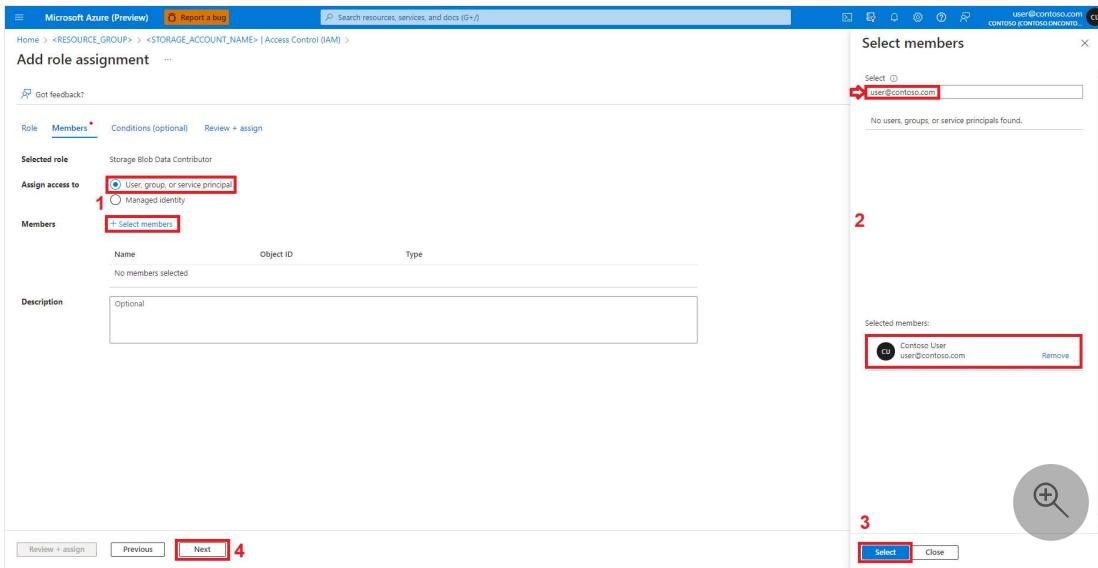
10. Select + **Select members**.

11. In the textbox under **Select**, search for the user identity.

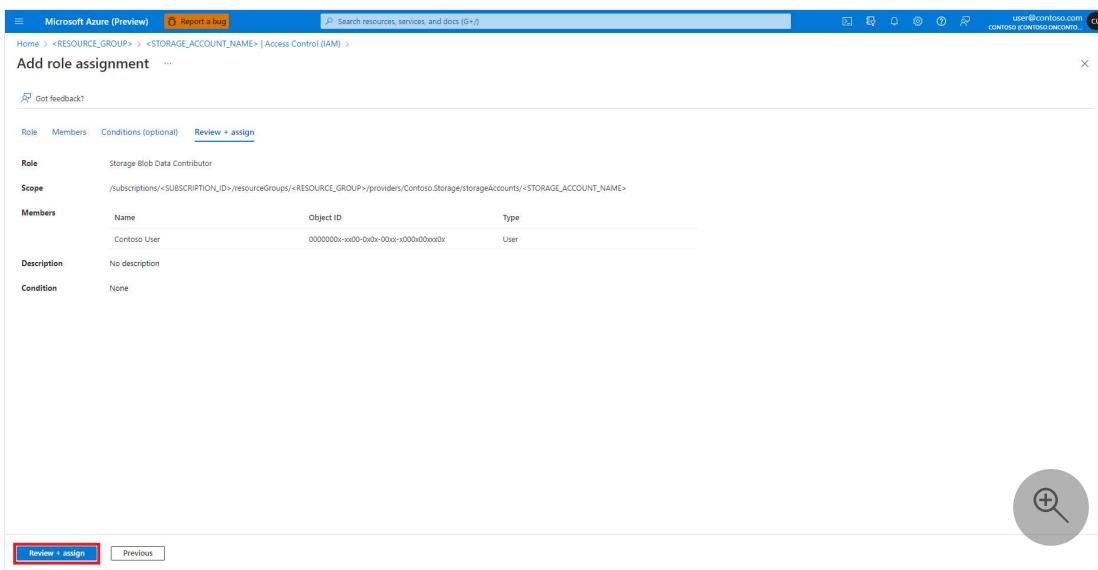
12. Select the user identity from the list so that it shows under **Selected members**.

13. Select the appropriate user identity.

14. Select **Next**.



15. Select Review + Assign.



16. Repeat steps 2-13 for Contributor role assignment.

Data in the Azure Data Lake Storage (ADLS) Gen 2 storage account should become accessible once the user identity has appropriate roles assigned.

Create parametrized Python code

A Spark job requires a Python script that takes arguments, which can be developed by modifying the Python code developed from [interactive data wrangling](#). A sample Python script is shown here.

Python

```
# titanic.py
import argparse
```

```
from operator import add
import pyspark.pandas as pd
from pyspark.ml.feature import Imputer

parser = argparse.ArgumentParser()
parser.add_argument("--titanic_data")
parser.add_argument("--wrangled_data")

args = parser.parse_args()
print(args.wrangled_data)
print(args.titanic_data)

df = pd.read_csv(args.titanic_data, index_col="PassengerId")
imputer = Imputer(inputCols=["Age"], outputCol="Age").setStrategy(
    "mean"
) # Replace missing values in Age column with the mean value
df.fillna(
    value={"Cabin": "None"}, inplace=True
) # Fill Cabin column with value "None" if missing
df.dropna(inplace=True) # Drop the rows which still have any missing value
df.to_csv(args.wrangled_data, index_col="PassengerId")
```

ⓘ Note

- This Python code sample uses `pyspark.pandas`, which is only supported by Spark runtime version 3.2.
- Please ensure that `titanic.py` file is uploaded to a folder named `src`. The `src` folder should be located in the same directory where you have created the Python script/notebook or the YAML specification file defining the standalone Spark job.

That script takes two arguments: `--titanic_data` and `--wrangled_data`. These arguments pass the input data path, and the output folder, respectively. The script uses the `titanic.csv` file, [available here](#). Upload this file to a container created in the Azure Data Lake Storage (ADLS) Gen 2 storage account.

Submit a standalone Spark job

CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

💡 Tip

You can submit a Spark job from:

- terminal of an Azure Machine Learning compute instance.
- terminal of Visual Studio Code connected to an Azure Machine Learning compute instance.
- your local computer that has the Azure Machine Learning CLI installed.

This example YAML specification shows a standalone Spark job. It uses an Azure Machine Learning Managed (Automatic) Spark compute, user identity passthrough, and input/output data URI in the

```
abfss://<FILE_SYSTEM_NAME>@<STORAGE_ACCOUNT_NAME>.dfs.core.windows.net/<PATH_TO_DATA>
```

format. Here, <FILE_SYSTEM_NAME> matches the container name.

YAML

```
$schema: http://azureml/sdk-2-0/SparkJob.json
type: spark

code: ./src
entry:
  file: titanic.py

conf:
  spark.driver.cores: 1
  spark.driver.memory: 2g
  spark.executor.cores: 2
  spark.executor.memory: 2g
  spark.executor.instances: 2

inputs:
  titanic_data:
    type: uri_file
    path:
      abfss://<FILE_SYSTEM_NAME>@<STORAGE_ACCOUNT_NAME>.dfs.core.windows.net/ata/titanic.csv
        mode: direct

outputs:
  wrangled_data:
    type: uri_folder
    path:
      abfss://<FILE_SYSTEM_NAME>@<STORAGE_ACCOUNT_NAME>.dfs.core.windows.net/ata/wrangled/
        mode: direct

args: >-
  --titanic_data ${inputs.titanic_data}
  --wrangled_data ${outputs.wrangled_data}
```

```
identity:  
  type: user_identity  
  
resources:  
  instance_type: standard_e4s_v3  
  runtime_version: "3.2"
```

In the above YAML specification file:

- `code` property defines relative path of the folder containing parameterized `titanic.py` file.
- `resource` property defines `instance_type` and Apache Spark `runtime_version` used by Managed (Automatic) Spark compute. The following instance types are currently supported:
 - `standard_e4s_v3`
 - `standard_e8s_v3`
 - `standard_e16s_v3`
 - `standard_e32s_v3`
 - `standard_e64s_v3`

The YAML file shown can be used in the `az ml job create` command, with the `--file` parameter, to create a standalone Spark job as shown:

Azure CLI

```
az ml job create --file <YAML_SPECIFICATION_FILE_NAME>.yaml --  
subscription <SUBSCRIPTION_ID> --resource-group <RESOURCE_GROUP> --  
workspace-name <AML_WORKSPACE_NAME>
```

💡 Tip

You might have an existing Synapse Spark pool in your Azure Synapse workspace. To use an existing Synapse Spark pool, please follow the instructions to [attach a Synapse Spark pool in Azure Machine Learning workspace](#).

Next steps

- [Apache Spark in Azure Machine Learning \(preview\)](#)
- [Quickstart: Interactive Data Wrangling with Apache Spark \(preview\)](#)
- [Attach and manage a Synapse Spark pool in Azure Machine Learning \(preview\)](#)
- [Interactive Data Wrangling with Apache Spark in Azure Machine Learning \(preview\)](#)

- Submit Spark jobs in Azure Machine Learning (preview)
 - Code samples for Spark jobs using Azure Machine Learning CLI ↗
 - Code samples for Spark jobs using Azure Machine Learning Python SDK ↗
-

Additional resources

Documentation

[Apache Spark in Azure Machine Learning \(preview\) - Azure Machine Learning](#)

This article explains the options for accessing Apache Spark in Azure Machine Learning.

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Interactive data wrangling with Apache Spark in Azure Machine Learning \(preview\) - Azure Machine Learning](#)

Learn how to use Apache Spark to wrangle data with Azure Machine Learning

[How to view AutoML model training code - Azure Machine Learning AutoML](#)

How to view model training code for an automated ML trained model and explanation of each stage.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Azure Machine Learning glossary - Azure Machine Learning](#)

Glossary of terms for the Azure Machine Learning platform.

[Train with MLflow Projects \(Preview\) - Azure Machine Learning](#)

Set up MLflow with Azure Machine Learning to log metrics and artifacts from ML models

[Experiment tracking and deploying models - Azure Data Science Virtual Machine](#)

Learn how to track and log experiments from the Data Science Virtual Machine with Azure Machine Learning and/or MLFlow.

[Show 5 more](#)

Quickstart: Run Jupyter notebooks in studio

Article • 12/13/2022 • 4 minutes to read

Get started with Azure Machine Learning by using Jupyter notebooks to learn more about the Python SDK.

In this quickstart, you'll learn how to run notebooks on a *compute instance* in Azure Machine Learning studio. A compute instance is an online compute resource that has a development environment already installed and ready to go.

You'll also learn where to find sample notebooks to help jump-start your path to training and deploying models with Azure Machine Learning.

Prerequisites

- An Azure account with an active subscription. [Create an account for free ↗](#).
- Run the [Quickstart: Create workspace resources you need to get started with Azure Machine Learning](#) to create a workspace and a compute instance.

Create a new notebook

Create a new notebook in studio.

1. Sign into [Azure Machine Learning studio ↗](#).
2. Select your workspace, if it isn't already open.
3. On the left, select **Notebooks**.
4. Select **Create new file**.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The top navigation bar displays 'Microsoft Azure Machine Learning Studio' and a search bar. The left sidebar contains links for 'Microsoft', 'New', 'Home', 'Author', 'Notebooks' (which is selected and highlighted with a red box), 'Automated ML', 'Designer', and 'Assets'. The main area is titled 'Notebooks' and shows two tabs: 'Files' (selected) and 'Samples'. Below the tabs is a toolbar with icons for 'Create new file' (highlighted with a red box), 'Create new folder', 'Upload files', and 'Upload folder'. The 'Files' tab shows a directory structure under 'Users': 'cqppublic'. A red box highlights the '+' icon in the toolbar.

5. Name your new notebook **my-new-notebook.ipynb**.

Create a markdown cell

1. On the upper right of each notebook cell is a toolbar of actions you can use for that cell. Select the **Convert to markdown cell** tool to change the cell to markdown.

The screenshot shows a Jupyter Notebook cell. The cell content is a single line of text '1' with the instruction 'Press shift + enter to run' below it. At the bottom of the cell are two buttons: '+ Code' and '+ Markdown'. In the upper right corner of the cell, there is a toolbar with several icons. One specific icon, labeled 'M1' and 'Convert to markdown cell - AzureML', is highlighted with a red box.

2. Double-click on the cell to open it.

3. Inside the cell, type:

```
markdown

# Testing a new notebook
Use markdown cells to add nicely formatted content to the notebook.
```

Create a code cell

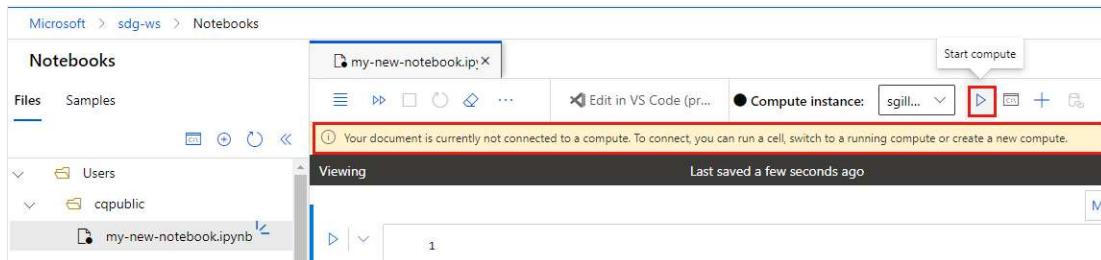
1. Just below the cell, select + **Code** to create a new code cell.

2. Inside this cell, add:

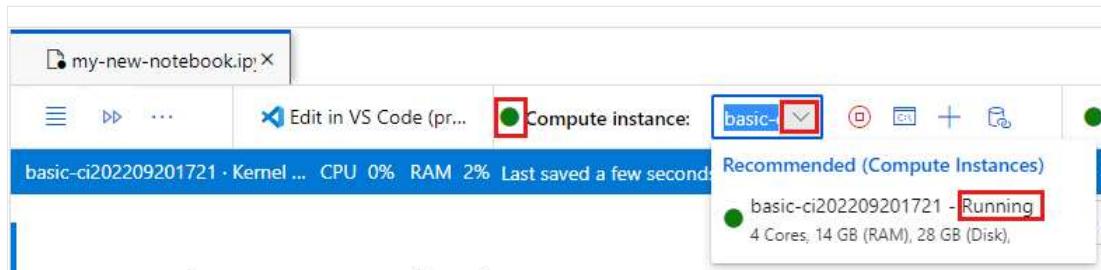
```
Python  
print("Hello, world!")
```

Run the code

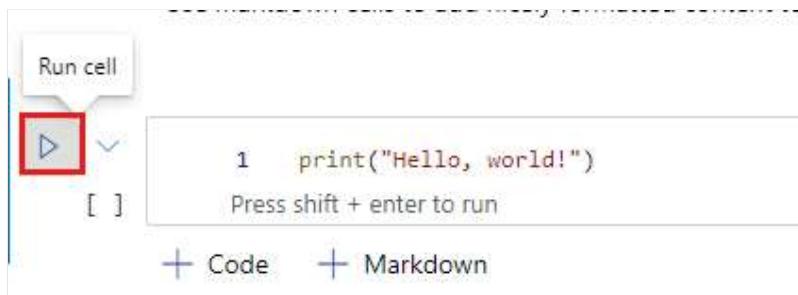
1. If you stopped your compute instance at the end of the [Quickstart: Create workspace resources you need to get started with Azure Machine Learning](#), start it again now:



2. Wait until the compute instance is "Running". When it is running, the **Compute instance** dot is green. You can also see the status after the compute instance name. You may have to select the arrow to see the full name.



3. You can run code cells either by using **Shift + Enter**, or by selecting the **Run cell** tool to the right of the cell. Use one of these methods to run the cell now.



4. The brackets to the left of the cell now have a number inside. The number represents the order in which cells were run. Since this is the first cell you've run, you'll see [1] next to the cell. You also see the output of the cell, `Hello, world!`.

5. Run the cell again. You'll see the same output (since you didn't change the code), but now the brackets contain [2]. As your notebook gets larger, these numbers help you understand what code was run, and in what order.

Run a second code cell

1. Add a second code cell:

```
Python

two = 1 + 1
print("One plus one is ",two)
```

2. Run the new cell.

3. Your notebook now looks like:

The screenshot shows a Jupyter Notebook interface with two code cells and their corresponding outputs. The first cell, labeled [2], contains the code `print("Hello, world!")`. The output of this cell, "Hello, world!", is shown in a red box. The second cell, labeled [3], contains the code `two = 1 + 1` followed by `print("One plus one is ",two)`. The output of this cell, "One plus one is 2", is also shown in a red box. The notebook toolbar at the bottom includes buttons for Code and Markdown.

```
Testing a new notebook

Use markdown cells to add nicely formatted content to the notebook.

[2]
1 print("Hello, world!")
✓ <1 sec
Hello, world!

[3]
1 two = 1 + 1
2 print("One plus one is ",two)
✓ <1 sec
...
... One plus one is 2

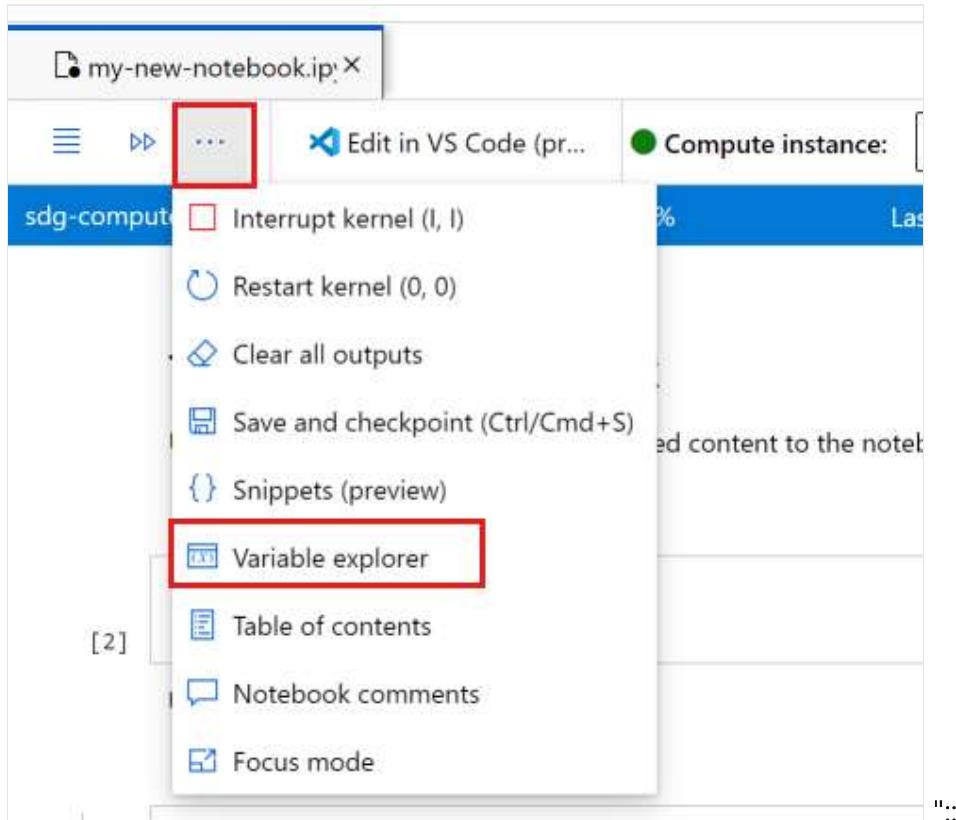
+ Code + Markdown
```

See your variables

Use the Variable explorer to see the variables that are defined in your session.

1. Select the "..." in the notebook toolbar.

2. Select Variable explorer.



The explorer appears at the bottom. You currently have one variable, `two`, assigned.

3. Add another code cell:

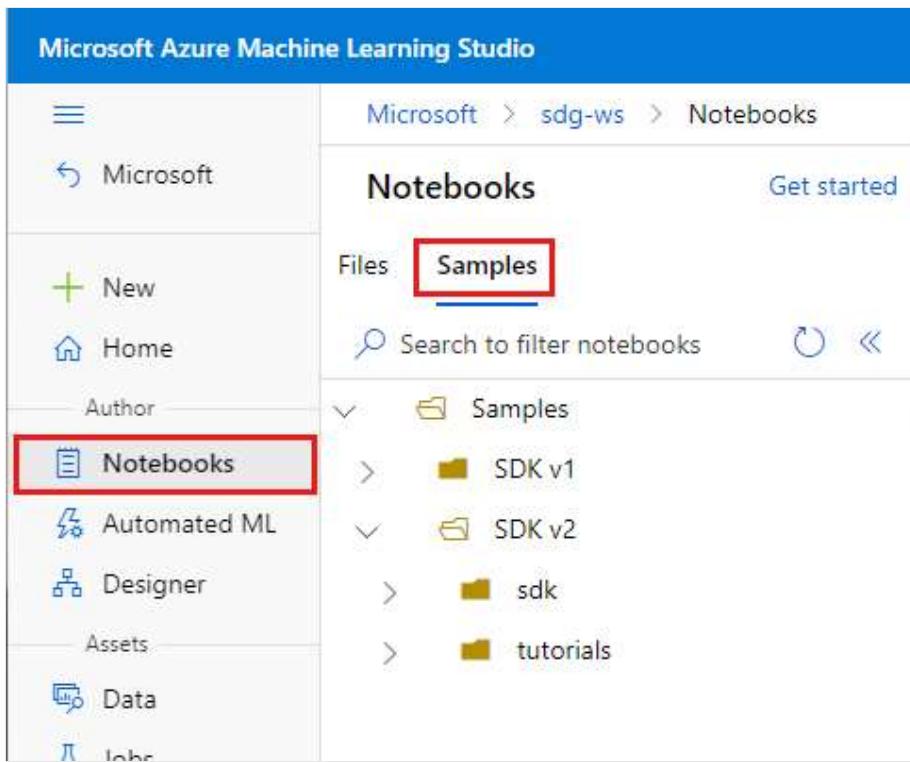
```
Python
three = 1+two
```

4. Run this cell to see the variable `three` appear in the variable explorer.

Learn from sample notebooks

There are sample notebooks available in studio to help you learn more about Azure Machine Learning. To find these samples:

1. Still in the **Notebooks** section, select **Samples** at the top.

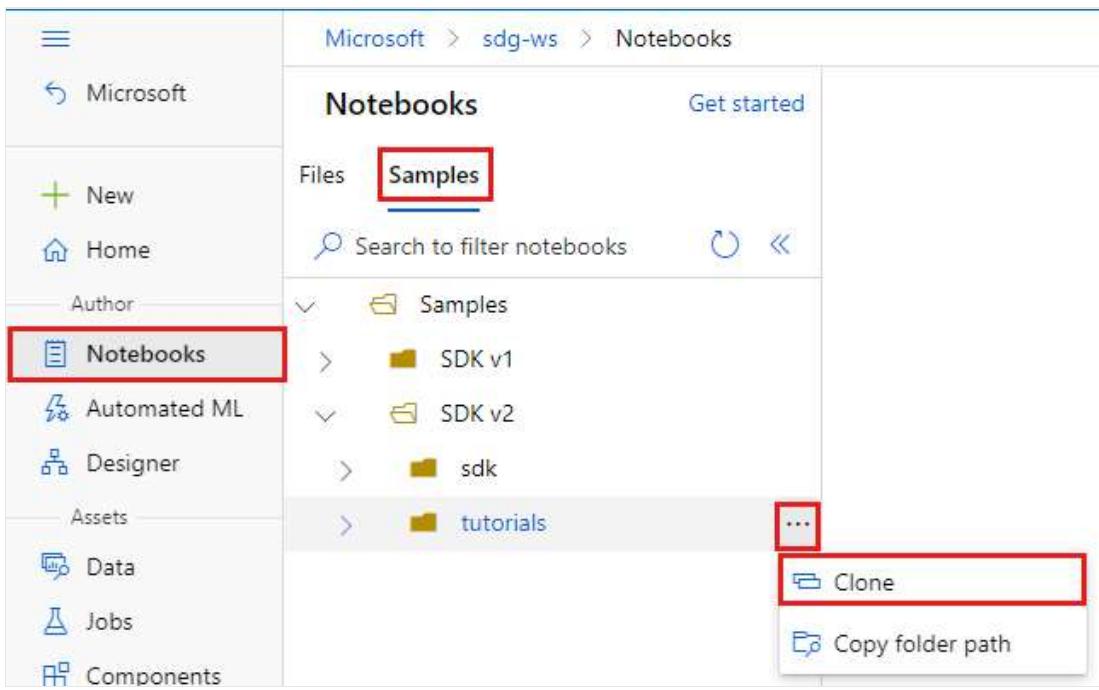


2. The **SDK v1** folder can be used with the previous, v1 version of the SDK. If you're just starting, you won't need these samples.
3. Use notebooks in the **SDK v2** folder for examples that show the current version of the SDK, v2.
4. Select the notebook **SDK v2/tutorials/azureml-in-a-day/azureml-in-a-day.ipynb**. You'll see a read-only version of the notebook.
5. To get your own copy, you can select **Clone this notebook**. This action will also copy the rest of the folder's content for that notebook. No need to do that now, though, as you're going to instead clone the whole folder.

Clone tutorials folder

You can also clone an entire folder. The **tutorials** folder is a good place to start learning more about how Azure Machine Learning works.

1. Open the **SDK v2** folder.
2. Select the "..." at the right of **tutorials** folder to get the menu, then select **Clone**.



3. Your new folder is now displayed in the **Files** section.

4. Run the notebooks in this folder to learn more about using the Python SDK v2 to train and deploy models.

Clean up resources

If you plan to continue now to the next tutorial, skip to [Next steps](#).

Delete all resources

i Important

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation sidebar with options like Home, Dashboard, All services, and Favorites (All resources, Resource groups, App Services, SQL databases, SQL data warehouses, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks). The main content area is titled 'Resource groups > my-rg'. It shows an 'Overview' tab selected. Below it are tabs for Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Deployments, Policies, Properties, Locks, and Export template. The 'Delete resource group' button is highlighted with a red box. The main content area also displays subscription information (Visual Studio Ultimate with MSDN), deployment details (1 succeeded), and a list of resources including 'myworkspace', 'my-workspace', and 'myworkspace0013141752'.

4. Enter the resource group name. Then select **Delete**.

Next steps

[Tutorial: Azure Machine Learning in a day](#)

Additional resources

Documentation

[Use automated ML in ML pipelines - Azure Machine Learning](#)

The AutoMLStep allows you to use automated machine learning in your pipelines.

[How to view AutoML model training code - Azure Machine Learning AutoML](#)

How to view model training code for an automated ML trained model and explanation of each stage.

[Tutorial: Train a first Python machine learning model \(SDK v1\) - Azure Machine Learning](#)

How to train a machine learning model in Azure Machine Learning, with SDK v1. This is part 2 of a three-part getting-started series.

[Train deep learning Keras models \(SDK v2\) - Azure Machine Learning](#)

Learn how to train and register a Keras deep neural network classification model running on TensorFlow using Azure Machine Learning SDK (v2).

[Tutorial: Upload data and train a model \(SDK v1\) - Azure Machine Learning](#)

How to upload and use your own data in a remote training job, with SDK v1. This is part 3 of a three-part getting-started series.

[Tutorial: Get started with a Python script \(v1\) - Azure Machine Learning](#)

Get started with your first Python script in Azure Machine Learning, with SDK v1. This is part 1 of a three-part getting-started series.

[Hyperparameter for AutoML computer vision tasks - Azure Machine Learning](#)

Learn which hyperparameters are available for computer vision tasks with automated ML.

[Train scikit-learn machine learning models \(v2\) - Azure Machine Learning](#)

Learn how Azure Machine Learning enables you to scale out a scikit-learn training job using elastic cloud compute resources (v2).

[Show 5 more](#)

Apache Spark in Azure Machine Learning (preview)

Article • 02/10/2023 • 5 minutes to read

Azure Machine Learning integration with Azure Synapse Analytics (preview) provides easy access to distributed computation resources through the Apache Spark framework. This integration offers these Apache Spark computing experiences:

- Managed (Automatic) Spark compute
- Attached Synapse Spark pool

Managed (Automatic) Spark compute

With the Apache Spark framework, Azure Machine Learning Managed (Automatic) Spark compute is the easiest way to accomplish distributed computing tasks in the Azure Machine Learning environment. Azure Machine Learning offers a fully managed, serverless, on-demand Apache Spark compute cluster. Its users can avoid the need to create an Azure Synapse workspace and a Synapse Spark pool.

Users can define resources, including instance type and Apache Spark runtime version. They can then use those resources to access Managed (Automatic) Spark compute in Azure Machine Learning notebooks for:

- [Interactive Spark code development](#)
- [Spark batch job submissions](#)
- [Running machine learning pipelines with a Spark component](#)

Points to consider

Managed (Automatic) Spark compute works well for most user scenarios that require quick access to distributed computing through Apache Spark. However, to make an informed decision, users should consider the advantages and disadvantages of this approach.

Advantages:

- No dependencies on other Azure resources to be created for Apache Spark (Azure Synapse infrastructure operates under the hood).
- No required subscription permissions to create Azure Synapse-related resources.
- No need for SQL pool quotas.

Disadvantages:

- A persistent Hive metastore is missing. Managed (Automatic) Spark compute supports only in-memory Spark SQL.
- No available tables or databases.
- Missing Azure Purview integration.
- No available linked services.
- Fewer data sources and connectors.
- No pool-level configuration.
- No pool-level library management.
- Only partial support for `mssqlutils`.

Network configuration

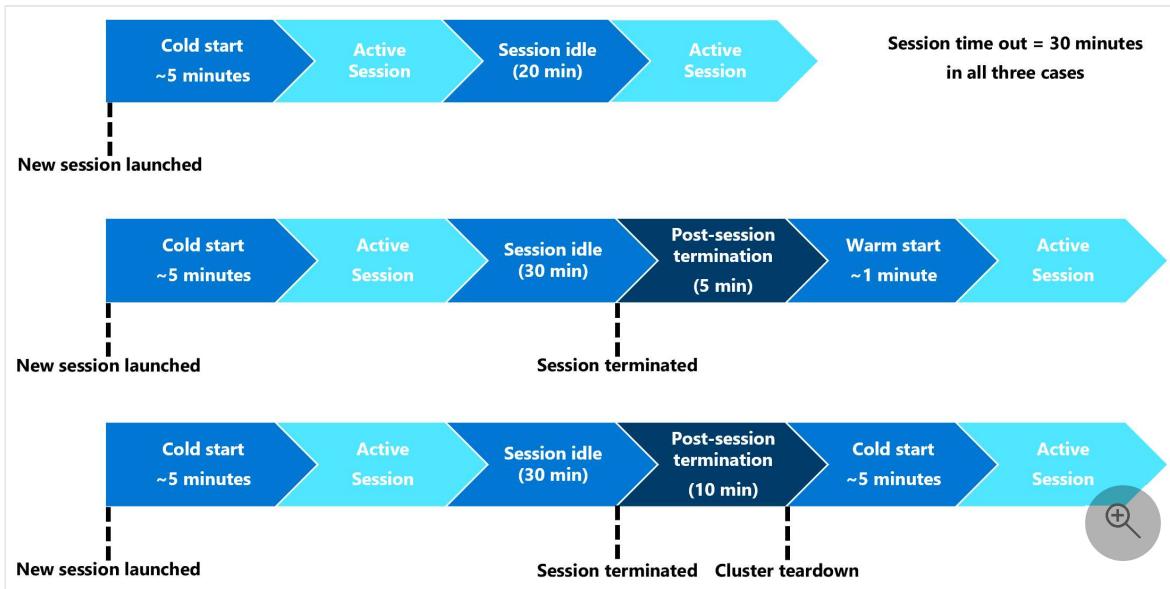
As of January 2023, creation of a Managed (Automatic) Spark compute, inside a virtual network, and creation of a private endpoint to Azure Synapse, aren't supported.

Inactivity periods and tear-down mechanism

At first launch, Managed (Automatic) Spark compute (*cold start*) resource might need three to five minutes to start the Spark session itself. The automated Managed (Automatic) Spark compute provisioning, backed by Azure Synapse, causes this delay. After the Managed (Automatic) Spark compute is provisioned, and an Apache Spark session starts, subsequent code executions (*warm start*) won't experience this delay.

The Spark session configuration offers an option that defines a session timeout (in minutes). The Spark session will end after an inactivity period that exceeds the user-defined timeout. If another Spark session doesn't start in the following ten minutes, resources provisioned for the Managed (Automatic) Spark compute will be torn down.

After the Managed (Automatic) Spark compute resource tear-down happens, submission of the next job will require a *cold start*. The next visualization shows some session inactivity period and cluster teardown scenarios.



ⓘ Note

For a session-level conda package:

- *Cold start* time will need about ten to fifteen minutes.
- *Warm start* time using same conda package will need about one minute.
- *Warm start* with a different conda package will also need about ten to fifteen minutes.

Attached Synapse Spark pool

A Spark pool created in an Azure Synapse workspace becomes available in the Azure Machine Learning workspace with the attached Synapse Spark pool. This option might be suitable for users who want to reuse an existing Synapse Spark pool.

Attachment of a Synapse Spark pool to an Azure Machine Learning workspace requires [other steps](#) before you can use the pool in Azure Machine Learning for:

- [Interactive Spark code development](#)
- [Spark batch job submission](#)
- [Running machine learning pipelines with a Spark component](#)

An attached Synapse Spark pool provides access to native Azure Synapse features. The user is responsible for the Synapse Spark pool provisioning, attaching, configuration, and management.

The Spark session configuration for an attached Synapse Spark pool also offers an option to define a session timeout (in minutes). The session timeout behavior resembles

the description in [the previous section](#), except that the associated resources are never torn down after the session timeout.

Defining Spark cluster size

You can define Spark cluster size with three parameter values in Azure Machine Learning Spark jobs:

- Number of executors
- Executor cores
- Executor memory

You should consider an Azure Machine Learning Apache Spark executor as an equivalent of Azure Spark worker nodes. An example can explain these parameters. Let's say that you defined the number of executors as 6 (equivalent to six worker nodes), executor cores as 4, and executor memory as 28 GB. Your Spark job then has access to a cluster with 24 cores and 168 GB of memory.

Ensuring resource access for Spark jobs

To access data and other resources, a Spark job can use either a user identity passthrough or a managed identity. This table summarizes the mechanisms that Spark jobs use to access resources.

Spark pool	Supported identities	Default identity
Managed (Automatic) Spark compute	User identity and managed identity	User identity
Attached Synapse Spark pool	User identity and managed identity	Managed identity - compute identity of the attached Synapse Spark pool

[This article](#) describes resource access for Spark jobs. In a notebook session, both the Managed (Automatic) Spark compute and the attached Synapse Spark pool use user identity passthrough for data access during [interactive data wrangling](#).

ⓘ Note

- To ensure successful Spark job execution, assign **Contributor** and **Storage Blob Data Contributor** roles (on the Azure storage account used for data input and output) to the identity that's used for submitting the Spark job.

- If an attached Synapse Spark pool points to a Synapse Spark pool in an Azure Synapse workspace, and that workspace has an associated managed virtual network, **configure a managed private endpoint to a storage account**. This configuration will help ensure data access.
- Both Managed (Automatic) Spark compute and attached Synapse Spark pool do not work in a notebook created in a private link enabled workspace.

This [quickstart](#) describes how to start using Managed (Automatic) Spark compute in Azure Machine Learning.

Next steps

- [Quickstart: Submit Apache Spark jobs in Azure Machine Learning \(preview\)](#)
 - [Attach and manage a Synapse Spark pool in Azure Machine Learning \(preview\)](#)
 - [Interactive data wrangling with Apache Spark in Azure Machine Learning \(preview\)](#)
 - [Submit Spark jobs in Azure Machine Learning \(preview\)](#)
 - [Code samples for Spark jobs using the Azure Machine Learning CLI ↗](#)
 - [Code samples for Spark jobs using the Azure Machine Learning Python SDK ↗](#)
-

Additional resources

Documentation

[Create compute clusters CLI v1 - Azure Machine Learning](#)

Learn how to create compute clusters in your Azure Machine Learning workspace with CLI v1. Use the compute cluster as a compute target for training or inference.

[Using MLflow models in batch deployments - Azure Machine Learning](#)

Learn how to deploy MLflow models in batch deployments

[Submit Spark jobs in Azure Machine Learning \(preview\) - Azure Machine Learning](#)

Learn how to submit standalone and pipeline Spark jobs in Azure Machine Learning

[mltable.mltable module - Azure Machine Learning Python](#)

Contains functionality to create and interact with MLTable objects

[Interactive data wrangling with Apache Spark in Azure Machine Learning \(preview\) - Azure Machine Learning](#)

Learn how to use Apache Spark to wrangle data with Azure Machine Learning

[mltable package - Azure Machine Learning Python](#)

Contains functionality for interacting with existing and creating new MLTable files. With the mltable package you can load, transform, and analyze data in any Python environment, including Jupyter Notebooks or your favorite Python IDE.

[Package models - Azure Machine Learning](#)

Package a model. Models can be packaged as either a docker image, which you can then download, or you can create a Dockerfile and use it to build the image.

[Use Apache Spark in a machine learning pipeline \(deprecated\) - Azure Machine Learning](#)

Link your Azure Synapse Analytics workspace to your Azure machine learning pipeline to use Apache Spark for data manipulation.

[Show 5 more](#)

Tutorial: Azure Machine Learning in a day

Article • 02/24/2023 • 12 minutes to read

APPLIES TO:  Python SDK azure-ai-ml v2 (current) ↗

Learn how a data scientist uses Azure Machine Learning to train a model, then use the model for prediction. This tutorial will help you become familiar with the core concepts of Azure Machine Learning and their most common usage.

You'll learn how to submit a *command job* to run your *training script* on a specified *compute resource*, configured with the *job environment* necessary to run the script.

The *training script* handles the data preparation, then trains and registers a model. Once you have the model, you'll deploy it as an *endpoint*, then call the endpoint for inferencing.

The steps you'll take are:

- ✓ Connect to your Azure Machine Learning workspace
- ✓ Create your compute resource and job environment
- ✓ Create your training script
- ✓ Create and run your command job to run the training script on the compute resource, configured with the appropriate job environment
- ✓ View the output of your training script
- ✓ Deploy the newly-trained model as an endpoint
- ✓ Call the Azure Machine Learning endpoint for inferencing

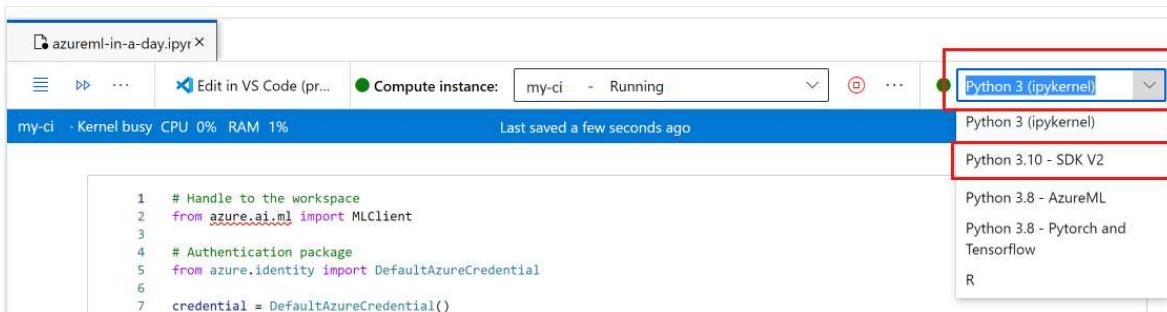
Prerequisites

- Complete the [Quickstart: Get started with Azure Machine Learning](#) to:
 - Create a workspace.
 - Create a cloud-based compute instance to use for your development environment.
- Create a new notebook or copy our notebook.
 - Follow the [Quickstart: Run Jupyter notebook in Azure Machine Learning studio](#) steps to create a new notebook.
 - Or use the steps in the quickstart to [clone the v2 tutorials folder](#), then open the notebook from the `tutorials/azureml-in-a-day/azureml-in-a-day.ipynb` folder in your **File** section.

Run your notebook

1. On the top bar, select the compute instance you created during the [Quickstart: Get started with Azure Machine Learning](#) to use for running the notebook.

2. Make sure that the kernel, found on the top right, is `Python 3.10 - SDK v2`. If not, use the dropdown to select this kernel.



ⓘ Important

The rest of this tutorial contains cells of the tutorial notebook. Copy/paste them into your new notebook, or switch to the notebook now if you cloned it.

To run a single code cell in a notebook, click the code cell and hit **Shift+Enter**. Or, run the entire notebook by choosing **Run all** from the top toolbar.

Connect to the workspace

Before you dive in the code, you'll need to connect to your Azure Machine Learning workspace. The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning.

We're using `DefaultAzureCredential` to get access to workspace.

`DefaultAzureCredential` is used to handle most Azure SDK authentication scenarios.

```
Python

# Handle to the workspace
from azure.ai.ml import MLClient

# Authentication package
from azure.identity import DefaultAzureCredential

credential = DefaultAzureCredential()
```

In the next cell, enter your Subscription ID, Resource Group name and Workspace name. To find these values:

1. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
2. Copy the value for workspace, resource group and subscription ID into the code.
3. You'll need to copy one value, close the area and paste, then come back for the next one.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a code cell containing Python code to initialize an MLClient. On the right, a 'Directory + Subscription + Workspace' pane is open, showing the current workspace set to 'my-workspace'. The 'Resource Group' field contains 'captopic-rg' with a copy icon highlighted with a red box. Below it, the 'Subscription ID' field contains a long GUID-like string 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX' with a copy icon highlighted with a red box. The 'Location' field shows 'eastus2'. At the bottom of the pane are 'Download config file' and 'View all properties in Azure Portal' buttons. A 'Create a Compute Resource to run our job' section is also visible at the bottom.

The screenshot shows the Azure Machine Learning studio interface with a Python code cell. The code is identical to the one in the previous screenshot, initializing an MLClient with specific parameters. The cell has '+ Code' and '+ Markdown' options below it. At the bottom right of the cell are 'Submit' and 'Cancel' buttons.

The result is a handler to the workspace that you'll use to manage other resources and jobs.

Important

Creating MLClient will not connect to the workspace. The client initialization is lazy, it will wait for the first time it needs to make a call (in the notebook below, that will happen during compute creation).

Create a compute resource to run your job

You'll need a compute resource for running a job. It can be single or multi-node machines with Linux or Windows OS, or a specific compute fabric like Spark.

You'll provision a Linux compute cluster. See the [full list on VM sizes and prices](#).

For this example, you only need a basic cluster, so you'll use a Standard_DS3_v2 model with 2 vCPU cores, 7-GB RAM and create an Azure Machine Learning Compute.

Python

```
from azure.ai.ml.entities import AmlCompute

# Name assigned to the compute cluster
cpu_compute_target = "cpu-cluster"

try:
    # let's see if the compute target already exists
    cpu_cluster = ml_client.compute.get(cpu_compute_target)
    print(
        f"You already have a cluster named {cpu_compute_target}, we'll reuse
it as is."
    )

except Exception:
    print("Creating a new cpu compute target...")

    # Let's create the Azure ML compute object with the intended parameters
    cpu_cluster = AmlCompute(
        name=cpu_compute_target,
        # Azure ML Compute is the on-demand VM service
        type="amlcompute",
        # VM Family
        size="STANDARD_DS3_V2",
        # Minimum running nodes when there is no job running
        min_instances=0,
        # Nodes in cluster
        max_instances=4,
        # How many seconds will the node running after the job termination
        idle_time_before_scale_down=180,
        # Dedicated or LowPriority. The latter is cheaper but there is a
```

```
chance of job termination
    tier="Dedicated",
)
print(
    f"AMLCompute with name {cpu_cluster.name} will be created, with
compute size {cpu_cluster.size}"
)
# Now, we pass the object to MLClient's create_or_update method
cpu_cluster = ml_client.compute.begin_create_or_update(cpu_cluster)
```

Create a job environment

To run your Azure Machine Learning job on your compute resource, you'll need an [environment](#). An environment lists the software runtime and libraries that you want installed on the compute where you'll be training. It's similar to your Python environment on your local machine.

Azure Machine Learning provides many curated or ready-made environments, which are useful for common training and inference scenarios. You can also create your own custom environments using a docker image, or a conda configuration.

In this example, you'll create a custom conda environment for your jobs, using a `conda.yaml` file.

First, create a directory to store the file in.

```
Python

import os

dependencies_dir = "./dependencies"
os.makedirs(dependencies_dir, exist_ok=True)
```

Now, create the file in the `dependencies` directory. The cell below uses IPython magic to write the file into the directory you just created.

```
Python

%%writefile {dependencies_dir}/conda.yaml
name: model-env
channels:
- conda-forge
dependencies:
- python=3.8
- numpy=1.21.2
- pip=21.2.4
- scikit-learn=0.24.2
```

```
- scipy=1.7.1
- pandas>=1.1,<1.2
- pip:
  - inference-schema[numpy-support]==1.3.0
  - xlrd==2.0.1
  - mlflow== 1.26.1
  - azureml-mlflow==1.42.0
  - psutil>=5.8,<5.9
  - tqdm>=4.59,<4.60
  - ipykernel~=6.0
- matplotlib
```

The specification contains some usual packages, that you'll use in your job (numpy, pip).

Reference this *yaml* file to create and register this custom environment in your workspace:

Python

```
from azure.ai.ml.entities import Environment

custom_env_name = "aml-scikit-learn"

pipeline_job_env = Environment(
    name=custom_env_name,
    description="Custom environment for Credit Card Defaults pipeline",
    tags={"scikit-learn": "0.24.2"},
    conda_file=os.path.join(dependencies_dir, "conda.yml"),
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest",
)
pipeline_job_env = ml_client.environments.create_or_update(pipeline_job_env)

print(
    f"Environment with name {pipeline_job_env.name} is registered to
workspace, the environment version is {pipeline_job_env.version}"
)
```

What is a command job?

You'll create an Azure Machine Learning *command job* to train a model for credit default prediction. The command job is used to run a *training script* in a specified environment on a specified compute resource. You've already created the environment and the compute resource. Next you'll create the training script.

The *training script* handles the data preparation, training and registering of the trained model. In this tutorial, you'll create a Python training script.

Command jobs can be run from CLI, Python SDK, or studio interface. In this tutorial, you'll use the Azure Machine Learning Python SDK v2 to create and run the command job.

After running the training job, you'll deploy the model, then use it to produce a prediction.

Create training script

Let's start by creating the training script - the *main.py* Python file.

First create a source folder for the script:

```
Python

import os

train_src_dir = "./src"
os.makedirs(train_src_dir, exist_ok=True)
```

This script handles the preprocessing of the data, splitting it into test and train data. It then consumes this data to train a tree based model and return the output model.

[MLFlow](#) will be used to log the parameters and metrics during our pipeline run.

The cell below uses IPython magic to write the training script into the directory you just created.

```
Python

%%writefile {train_src_dir}/main.py
import os
import argparse
import pandas as pd
import mlflow
import mlflow.sklearn
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

def main():
    """Main function of the script."""

    # input and output arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, help="path to input data")
    parser.add_argument("--test_train_ratio", type=float, required=False,
default=0.25)
```

```

parser.add_argument("--n_estimators", required=False, default=100,
type=int)
parser.add_argument("--learning_rate", required=False, default=0.1,
type=float)
parser.add_argument("--registered_model_name", type=str, help="model
name")
args = parser.parse_args()

# Start Logging
mlflow.start_run()

# enable autologging
mlflow.sklearn.autolog()

#####
#<prepare the data>
#####
print(" ".join(f"{k}={v}" for k, v in vars(args).items()))

print("input data:", args.data)

credit_df = pd.read_excel(args.data, header=1, index_col=0)

mlflow.log_metric("num_samples", credit_df.shape[0])
mlflow.log_metric("num_features", credit_df.shape[1] - 1)

train_df, test_df = train_test_split(
    credit_df,
    test_size=args.test_train_ratio,
)
#####
#</prepare the data>
#####

#####
#<train the model>
#####
# Extracting the label column
y_train = train_df.pop("default payment next month")

# convert the dataframe values to array
X_train = train_df.values

# Extracting the label column
y_test = test_df.pop("default payment next month")

# convert the dataframe values to array
X_test = test_df.values

print(f"Training with data of shape {X_train.shape}")

clf = GradientBoostingClassifier(
    n_estimators=args.n_estimators, learning_rate=args.learning_rate
)
clf.fit(X_train, y_train)

```

```

y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
#####
#</train the model>
#####

#####
#<save and register model>
#####
# Registering the model to the workspace
print("Registering the model via MLFlow")
mlflow.sklearn.log_model(
    sk_model=clf,
    registered_model_name=args.registered_model_name,
    artifact_path=args.registered_model_name,
)

# Saving the model to a file
mlflow.sklearn.save_model(
    sk_model=clf,
    path=os.path.join(args.registered_model_name, "trained_model"),
)
#####
#</save and register model>
#####

# Stop Logging
mlflow.end_run()

if __name__ == "__main__":
    main()

```

As you can see in this script, once the model is trained, the model file is saved and registered to the workspace. Now you can use the registered model in inferencing endpoints.

Configure the command

Now that you have a script that can perform the desired tasks, you'll use the general purpose **command** that can run command line actions. This command line action can be directly calling system commands or by running a script.

Here, you'll create input variables to specify the input data, split ratio, learning rate and registered model name. The command script will:

- Use the compute created earlier to run this command.

- Use the environment created earlier - you can use the `@latest` notation to indicate the latest version of the environment when the command is run.
- Configure some metadata like display name, experiment name etc. An *experiment* is a container for all the iterations you do on a certain project. All the jobs submitted under the same experiment name would be listed next to each other in Azure Machine Learning studio.
- Configure the command line action itself - `python main.py` in this case. The inputs/outputs are accessible in the command via the `$("#...")` notation.

Python

```
from azure.ai.ml import command
from azure.ai.ml import Input

registered_model_name = "credit_defaults_model"

job = command(
    inputs=dict(
        data=Input(
            type="uri_file",
            path="https://archive.ics.uci.edu/ml/machine-learning-
databases/00350/default%20of%20credit%20card%20clients.xls",
        ),
        test_train_ratio=0.2,
        learning_rate=0.25,
        registered_model_name=registered_model_name,
    ),
    code=".src/", # location of source code
    command="python main.py --data ${inputs.data} --test_train_ratio
${inputs.test_train_ratio} --learning_rate ${inputs.learning_rate} --
registered_model_name ${inputs.registered_model_name}",
    environment="aml-scikit-learn@latest",
    compute="cpu-cluster",
    experiment_name="train_model_credit_default_prediction",
    display_name="credit_default_prediction",
)
```

Submit the job

It's now time to submit the job to run in Azure Machine Learning. This time you'll use `create_or_update` on `ml_client.jobs`.

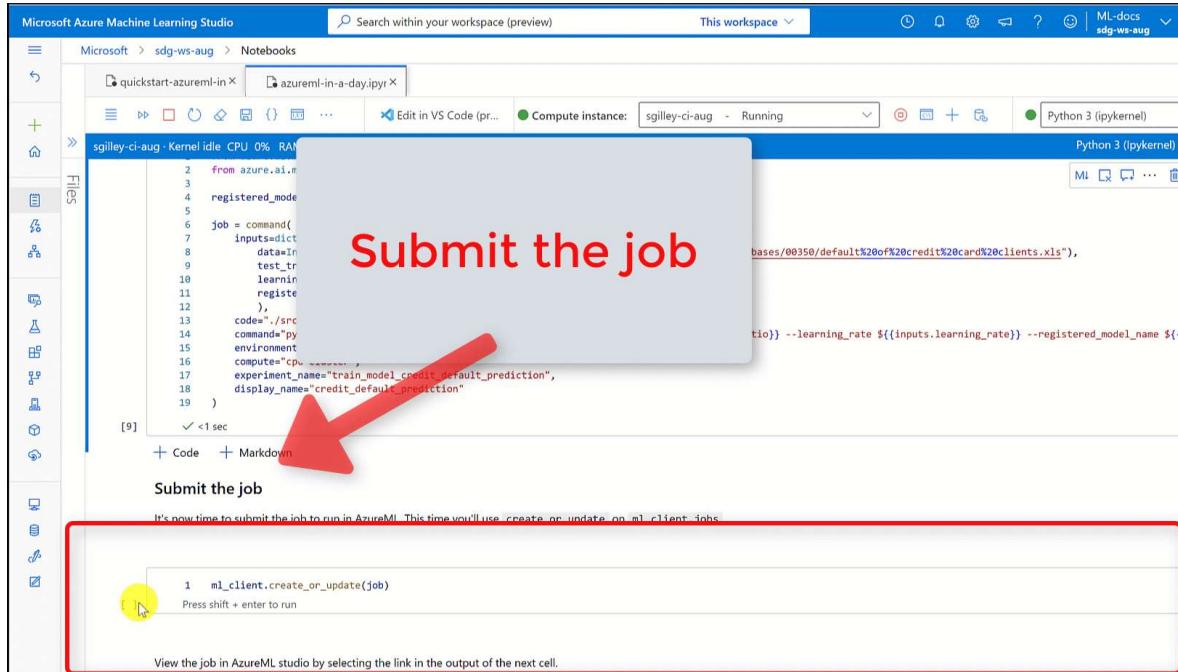
Python

```
ml_client.create_or_update(job)
```

View job output and wait for job completion

View the job in Azure Machine Learning studio by selecting the link in the output of the previous cell.

The output of this job will look like this in the Azure Machine Learning studio. Explore the tabs for various details like metrics, outputs etc. Once completed, the job will register a model in your workspace as a result of training.



Important

Wait until the status of the job is complete before returning to this notebook to continue. The job will take 2 to 3 minutes to run. It could take longer (up to 10 minutes) if the compute cluster has been scaled down to zero nodes and custom environment is still building.

Deploy the model as an online endpoint

Now deploy your machine learning model as a web service in the Azure cloud, an [online endpoint](#).

To deploy a machine learning service, you usually need:

- The model assets (file, metadata) that you want to deploy. You've already registered these assets in your training job.

- Some code to run as a service. The code executes the model on a given input request. This entry script receives data submitted to a deployed web service and passes it to the model, then returns the model's response to the client. The script is specific to your model. The entry script must understand the data that the model expects and returns. With an MLFlow model, as in this tutorial, this script is automatically created for you. Samples of scoring scripts can be found [here](#).

Create a new online endpoint

Now that you have a registered model and an inference script, it's time to create your online endpoint. The endpoint name needs to be unique in the entire Azure region. For this tutorial, you'll create a unique name using [UUID](#).

Python

```
import uuid

# Creating a unique name for the endpoint
online_endpoint_name = "credit-endpoint-" + str(uuid.uuid4())[:8]
```

ⓘ Note

Expect the endpoint creation to take approximately 6 to 8 minutes.

Python

```
from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
)

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is an online endpoint",
    auth_mode="key",
    tags={
        "training_dataset": "credit_defaults",
        "model_type": "sklearn.GradientBoostingClassifier",
    },
)

endpoint =
ml_client.online_endpoints.begin_create_or_update(endpoint).result()
```

```
print(f"Endpoint {endpoint.name} provisioning state:  
{endpoint.provisioning_state}")
```

Once you've created an endpoint, you can retrieve it as below:

Python

```
endpoint = ml_client.online_endpoints.get(name=online_endpoint_name)  
  
print(  
    f'Endpoint "{endpoint.name}" with provisioning state "  
{endpoint.provisioning_state}" is retrieved'  
)
```

Deploy the model to the endpoint

Once the endpoint is created, deploy the model with the entry script. Each endpoint can have multiple deployments. Direct traffic to these deployments can be specified using rules. Here you'll create a single deployment that handles 100% of the incoming traffic. We have chosen a color name for the deployment, for example, *blue*, *green*, *red* deployments, which is arbitrary.

You can check the **Models** page on the Azure Machine Learning studio, to identify the latest version of your registered model. Alternatively, the code below will retrieve the latest version number for you to use.

Python

```
# Let's pick the latest version of the model  
latest_model_version = max(  
    [int(m.version) for m in  
     ml_client.models.list(name=registered_model_name)]  
)
```

Deploy the latest version of the model.

 **Note**

Expect this deployment to take approximately 6 to 8 minutes.

Python

```

# picking the model to deploy. Here we use the latest version of our
# registered model
model = ml_client.models.get(name=registered_model_name,
version=latest_model_version)

# create an online deployment.
blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    instance_type="Standard_DS3_v2",
    instance_count=1,
)
blue_deployment = ml_client.begin_create_or_update(blue_deployment).result()

```

Test with a sample query

Now that the model is deployed to the endpoint, you can run inference with it.

Create a sample request file following the design expected in the run method in the score script.

Python

```

deploy_dir = "./deploy"
os.makedirs(deploy_dir, exist_ok=True)

```

Python

```

%%writefile {deploy_dir}/sample-request.json
{
  "input_data": {
    "columns": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22],
    "index": [0, 1],
    "data": [
      [20000,2,2,1,24,2,2,-1,-1,-2,3913,3102,689,0,0,0,0,689,0,0,0,0],
      [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
       10, 9, 8]
    ]
  }
}

```

Python

```
# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    request_file="./deploy/sample-request.json",
    deployment_name="blue",
)
```

Clean up resources

If you're not going to use the endpoint, delete it to stop using the resource. Make sure no other deployments are using an endpoint before you delete it.

 **Note**

Expect this step to take approximately 6 to 8 minutes.

Python

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Delete everything

Use these steps to delete your Azure Machine Learning workspace and all compute resources.

 **Important**

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'FAVORITES' section with links like Home, Dashboard, All services, Resource groups, App Services, SQL databases, SQL data warehouses, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, and Virtual networks. The main content area is titled 'Resource groups > my-rg'. It shows an 'Overview' tab selected. Below it are tabs for Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Deployments, Policies, Properties, Locks, and Export template. On the right, there's a summary card for 'Subscription (change)' showing 'Visual Studio Ultimate with MSDN' and 'Subscription ID: xxxxx-xxx-xxxx-xxxx'. Below that are sections for Tags, Deployments (1 Succeeded), and a list of resources. A red box highlights the 'Delete resource group' button in the top right of the main content area.

4. Enter the resource group name. Then select **Delete**.

Next steps

- Convert this tutorial into a production ready [pipeline with reusable components](#).
- Learn about all of the [deployment options](#) for Azure Machine Learning.
- Learn how to [authenticate to the deployed model](#).

Additional resources

Documentation

[Quickstart: Interactive Data Wrangling with Apache Spark \(preview\) - Azure Machine Learning](#)

Learn how to perform interactive data wrangling with Apache Spark in Azure Machine Learning

[Train deep learning Keras models \(SDK v2\) - Azure Machine Learning](#)

Learn how to train and register a Keras deep neural network classification model running on TensorFlow using Azure Machine Learning SDK (v2).

[Build & train models - Azure Machine Learning](#)

Learn how to train models with Azure Machine Learning. Explore the different training methods and choose the right one for your project.

[Quickstart: Run notebooks - Azure Machine Learning](#)

Learn to run Jupyter notebooks in studio, and find sample notebooks to learn more about Azure

Machine Learning.

[Assess AI systems and make data-driven decisions with Azure Machine Learning Responsible AI dashboard - Azure Machine Learning](#)

Learn how to use the comprehensive UI and SDK/YAML components in the Responsible AI dashboard to debug your machine learning models and make data-driven decisions.

[What is distributed training? - Azure Machine Learning](#)

Learn what type of distributed training Azure Machine Learning supports and the open source framework integrations available for distributed training.

[How to view AutoML model training code - Azure Machine Learning AutoML](#)

How to view model training code for an automated ML trained model and explanation of each stage.

[Train deep learning PyTorch models \(SDK v2\) - Azure Machine Learning](#)

Learn how to run your PyTorch training scripts at enterprise scale using Azure Machine Learning SDK (v2).

[Show 5 more](#)

Training

Learning paths and modules

[Use AutoML to train a labeled dataset and develop a production model - Training](#)

Use AutoML to train a labeled dataset and develop a production model.

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Tutorial: Create production ML pipelines with Python SDK v2 in a Jupyter notebook

Article • 02/24/2023 • 17 minutes to read

APPLIES TO:  Python SDK `azure-ai-ml v2 (current)` ↗

Note

For a tutorial that uses SDK v1 to build a pipeline, see [Tutorial: Build an Azure Machine Learning pipeline for image classification](#)

In this tutorial, you'll use Azure Machine Learning to create a production ready machine learning (ML) project, using Azure Machine Learning Python SDK v2.

You'll learn how to use the Azure Machine Learning Python SDK v2 to:

- ✓ Connect to your Azure Machine Learning workspace
- ✓ Create Azure Machine Learning data assets
- ✓ Create reusable Azure Machine Learning components
- ✓ Create, validate and run Azure Machine Learning pipelines
- ✓ Deploy the newly-trained model as an endpoint
- ✓ Call the Azure Machine Learning endpoint for inferencing

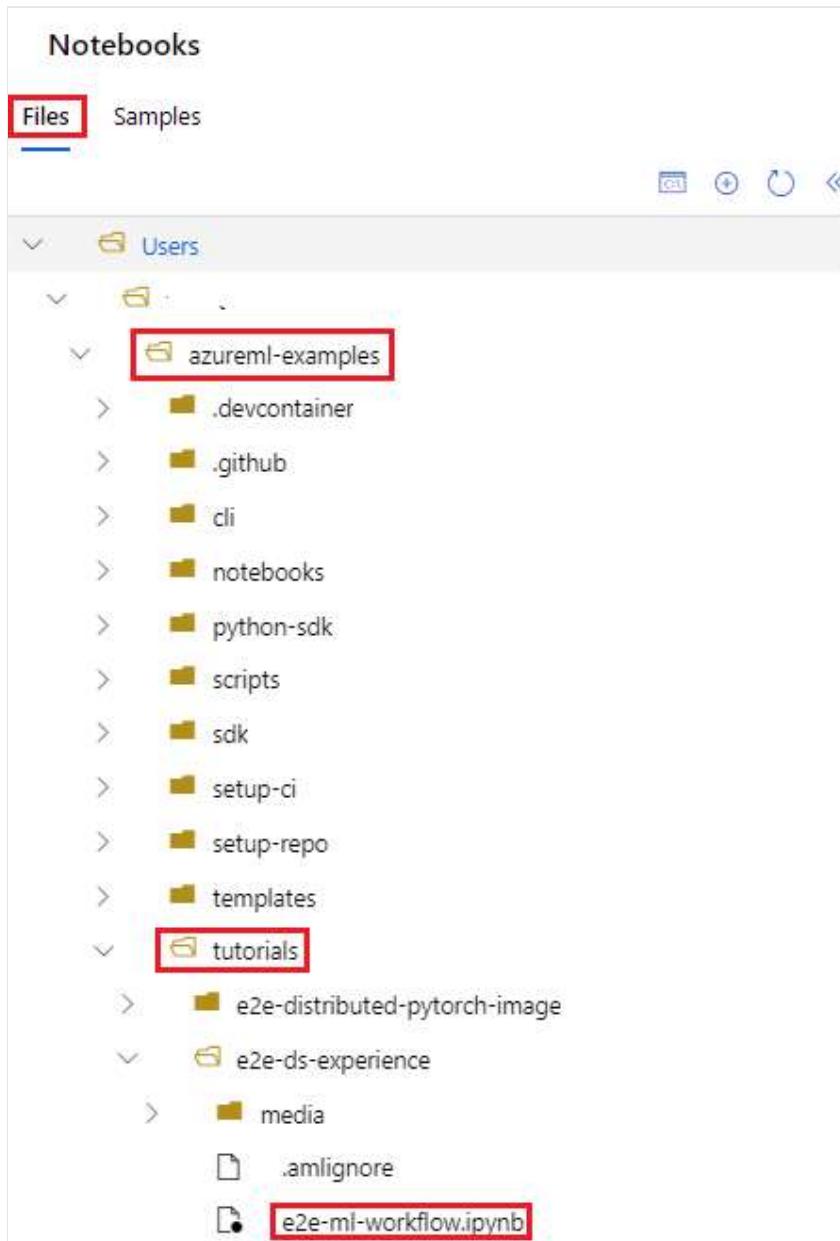
Prerequisites

- Complete the [Quickstart: Get started with Azure Machine Learning](#) to:
 - Create a workspace.
 - Create a cloud-based compute instance to use for your development environment.
 - Create a cloud-based compute cluster to use for training your model.
- Complete the [Quickstart: Run Jupyter notebooks in studio](#) to clone the `SDK v2/tutorials` folder.

Open the notebook

1. Open the `tutorials` folder that was cloned into your `Files` section from the [Quickstart: Run Jupyter notebooks in studio](#).

2. Select the `e2e-ml-workflow.ipynb` file from your `tutorials/azureml-examples/tutorials/e2e-ds-experience/` folder.



3. On the top bar, select the compute instance you created during the [Quickstart: Get started with Azure Machine Learning](#) to use for running the notebook.

ⓘ Important

The rest of this article contains the same content as you see in the notebook.

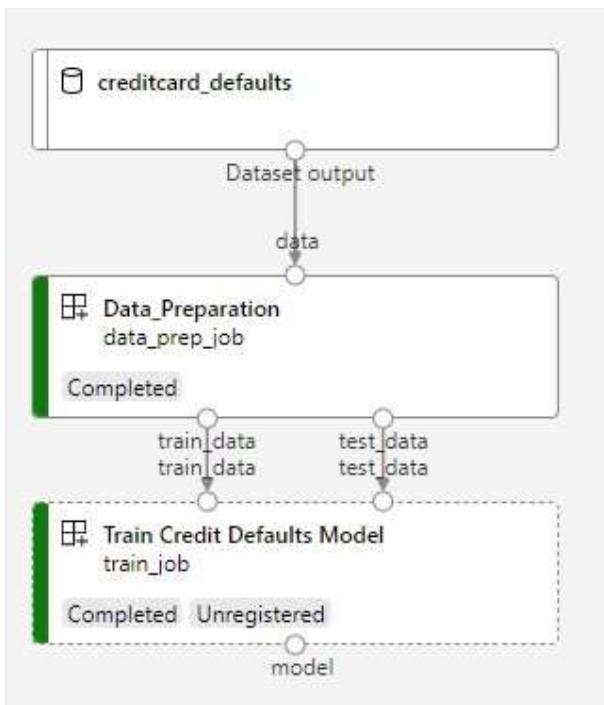
Switch to the Jupyter Notebook now if you want to run the code while you read along. To run a single code cell in a notebook, click the code cell and hit **Shift+Enter**. Or, run the entire notebook by choosing **Run all** from the top toolbar

Introduction

In this tutorial, you'll create an Azure Machine Learning pipeline to train a model for credit default prediction. The pipeline handles the data preparation, training and registering the trained model. You'll then run the pipeline, deploy the model and use it.

The image below shows the pipeline as you'll see it in the Azure Machine Learning portal once submitted. It's a rather simple pipeline we'll use to walk you through the Azure Machine Learning SDK v2.

The two steps are first data preparation and second training.



Set up the pipeline resources

The Azure Machine Learning framework can be used from CLI, Python SDK, or studio interface. In this example, you'll use the Azure Machine Learning Python SDK v2 to create a pipeline.

Before creating the pipeline, you'll set up the resources the pipeline will use:

- The data asset for training
- The software environment to run the pipeline
- A compute resource to where the job will run

Connect to the workspace

Before we dive in the code, you'll need to connect to your Azure Machine Learning workspace. The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning.

Python

```
# Handle to the workspace
from azure.ai.ml import MLClient

# Authentication package
from azure.identity import DefaultAzureCredential,
InteractiveBrowserCredential

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case
DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()
```

In the next cell, enter your Subscription ID, Resource Group name and Workspace name. To find your Subscription ID:

1. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
2. You'll see the values you need for <SUBSCRIPTION_ID>, <RESOURCE_GROUP>, and <AML_WORKSPACE_NAME>.
3. Copy a value, then close the window and paste that into your code. Open the tool again to get the next value.

The screenshot shows the Azure ML workspace configuration interface. At the top, there are icons for notifications (3), gear, question mark, and user profile. The workspace name "ML-docs my-workspace" is displayed with a dropdown arrow. Below this, the title "Directory + Subscription + Workspace" is shown, followed by "X". Under "Current directory", the value "Microsoft" is selected from a dropdown. Under "Current subscription", the value "ML-docs" is selected from a dropdown. Under "Current workspace", the value "my-workspace" is selected from a dropdown, which is highlighted with a red box. In the bottom section, there are fields for "Resource Group" (sgilley-rg), "Subscription ID" (xxxx-xxxx-xxxxxx-xxxx-xxxx-xxxx-xxxxxx-x8), and "Location" (eastus2). There are also links for "Download config file" and "View all properties in Azure Portal".

Python

```
# Get a handle to the workspace
ml_client = MLClient(
    credential=credential,
    subscription_id=<SUBSCRIPTION_ID>,
    resource_group_name=<RESOURCE_GROUP>,
    workspace_name=<AML_WORKSPACE_NAME>,
)
```

The result is a handler to the workspace that you'll use to manage other resources and jobs.

Important

Creating `MLClient` will not connect to the workspace. The client initialization is lazy, it will wait for the first time it needs to make a call (in the notebook below, that will

happen during dataset registration).

Register data from an external url

The data you use for training is usually in one of the locations below:

- Local machine
- Web
- Big Data Storage services (for example, Azure Blob, Azure Data Lake Storage, SQL)

Azure Machine Learning uses a `Data` object to register a reusable definition of data, and consume data within a pipeline. In the section below, you'll consume some data from web url as one example. Data from other sources can be created as well. `Data` assets from other sources can be created as well.

Python

```
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

web_path = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00350/default%20of%20credit%20card%20clients.xls"

credit_data = Data(
    name="creditcard_defaults",
    path=web_path,
    type=AssetTypes.URI_FILE,
    description="Dataset for credit card defaults",
    tags={"source_type": "web", "source": "UCI ML Repo"},
    version="1.0.0",
)
```

This code just created a `Data` asset, ready to be consumed as an input by the pipeline that you'll define in the next sections. In addition, you can register the data to your workspace so it becomes reusable across pipelines.

Registering the data asset will enable you to:

- Reuse and share the data asset in future pipelines
- Use versions to track the modification to the data asset
- Use the data asset from Azure Machine Learning designer, which is Azure Machine Learning's GUI for pipeline authoring

Since this is the first time that you're making a call to the workspace, you may be asked to authenticate. Once the authentication is complete, you'll then see the dataset

registration completion message.

Python

```
credit_data = ml_client.data.create_or_update(credit_data)
print(
    f"Dataset with name {credit_data.name} was registered to workspace, the
dataset version is {credit_data.version}"
)
```

In the future, you can fetch the same dataset from the workspace using `credit_dataset = ml_client.data.get("<DATA ASSET NAME>", version='<VERSION>').`

Create a compute resource to run your pipeline

Each step of an Azure Machine Learning pipeline can use a different compute resource for running the specific job of that step. It can be single or multi-node machines with Linux or Windows OS, or a specific compute fabric like Spark.

In this section, you'll provision a Linux [compute cluster](#). See the [full list on VM sizes and prices ↗](#).

For this tutorial you only need a basic cluster, so we'll use a Standard_DS3_v2 model with 2 vCPU cores, 7 GB RAM and create an Azure Machine Learning Compute.

💡 Tip

If you already have a compute cluster, replace "cpu-cluster" in the code below with the name of your cluster. This will keep you from creating another one.

Python

```
from azure.ai.ml.entities import AmlCompute

cpu_compute_target = "cpu-cluster"

try:
    # let's see if the compute target already exists
    cpu_cluster = ml_client.compute.get(cpu_compute_target)
    print(
        f"You already have a cluster named {cpu_compute_target}, we'll reuse
it as is."
    )

except Exception:
    print("Creating a new cpu compute target...")
```

```

# Let's create the Azure ML compute object with the intended parameters
cpu_cluster = AmlCompute(
    # Name assigned to the compute cluster
    name="cpu-cluster",
    # Azure ML Compute is the on-demand VM service
    type="amlcompute",
    # VM Family
    size="STANDARD_DS3_V2",
    # Minimum running nodes when there is no job running
    min_instances=0,
    # Nodes in cluster
    max_instances=4,
    # How many seconds will the node running after the job termination
    idle_time_before_scale_down=180,
    # Dedicated or LowPriority. The latter is cheaper but there is a
    chance of job termination
    tier="Dedicated",
)

# Now, we pass the object to MLClient's create_or_update method
cpu_cluster = ml_client.begin_create_or_update(cpu_cluster)

print(
    f"AMLCompute with name {cpu_cluster.name} is created, the compute size
is {cpu_cluster.size}"
)

```

Create a job environment for pipeline steps

So far, you've created a development environment on the compute instance, your development machine. You'll also need an environment to use for each step of the pipeline. Each step can have its own environment, or you can use some common environments for multiple steps.

In this example, you'll create a conda environment for your jobs, using a conda yaml file. First, create a directory to store the file in.

Python

```

import os

dependencies_dir = "./dependencies"
os.makedirs(dependencies_dir, exist_ok=True)

```

Now, create the file in the dependencies directory.

Python

```
%%writefile {dependencies_dir}/conda.yml
name: model-env
channels:
- conda-forge
dependencies:
- python=3.8
- numpy=1.21.2
- pip=21.2.4
- scikit-learn=0.24.2
- scipy=1.7.1
- pandas>=1.1,<1.2
- pip:
  - inference-schema[numpy-support]==1.3.0
  - xlrd==2.0.1
  - mlflow== 1.26.1
  - azureml-mlflow==1.42.0
```

The specification contains some usual packages, that you'll use in your pipeline (numpy, pip), together with some Azure Machine Learning specific packages (azureml-defaults, azureml-mlflow).

The Azure Machine Learning packages aren't mandatory to run Azure Machine Learning jobs. However, adding these packages will let you interact with Azure Machine Learning for logging metrics and registering models, all inside the Azure Machine Learning job. You'll use them in the training script later in this tutorial.

Use the *yaml* file to create and register this custom environment in your workspace:

Python

```
from azure.ai.ml.entities import Environment

custom_env_name = "aml-scikit-learn"

pipeline_job_env = Environment(
    name=custom_env_name,
    description="Custom environment for Credit Card Defaults pipeline",
    tags={"scikit-learn": "0.24.2"},
    conda_file=os.path.join(dependencies_dir, "conda.yml"),
    image="mcr.microsoft.com/azureml/openmpi4.1.0-ubuntu20.04:latest",
    version="0.1.0",
)
pipeline_job_env = ml_client.environments.create_or_update(pipeline_job_env)

print(
    f"Environment with name {pipeline_job_env.name} is registered to
workspace, the environment version is {pipeline_job_env.version}"
)
```

Build the training pipeline

Now that you have all assets required to run your pipeline, it's time to build the pipeline itself, using the Azure Machine Learning Python SDK v2.

Azure Machine Learning pipelines are reusable ML workflows that usually consist of several components. The typical life of a component is:

- Write the yaml specification of the component, or create it programmatically using `ComponentMethod`.
- Optionally, register the component with a name and version in your workspace, to make it reusable and shareable.
- Load that component from the pipeline code.
- Implement the pipeline using the component's inputs, outputs and parameters
- Submit the pipeline.

Create component 1: data prep (using programmatic definition)

Let's start by creating the first component. This component handles the preprocessing of the data. The preprocessing task is performed in the `data_prep.py` Python file.

First create a source folder for the `data_prep` component:

```
Python

import os

data_prep_src_dir = "./components/data_prep"
os.makedirs(data_prep_src_dir, exist_ok=True)
```

This script performs the simple task of splitting the data into train and test datasets. Azure Machine Learning mounts datasets as folders to the computes, therefore, we created an auxiliary `select_first_file` function to access the data file inside the mounted input folder.

[MLFlow](#) will be used to log the parameters and metrics during our pipeline run.

```
Python

%%writefile {data_prep_src_dir}/data_prep.py
import os
import argparse
import pandas as pd
```

```

from sklearn.model_selection import train_test_split
import logging
import mlflow

def main():
    """Main function of the script."""

    # input and output arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, help="path to input data")
    parser.add_argument("--test_train_ratio", type=float, required=False,
    default=0.25)
    parser.add_argument("--train_data", type=str, help="path to train data")
    parser.add_argument("--test_data", type=str, help="path to test data")
    args = parser.parse_args()

    # Start Logging
    mlflow.start_run()

    print(" ".join(f"{k}={v}" for k, v in vars(args).items()))

    print("input data:", args.data)

    credit_df = pd.read_excel(args.data, header=1, index_col=0)

    mlflow.log_metric("num_samples", credit_df.shape[0])
    mlflow.log_metric("num_features", credit_df.shape[1] - 1)

    credit_train_df, credit_test_df = train_test_split(
        credit_df,
        test_size=args.test_train_ratio,
    )

    # output paths are mounted as folder, therefore, we are adding a
    # filename to the path
    credit_train_df.to_csv(os.path.join(args.train_data, "data.csv"),
    index=False)

    credit_test_df.to_csv(os.path.join(args.test_data, "data.csv"),
    index=False)

    # Stop Logging
    mlflow.end_run()

if __name__ == "__main__":
    main()

```

Now that you have a script that can perform the desired task, create an Azure Machine Learning Component from it.

You'll use the general purpose `CommandComponent` that can run command line actions. This command line action can directly call system commands or run a script. The inputs/outputs are specified on the command line via the `${{ ... }}` notation.

Python

```
from azure.ai.ml import command
from azure.ai.ml import Input, Output

data_prep_component = command(
    name="data_prep_credit_defaults",
    display_name="Data preparation for training",
    description="reads a .xl input, split the input to train and test",
    inputs={
        "data": Input(type="uri_folder"),
        "test_train_ratio": Input(type="number"),
    },
    outputs=dict(
        train_data=Output(type="uri_folder", mode="rw_mount"),
        test_data=Output(type="uri_folder", mode="rw_mount"),
    ),
    # The source folder of the component
    code=data_prep_src_dir,
    command="""python data_prep.py \
        --data ${{inputs.data}} --test_train_ratio
${{inputs.test_train_ratio}} \
        --train_data ${{outputs.train_data}} --test_data
${{outputs.test_data}} \
        """,
    environment=f"{pipeline_job_env.name}:{pipeline_job_env.version}",
)
```

Optionally, register the component in the workspace for future re-use.

Create component 2: training (using yaml definition)

The second component that you'll create will consume the training and test data, train a tree based model and return the output model. You'll use Azure Machine Learning logging capabilities to record and visualize the learning progress.

You used the `CommandComponent` class to create your first component. This time you'll use the yaml definition to define the second component. Each method has its own advantages. A yaml definition can actually be checked-in along the code, and would provide a readable history tracking. The programmatic method using `CommandComponent` can be easier with built-in class documentation and code completion.

Create the directory for this component:

Python

```
import os

train_src_dir = "./components/train"
os.makedirs(train_src_dir, exist_ok=True)
```

Create the training script in the directory:

Python

```
%%writefile {train_src_dir}/train.py
import argparse
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
import os
import pandas as pd
import mlflow

def select_first_file(path):
    """Selects first file in folder, use under assumption there is only one
    file in folder
    Args:
        path (str): path to directory or file to choose
    Returns:
        str: full path of selected file
    """
    files = os.listdir(path)
    return os.path.join(path, files[0])

# Start Logging
mlflow.start_run()

# enable autologging
mlflow.sklearn.autolog()

os.makedirs("./outputs", exist_ok=True)

def main():
    """Main function of the script."""

    # input and output arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--train_data", type=str, help="path to train data")
    parser.add_argument("--test_data", type=str, help="path to test data")
    parser.add_argument("--n_estimators", required=False, default=100,
    type=int)
```

```

parser.add_argument("--learning_rate", required=False, default=0.1,
type=float)
parser.add_argument("--registered_model_name", type=str, help="model
name")
parser.add_argument("--model", type=str, help="path to model file")
args = parser.parse_args()

# paths are mounted as folder, therefore, we are selecting the file from
folder
train_df = pd.read_csv(select_first_file(args.train_data))

# Extracting the label column
y_train = train_df.pop("default payment next month")

# convert the dataframe values to array
X_train = train_df.values

# paths are mounted as folder, therefore, we are selecting the file from
folder
test_df = pd.read_csv(select_first_file(args.test_data))

# Extracting the label column
y_test = test_df.pop("default payment next month")

# convert the dataframe values to array
X_test = test_df.values

print(f"Training with data of shape {X_train.shape}")

clf = GradientBoostingClassifier(
    n_estimators=args.n_estimators, learning_rate=args.learning_rate
)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))

# Registering the model to the workspace
print("Registering the model via MLFlow")
mlflow.sklearn.log_model(
    sk_model=clf,
    registered_model_name=args.registered_model_name,
    artifact_path=args.registered_model_name,
)

# Saving the model to a file
mlflow.sklearn.save_model(
    sk_model=clf,
    path=os.path.join(args.model, "trained_model"),
)

# Stop Logging
mlflow.end_run()

```

```
if __name__ == "__main__":
    main()
```

As you can see in this training script, once the model is trained, the model file is saved and registered to the workspace. Now you can use the registered model in inferencing endpoints.

For the environment of this step, you'll use one of the built-in (curated) Azure Machine Learning environments. The tag `azureml`, tells the system to use look for the name in curated environments.

First, create the *yaml* file describing the component:

Python

```
%>writefile {train_src_dir}/train.yml
# <component>
name: train_credit_defaults_model
display_name: Train Credit Defaults Model
# version: 1 # Not specifying a version will automatically update the
version
type: command
inputs:
    train_data:
        type: uri_folder
    test_data:
        type: uri_folder
    learning_rate:
        type: number
    registered_model_name:
        type: string
outputs:
    model:
        type: uri_folder
code: .
environment:
    # for this step, we'll use an AzureML curate environment
    azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu:1
command: >-
    python train.py
    --train_data ${inputs.train_data}
    --test_data ${inputs.test_data}
    --learning_rate ${inputs.learning_rate}
    --registered_model_name ${inputs.registered_model_name}
    --model ${outputs.model}
# </component>
```

Now create and register the component:

Python

```
# importing the Component Package
from azure.ai.ml import load_component

# Loading the component from the yml file
train_component = load_component(source=os.path.join(train_src_dir,
"train.yml"))
```

Python

```
# Now we register the component to the workspace
train_component = ml_client.create_or_update(train_component)

# Create (register) the component in your workspace
print(
    f"Component {train_component.name} with Version
{train_component.version} is registered"
)
```

Create the pipeline from components

Now that both your components are defined and registered, you can start implementing the pipeline.

Here, you'll use *input data*, *split ratio* and *registered model name* as input variables. Then call the components and connect them via their inputs/outputs identifiers. The outputs of each step can be accessed via the `.outputs` property.

The Python functions returned by `load_component()` work as any regular Python function that we'll use within a pipeline to call each step.

To code the pipeline, you use a specific `@dsl.pipeline` decorator that identifies the Azure Machine Learning pipelines. In the decorator, we can specify the pipeline description and default resources like compute and storage. Like a Python function, pipelines can have inputs. You can then create multiple instances of a single pipeline with different inputs.

Here, we used *input data*, *split ratio* and *registered model name* as input variables. We then call the components and connect them via their inputs/outputs identifiers. The outputs of each step can be accessed via the `.outputs` property.

Python

```

# the dsl decorator tells the sdk that we are defining an Azure ML pipeline
from azure.ai.ml import dsl, Input, Output

@dsl.pipeline(
    compute=cpu_compute_target,
    description="E2E data_perp-train pipeline",
)
def credit_defaults_pipeline(
    pipeline_job_data_input,
    pipeline_job_test_train_ratio,
    pipeline_job_learning_rate,
    pipeline_job_registered_model_name,
):
    # using data_prep_function like a python call with its own inputs
    data_prep_job = data_prep_component(
        data=pipeline_job_data_input,
        test_train_ratio=pipeline_job_test_train_ratio,
    )

    # using train_func like a python call with its own inputs
    train_job = train_component(
        train_data=data_prep_job.outputs.train_data, # note: using outputs
        from previous step
        test_data=data_prep_job.outputs.test_data, # note: using outputs
        from previous step
        learning_rate=pipeline_job_learning_rate, # note: using a pipeline
        input as parameter
        registered_model_name=pipeline_job_registered_model_name,
    )

    # a pipeline returns a dictionary of outputs
    # keys will code for the pipeline output identifier
    return {
        "pipeline_job_train_data": data_prep_job.outputs.train_data,
        "pipeline_job_test_data": data_prep_job.outputs.test_data,
    }

```

Now use your pipeline definition to instantiate a pipeline with your dataset, split rate of choice and the name you picked for your model.

Python

```

registered_model_name = "credit_defaults_model"

# Let's instantiate the pipeline with the parameters of our choice
pipeline = credit_defaults_pipeline(
    pipeline_job_data_input=Input(type="uri_file", path=credit_data.path),
    pipeline_job_test_train_ratio=0.25,
    pipeline_job_learning_rate=0.05,
    pipeline_job_registered_model_name=registered_model_name,
)

```

Submit the job

It's now time to submit the job to run in Azure Machine Learning. This time you'll use `create_or_update` on `ml_client.jobs`.

Here you'll also pass an experiment name. An experiment is a container for all the iterations one does on a certain project. All the jobs submitted under the same experiment name would be listed next to each other in Azure Machine Learning studio.

Once completed, the pipeline will register a model in your workspace as a result of training.

```
Python

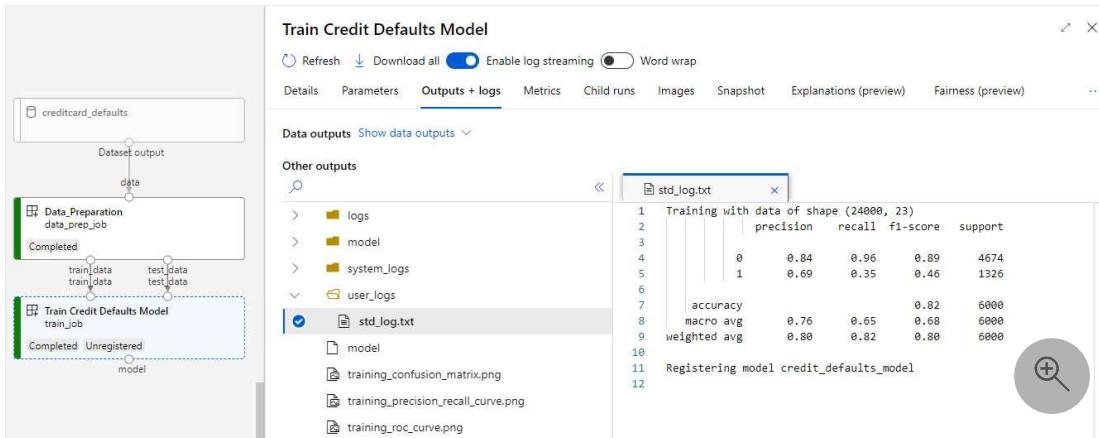
import webbrowser

# submit the pipeline job
pipeline_job = ml_client.jobs.create_or_update(
    pipeline,
    # Project's name
    experiment_name="e2e_registered_components",
)
# open the pipeline in web browser
webbrowser.open(pipeline_job.studio_url)
```

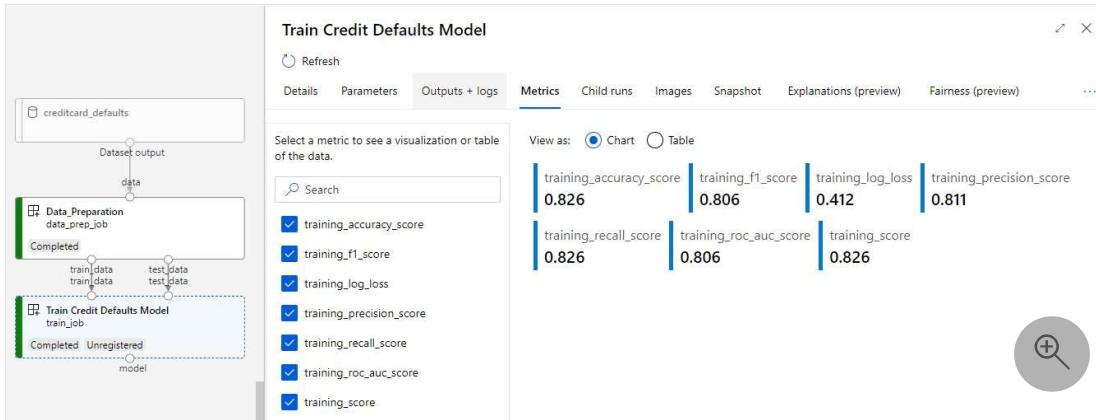
An output of "False" is expected from the above cell. You can track the progress of your pipeline, by using the link generated in the cell above.

When you select on each component, you'll see more information about the results of that component. There are two important parts to look for at this stage:

- `Outputs+logs > user_logs > std_log.txt` This section shows the script run sdtout.



- **Outputs+logs > Metric** This section shows different logged metrics. In this example, mlflow autologging, has automatically logged the training metrics.



Deploy the model as an online endpoint

Now deploy your machine learning model as a web service in the Azure cloud, an [online endpoint](#). To deploy a machine learning service, you usually need:

- The model assets (filed, metadata) that you want to deploy. You've already registered these assets in your training component.
- Some code to run as a service. The code executes the model on a given input request. This entry script receives data submitted to a deployed web service and passes it to the model, then returns the model's response to the client. The script is specific to your model. The entry script must understand the data that the model expects and returns. When using a MLflow model, as in this tutorial, this script is automatically created for you

Create a new online endpoint

Now that you have a registered model and an inference script, it's time to create your online endpoint. The endpoint name needs to be unique in the entire Azure region. For this tutorial, you'll create a unique name using [UUID](#).

```
Python

import uuid

# Creating a unique name for the endpoint
online_endpoint_name = "credit-endpoint-" + str(uuid.uuid4())[:8]
```

```
Python
```

```

from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
)

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is an online endpoint",
    auth_mode="key",
    tags={
        "training_dataset": "credit_defaults",
        "model_type": "sklearn.GradientBoostingClassifier",
    },
)
endpoint_result = ml_client.begin_create_or_update(endpoint).result()

print(
    f"Endpoint {endpoint_result.name} provisioning state: "
    f"{endpoint_result.provisioning_state}"
)

```

Once you've created an endpoint, you can retrieve it as below:

Python

```

endpoint = ml_client.online_endpoints.get(name=online_endpoint_name)

print(
    f'Endpoint "{endpoint.name}" with provisioning state "'
    f'{endpoint.provisioning_state}" is retrieved'
)

```

Deploy the model to the endpoint

Once the endpoint is created, deploy the model with the entry script. Each endpoint can have multiple deployments and direct traffic to these deployments can be specified using rules. Here you'll create a single deployment that handles 100% of the incoming traffic. We have chosen a color name for the deployment, for example, *blue*, *green*, *red* deployments, which is arbitrary.

You can check the *Models* page on the Azure Machine Learning studio, to identify the latest version of your registered model. Alternatively, the code below will retrieve the latest version number for you to use.

Python

```
# Let's pick the latest version of the model
latest_model_version = max(
    [int(m.version) for m in
     ml_client.models.list(name=registered_model_name)]
)
```

Deploy the latest version of the model.

ⓘ Note

Expect this deployment to take approximately 6 to 8 minutes.

Python

```
# picking the model to deploy. Here we use the latest version of our
# registered model
model = ml_client.models.get(name=registered_model_name,
                             version=latest_model_version)

# create an online deployment.
blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    instance_type="Standard_F4s_v2",
    instance_count=1,
)

blue_deployment_results =
ml_client.online_deployments.begin_create_or_update(
    blue_deployment
).result()

print(
    f"Deployment {blue_deployment_results.name} provisioning state:
{blue_deployment_results.provisioning_state}"
)
```

Test with a sample query

Now that the model is deployed to the endpoint, you can run inference with it.

Create a sample request file following the design expected in the run method in the score script.

```
Python
```

```
deploy_dir = "./deploy"
os.makedirs(deploy_dir, exist_ok=True)
```

```
Python
```

```
%%writefile {deploy_dir}/sample-request.json
{
    "input_data": {
        "columns": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22],
        "index": [0, 1],
        "data": [
            [20000,2,2,1,24,2,2,-1,-1,-2,-2,3913,3102,689,0,0,0,0,689,0,0,0,0],
            [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
            10, 9, 8]
        ]
    }
}
```

```
Python
```

```
# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    request_file="./deploy/sample-request.json",
    deployment_name="blue",
)
```

Clean up resources

If you're not going to use the endpoint, delete it to stop using the resource. Make sure no other deployments are using an endpoint before you delete it.

ⓘ Note

Expect this step to take approximately 6 to 8 minutes.

```
Python
```

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Next steps

Learn more about

[Azure Machine Learning logging](#)

Additional resources

Documentation

[Tutorial: Upload data and train a model \(SDK v1\) - Azure Machine Learning](#)

How to upload and use your own data in a remote training job, with SDK v1. This is part 3 of a three-part getting-started series.

[Hyperparameter tuning a model \(v1\) - Azure Machine Learning](#)

Automate hyperparameter tuning for deep learning and machine learning models using Azure Machine Learning.(v1)

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Create and explore datasets with labels - Azure Machine Learning](#)

Learn how to export data labels from your Azure Machine Learning labeling projects and use them for machine learning tasks.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Use automated ML in ML pipelines - Azure Machine Learning](#)

The AutoMLStep allows you to use automated machine learning in your pipelines.

[How to do hyperparameter sweep in pipeline - Azure Machine Learning](#)

How to use sweep to do hyperparameter tuning in Azure Machine Learning pipeline using CLI v2 and Python SDK

[Show 5 more](#)

Training

Learning paths and modules

[Define and implement continuous integration - Training](#)

Create automated pipelines that continuously build, test, and deploy your applications and prepare for Exam AZ-400: Designing and Implementing Microsoft DevOps Solutions.

Learning certificate

Microsoft Certified: Azure Data Scientist Associate - Certifications

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Tutorial: Train an object detection model with AutoML and Python

Article • 02/24/2023 • 17 minutes to read

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) [🔗](#)

In this tutorial, you learn how to train an object detection model using Azure Machine Learning automated ML with the Azure Machine Learning CLI extension v2 or the Azure Machine Learning Python SDK v2. This object detection model identifies whether the image contains objects, such as a can, carton, milk bottle, or water bottle.

Automated ML accepts training data and configuration settings, and automatically iterates through combinations of different feature normalization/standardization methods, models, and hyperparameter settings to arrive at the best model.

You'll write code using the Python SDK in this tutorial and learn the following tasks:

-  Download and transform data
-  Train an automated machine learning object detection model
-  Specify hyperparameter values for your model
-  Perform a hyperparameter sweep
-  Deploy your model
-  Visualize detections

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version !\[\]\(2a06d23bba00fd4875cb69b639e36cef_img.jpg\)](#) of Azure Machine Learning today.
- Python 3.6 or 3.7 are supported for this feature
- Complete the [Quickstart: Get started with Azure Machine Learning](#) if you don't already have an Azure Machine Learning workspace.
- Download and unzip the [*odFridgeObjects.zip !\[\]\(3ebfc1501b74ab9d419c0b564d61aad1_img.jpg\)](#) data file. The dataset is annotated in Pascal VOC format, where each image corresponds to an xml file. Each xml file contains information on where its corresponding image file is located and also contains information about the bounding boxes and the object labels. In order to use this data, you first need to convert it to the required JSONL format as seen in the [Convert the downloaded data to JSONL !\[\]\(ff96ad93d3eae1c7ec5212a14ae07064_img.jpg\)](#) section of the notebook.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

This tutorial is also available in the [azureml-examples repository on GitHub](#). If you wish to run it in your own local environment, setup using the following instructions

- Install and [set up CLI \(v2\)](#) and make sure you install the `ml` extension.

Compute target setup

You first need to set up a compute target to use for your automated ML model training. Automated ML models for image tasks require GPU SKUs.

This tutorial uses the NCsv3-series (with V100 GPUs) as this type of compute target leverages multiple GPUs to speed up training. Additionally, you can set up multiple nodes to take advantage of parallelism when tuning hyperparameters for your model.

The following code creates a GPU compute of size `Standard_NC24s_v3` with four nodes.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

Create a .yml file with the following configuration.

yml

```
$schema:  
  https://azuremlschemas.azureedge.net/latest/amlCompute.schema.json  
name: gpu-cluster  
type: amlcompute  
size: Standard_NC24s_v3  
min_instances: 0  
max_instances: 4  
idle_time_before_scale_down: 120
```

To create the compute, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

Azure CLI

```
az ml compute create -f [PATH_TO_YML_FILE] --workspace-name  
[YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --
```

```
subscription [YOUR_AZURE_SUBSCRIPTION]
```

The created compute can be provided using `compute` key in the `automl` task configuration yaml:

YAML

```
compute: azureml:gpu-cluster
```

Experiment setup

You can use an Experiment to track your model training jobs.

Azure CLI

APPLIES TO: Azure CLI ml extension v2 (current)

Experiment name can be provided using `experiment_name` key as follows:

YAML

```
experiment_name: dpv2-cli-automl-image-object-detection-experiment
```

Visualize input data

Once you have the input image data prepared in [JSONL](#) (JSON Lines) format, you can visualize the ground truth bounding boxes for an image. To do so, be sure you have `matplotlib` installed.

```
%pip install --upgrade matplotlib
```

Python

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.patches as patches
from PIL import Image as pil_image
```

```

import numpy as np
import json
import os

def plot_ground_truth_boxes(image_file, ground_truth_boxes):
    # Display the image
    plt.figure()
    img_np = mpimg.imread(image_file)
    img = pil_image.fromarray(img_np.astype("uint8"), "RGB")
    img_w, img_h = img.size

    fig,ax = plt.subplots(figsize=(12, 16))
    ax.imshow(img_np)
    ax.axis("off")

    label_to_color_mapping = {}

    for gt in ground_truth_boxes:
        label = gt["label"]

        xmin, ymin, xmax, ymax = gt["topX"], gt["topY"], gt["bottomX"],
        gt["bottomY"]
        topleft_x, topleft_y = img_w * xmin, img_h * ymin
        width, height = img_w * (xmax - xmin), img_h * (ymax - ymin)

        if label in label_to_color_mapping:
            color = label_to_color_mapping[label]
        else:
            # Generate a random color. If you want to use a specific color,
            # you can use something like "red".
            color = np.random.rand(3)
            label_to_color_mapping[label] = color

        # Display bounding box
        rect = patches.Rectangle((topleft_x, topleft_y), width, height,
                                linewidth=2, edgecolor=color,
                                facecolor="none")
        ax.add_patch(rect)

        # Display label
        ax.text(topleft_x, topleft_y - 10, label, color=color, fontsize=20)

    plt.show()

def plot_ground_truth_boxes_jsonl(image_file, jsonl_file):
    image_base_name = os.path.basename(image_file)
    ground_truth_data_found = False
    with open(jsonl_file) as fp:
        for line in fp.readlines():
            line_json = json.loads(line)
            filename = line_json["image_url"]
            if image_base_name in filename:
                ground_truth_data_found = True
                plot_ground_truth_boxes(image_file, line_json["label"])
                break

```

```
if not ground_truth_data_found:  
    print("Unable to find ground truth information for image:  
{}}".format(image_file))
```

Using the above helper functions, for any given image, you can run the following code to display the bounding boxes.

Python

```
image_file = "./odFridgeObjects/images/31.jpg"  
jsonl_file = "./odFridgeObjects/train_annotations.jsonl"  
  
plot_ground_truth_boxes_jsonl(image_file, jsonl_file)
```

Upload data and create MLTable

In order to use the data for training, upload data to default Blob Storage of your Azure Machine Learning Workspace and register it as an asset. The benefits of registering data are:

- Easy to share with other members of the team
- Versioning of the metadata (location, description, etc.)
- Lineage tracking

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

Create a .yml file with the following configuration.

yml

```
$schema: https://azureschemas.azureedge.net/latest/data.schema.json  
name: fridge-items-images-object-detection  
description: Fridge-items images Object detection  
path: ./data/odFridgeObjects  
type: uri_folder
```

To upload the images as a data asset, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

Azure CLI

```
az ml data create -f [PATH_TO_YML_FILE] --workspace-name  
[YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --
```

```
subscription [YOUR_AZURE_SUBSCRIPTION]
```

Next step is to create `MLTable` from your data in jsonl format as shown below. MLtable package your data into a consumable object for training.

YAML

```
paths:
- file: ./train_annotations.jsonl
transformations:
- read_json_lines:
  encoding: utf8
  invalid_lines: error
  include_path_column: false
- convert_column_types:
  - columns: image_url
    column_type: stream_info
```

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

The following configuration creates training and validation data from the MLTable.

YAML

```
target_column_name: label
training_data:
  path: data/training-mltable-folder
  type: mltable
validation_data:
  path: data/validation-mltable-folder
  type: mltable
```

Configure your object detection experiment

To configure automated ML jobs for image-related tasks, create a task specific AutoML job.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

```
task: image_object_detection  
primary_metric: mean_average_precision
```

Automatic hyperparameter sweeping for image tasks (AutoMode)

Important

This feature is currently in public preview. This preview version is provided without a service-level agreement. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In your AutoML job, you can perform an automatic hyperparameter sweep in order to find the optimal model (we call this functionality AutoMode). You only specify the number of trials; the hyperparameter search space, sampling method and early termination policy are not needed. The system will automatically determine the region of the hyperparameter space to sweep based on the number of trials. A value between 10 and 20 will likely work well on many datasets.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

```
limits:  
  max_trials: 10  
  max_concurrent_trials: 2
```

You can then submit the job to train an image model.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

To submit your AutoML job, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

Azure CLI

```
az ml job create --file ./hello-automl-job-basic.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Manual hyperparameter sweeping for image tasks

In your AutoML job, you can specify the model architectures by using `model_name` parameter and configure the settings to perform a hyperparameter sweep over a defined search space to find the optimal model.

In this example, we will train an object detection model with `yolov5` and `fasterrcnn_resnet50_fpn`, both of which are pretrained on COCO, a large-scale object detection, segmentation, and captioning dataset that contains over thousands of labeled images with over 80 label categories.

You can perform a hyperparameter sweep over a defined search space to find the optimal model.

Job limits

You can control the resources spent on your AutoML Image training job by specifying the `timeout_minutes`, `max_trials` and the `max_concurrent_trials` for the job in limit settings. Please refer to [detailed description on Job Limits parameters](#).

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

```
limits:  
  timeout_minutes: 60  
  max_trials: 10  
  max_concurrent_trials: 2
```

The following code defines the search space in preparation for the hyperparameter sweep for each defined architecture, `yolov5` and `fasterrcnn_resnet50_fpn`. In the search space, specify the range of values for `learning_rate`, `optimizer`, `lr_scheduler`, etc., for AutoML to choose from as it attempts to generate a model with the optimal primary

metric. If hyperparameter values are not specified, then default values are used for each architecture.

For the tuning settings, use random sampling to pick samples from this parameter space by using the `random` sampling_algorithm. The job limits configured above, tells automated ML to try a total of 10 trials with these different samples, running two trials at a time on our compute target, which was set up using four nodes. The more parameters the search space has, the more trials you need to find optimal models.

The Bandit early termination policy is also used. This policy terminates poor performing trials; that is, those trials that are not within 20% slack of the best performing trial, which significantly saves compute resources.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

```
sweep:
    sampling_algorithm: random
    early_termination:
        type: bandit
        evaluation_interval: 2
        slack_factor: 0.2
        delay_evaluation: 6
```

YAML

```
search_space:
    - model_name:
        type: choice
        values: [yolov5]
    learning_rate:
        type: uniform
        min_value: 0.0001
        max_value: 0.01
    model_size:
        type: choice
        values: [small, medium]

    - model_name:
        type: choice
        values: [fasterrcnn_resnet50_fpn]
    learning_rate:
        type: uniform
        min_value: 0.0001
        max_value: 0.001
```

```
optimizer:  
    type: choice  
    values: [sgd, adam, adamw]  
min_size:  
    type: choice  
    values: [600, 800]
```

Once the search space and sweep settings are defined, you can then submit the job to train an image model using your training dataset.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

To submit your AutoML job, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

Azure CLI

```
az ml job create --file ./hello-automl-job-basic.yml --workspace-name  
[YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --  
subscription [YOUR_AZURE_SUBSCRIPTION]
```

When doing a hyperparameter sweep, it can be useful to visualize the different trials that were tried using the HyperDrive UI. You can navigate to this UI by going to the 'Child jobs' tab in the UI of the main automl_image_job from above, which is the HyperDrive parent job. Then you can go into the 'Child jobs' tab of this one.

Alternatively, here below you can see directly the HyperDrive parent job and navigate to its 'Child jobs' tab:

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

CLI example not available, please use Python SDK.

Register and deploy model

Once the job completes, you can register the model that was created from the best trial (configuration that resulted in the best primary metric). You can either register the model after downloading or by specifying the `azureml` path with corresponding `jobid`.

Get the best trial

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

CLI example not available, please use Python SDK.

Register the model

Register the model either using the `azureml` path or your locally downloaded path.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

Azure CLI

```
az ml model create --name od-fridge-items-mlflow-model --version 1 --  
path azureml://jobs/$best_run/outputs/artifacts/outputs/mlflow-model/ --  
type mlflow_model --workspace-name [YOUR_AZURE_WORKSPACE] --resource-  
group [YOUR_AZURE_RESOURCE_GROUP] --subscription  
[YOUR_AZURE_SUBSCRIPTION]
```

After you register the model you want to use, you can deploy it using the managed online endpoint [deploy-managed-online-endpoint](#)

Configure online endpoint

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

```
$schema:  
https://azuremlschemas.azureedge.net/latest/managedOnlineEndpoint.schema  
.json  
name: od-fridge-items-endpoint  
auth_mode: key
```

Create the endpoint

Using the `MLClient` created earlier, we'll now create the Endpoint in the workspace. This command will start the endpoint creation and return a confirmation response while the endpoint creation continues.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

Azure CLI

```
az ml online-endpoint create --file .\create_endpoint.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

We can also create a batch endpoint for batch inferencing on large volumes of data over a period of time. Check out the [object detection batch scoring](#) notebook for batch inferencing using the batch endpoint.

Configure online deployment

A deployment is a set of resources required for hosting the model that does the actual inferencing. We'll create a deployment for our endpoint using the `ManagedOnlineDeployment` class. You can use either GPU or CPU VM SKUs for your deployment cluster.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

```
name: od-fridge-items-mlflow-deploy  
endpoint_name: od-fridge-items-endpoint
```

```
model: azureml:od-fridge-items-mlflow-model@latest
instance_type: Standard_DS3_v2
instance_count: 1
liveness_probe:
    failure_threshold: 30
    success_threshold: 1
    timeout: 2
    period: 10
    initial_delay: 2000
readiness_probe:
    failure_threshold: 10
    success_threshold: 1
    timeout: 10
    period: 10
    initial_delay: 2000
```

Create the deployment

Using the `MLClient` created earlier, we'll now create the deployment in the workspace. This command will start the deployment creation and return a confirmation response while the deployment creation continues.

Azure CLI

APPLIES TO: Azure CLI ml extension v2 (current)

Azure CLI

```
az ml online-deployment create --file .\create_deployment.yml --
workspace-name [YOUR_AZURE_WORKSPACE] --resource-group
[YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Update traffic:

By default the current deployment is set to receive 0% traffic. you can set the traffic percentage current deployment should receive. Sum of traffic percentages of all the deployments with one end point should not exceed 100%.

Azure CLI

APPLIES TO: Azure CLI ml extension v2 (current)

Azure CLI

```
az ml online-endpoint update --name 'od-fridge-items-endpoint' --traffic  
'od-fridge-items-mlflow-deploy=100' --workspace-name  
[YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --  
subscription [YOUR_AZURE_SUBSCRIPTION]
```

Test the deployment

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

CLI example not available, please use Python SDK.

Visualize detections

Now that you have scored a test image, you can visualize the bounding boxes for this image. To do so, be sure you have matplotlib installed.

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

YAML

CLI example not available, please use Python SDK.

Clean up resources

Do not complete this section if you plan on running other Azure Machine Learning tutorials.

If you don't plan to use the resources you created, delete them, so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group you created.

3. Select **Delete resource group**.
4. Enter the resource group name. Then select **Delete**.

You can also keep the resource group but delete a single workspace. Display the workspace properties and select **Delete**.

Next steps

In this automated machine learning tutorial, you did the following tasks:

- ✓ Configured a workspace and prepared data for an experiment.
 - ✓ Trained an automated object detection model
 - ✓ Specified hyperparameter values for your model
 - ✓ Performed a hyperparameter sweep
 - ✓ Deployed your model
 - ✓ Visualized detections
- [Learn more about computer vision in automated ML](#).
 - [Learn how to set up AutoML to train computer vision models with Python](#).
 - [Learn how to configure incremental training on computer vision models](#).
 - See [what hyperparameters are available for computer vision tasks](#).
 - Code examples:

Azure CLI

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

- Review detailed code examples and use cases in the [azureml-examples repository for automated machine learning samples](#). Please check the folders with 'cli-automl-image-' prefix for samples specific to building computer vision models.

Note

Use of the fridge objects dataset is available through the license under the [MIT License](#).

Additional resources

Documentation

[Prepare data for computer vision tasks - Azure Machine Learning](#)

Image data preparation for Azure Machine Learning automated ML to train computer vision models on classification, object detection, and segmentation

[Set up AutoML for computer vision - Azure Machine Learning](#)

Set up Azure Machine Learning automated ML to train computer vision models with the CLI v2 and Python SDK v2.

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[JSONL format for computer vision tasks - Azure Machine Learning](#)

Learn how to format your JSONL files for data consumption in automated ML experiments for computer vision tasks with the CLI v2 and Python SDK v2.

[Overview of forecasting methods in AutoML - Azure Machine Learning](#)

Learn how Azure Machine Learning's AutoML uses machine learning to build forecasting models

[Build & train models - Azure Machine Learning](#)

Learn how to train models with Azure Machine Learning. Explore the different training methods and choose the right one for your project.

[What is distributed training? - Azure Machine Learning](#)

Learn what type of distributed training Azure Machine Learning supports and the open source framework integrations available for distributed training.

[Share Responsible AI insights and make data-driven decisions with Azure Machine Learning Responsible AI scorecard - Azure Machine Learning](#)

Learn about how to use the Responsible AI scorecard to share responsible AI insights from your machine learning models and make data-driven decisions with non-technical and technical stakeholders.

[Show 5 more](#)

Training

Learning paths and modules

[Run jobs in Azure Machine Learning with CLI \(v2\) - Training](#)

Run jobs in Azure Machine Learning with CLI (v2)

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Tutorial: Train a classification model with no-code AutoML in the Azure Machine Learning studio

Article • 10/19/2022 • 13 minutes to read

Learn how to train a classification model with no-code AutoML using Azure Machine Learning automated ML in the Azure Machine Learning studio. This classification model predicts if a client will subscribe to a fixed term deposit with a financial institution.

With automated ML, you can automate away time intensive tasks. Automated machine learning rapidly iterates over many combinations of algorithms and hyperparameters to help you find the best model based on a success metric of your choosing.

You won't write any code in this tutorial, you'll use the studio interface to perform training. You'll learn how to do the following tasks:

- ✓ Create an Azure Machine Learning workspace.
- ✓ Run an automated machine learning experiment.
- ✓ Explore model details.
- ✓ Deploy the recommended model.

Also try automated machine learning for these other model types:

- For a no-code example of forecasting, see [Tutorial: Demand forecasting & AutoML](#).
- For a code first example of an object detection model, see the [Tutorial: Train an object detection model with AutoML and Python](#),

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#).
- Download the [bankmarketing_train.csv](#) data file. The y column indicates if a customer subscribed to a fixed term deposit, which is later identified as the target column for predictions in this tutorial.

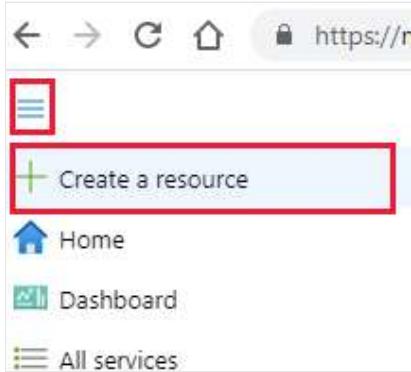
Create a workspace

An Azure Machine Learning workspace is a foundational resource in the cloud that you use to experiment, train, and deploy machine learning models. It ties your Azure

subscription and resource group to an easily consumed object in the service.

There are many [ways to create a workspace](#). In this tutorial, you create a workspace via the Azure portal, a web-based console for managing your Azure resources.

1. Sign in to the [Azure portal](#) by using the credentials for your Azure subscription.
2. In the upper-left corner of the Azure portal, select the three bars, then **+ Create a resource**.



3. Use the search bar to find **Azure Machine Learning**.
4. Select **Azure Machine Learning**.

Azure Machine Learning

Category : All

Pricing : All

Azure benefit eligible only Publisher Type : All Product

Showing 1 to 20 of 362 results for 'Azure Machine Learning'. [Clear search](#)

 <h3>Azure Machine Learning</h3> <p>Microsoft</p> <p>Azure Service</p> <p>Enterprise-grade machine learning to build and deploy models faster</p> <p>Create </p>	 <h3>Jupyter Hub for Machine Learning using Python</h3> <p>Data Science Dojo</p> <p>Virtual Machine</p> <p>Our Jupyter Instance provides easy to use environment for Machine Learning applications.</p> <p>Starts at Free</p> <p>Create </p>
--	---

5. In the **Machine Learning** pane, select **Create** to begin.

6. Provide the following information to configure your new workspace:

Field	Description
Workspace name	Enter a unique name that identifies your workspace. In this example, we use docs-ws . Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others.
Subscription	Select the Azure subscription that you want to use.
Resource group	Use an existing resource group in your subscription, or enter a name to create a new resource group. A resource group holds related resources for an Azure solution. In this example, we use docs-aml .
Region	Select the location closest to your users and the data resources to create your workspace.

Field	Description
Storage account	A storage account is used as the default datastore for the workspace. You may create a new Azure Storage resource or select an existing one in your subscription.
Key vault	A key vault is used to store secrets and other sensitive information that is needed by the workspace. You may create a new Azure Key Vault resource or select an existing one in your subscription.
Application insights	The workspace uses Azure Application Insights to store monitoring information about your deployed models. You may create a new Azure Application Insights resource or select an existing one in your subscription.
Container registry	A container registry is used to register docker images used in training and deployments. You may choose to create a resource or select an existing one in your subscription.

7. After you're finished configuring the workspace, select **Review + Create**.

8. Select **Create** to create the workspace.

 **Warning**

It can take several minutes to create your workspace in the cloud.

When the process is finished, a deployment success message appears.

9. To view the new workspace, select **Go to resource**.

10. From the portal view of your workspace, select **Launch studio** to go to the Azure Machine Learning studio.

 **Important**

Take note of your **workspace** and **subscription**. You'll need these to ensure you create your experiment in the right place.

Sign in to the studio

You complete the following experiment set-up and run steps via the Azure Machine Learning studio at <https://ml.azure.com>, a consolidated web interface that includes machine learning tools to perform data science scenarios for data science practitioners of all skill levels. The studio is not supported on Internet Explorer browsers.

1. Sign in to [Azure Machine Learning studio](#).
2. Select your subscription and the workspace you created.
3. Select **Get started**.
4. In the left pane, select **Automated ML** under the **Author** section.

Since this is your first automated ML experiment, you'll see an empty list and links to documentation.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar has a tree view with 'Microsoft' expanded, showing 'New', 'Home', 'Author' (which is selected), and 'Notebooks'. Under 'Author', there's a 'Automated ML' section with 'Designer' selected, followed by 'Assets', 'Data', 'Jobs', 'Components', 'Pipelines', 'Environments', 'Models', and 'Endpoints'. Below this is a 'Manage' section with 'Compute' and 'Linked Services'. The main content area has a breadcrumb path 'Microsoft > automl_ws > Automated ML'. It displays a heading 'Automated ML' with a sub-instruction: 'Let Automated ML train and find the best model based on your data without writing a single line of code. [Learn more about Automated ML](#)'. Below this is a button '+ New Automated ML job' which is highlighted with a red box. To its right is a 'Refresh' button. The central message says 'No recent Automated ML jobs to display.' and 'Click "New Automated ML job" to create your first job'. At the bottom of this section is a link 'Learn more about creating Automated ML jobs'. The bottom part of the screen shows a 'Documentation' section with three items: 'Concept: What is Automated ML?', 'Tutorial: Create your first classification model with Automated ML', and 'Blog: Build more accurate forecasts with new capabilities in Automated ML'. There is also a 'View all documentation' link.

5. Select **+ New automated ML job**.

Create and load dataset

Before you configure your experiment, upload your data file to your workspace in the form of an Azure Machine Learning dataset. Doing so, allows you to ensure that your data is formatted appropriately for your experiment.

1. Create a new dataset by selecting **From local files** from the **+Create dataset** dropdown.
 - a. On the **Basic info** form, give your dataset a name and provide an optional description. The automated ML interface currently only supports TabularDatasets, so the dataset type should default to *Tabular*.
 - b. Select **Next** on the bottom left

c. On the **Datastore and file selection** form, select the default datastore that was automatically set up during your workspace creation, **workspaceblobstore (Azure Blob Storage)**. This is where you'll upload your data file to make it available to your workspace.

- d. Select **Upload files** from the **Upload** drop-down.
- e. Choose the **bankmarketing_train.csv** file on your local computer. This is the file you downloaded as a [prerequisite ↗](#).
- f. Select **Next** on the bottom left, to upload it to the default container that was automatically set up during your workspace creation.

When the upload is complete, the **Settings and preview** form is pre-populated based on the file type.

- g. Verify that the **Settings and preview** form is populated as follows and select **Next**.

Field	Description	Value for tutorial
File format	Defines the layout and type of data stored in a file.	Delimited
Delimiter	One or more characters for specifying the boundary between separate, independent regions in plain text or other data streams.	Comma
Encoding	Identifies what bit to character schema table to use to read your dataset.	UTF-8
Column headers	Indicates how the headers of the dataset, if any, will be treated.	All files have same headers
Skip rows	Indicates how many, if any, rows are skipped in the dataset.	None

- h. The **Schema** form allows for further configuration of your data for this experiment. For this example, select the toggle switch for the **day_of_week**, so

as to not include it. Select **Next**.

Include	Column name	Properties	Type
Path	age	Not applicable to selected type	String
job	marital	Not applicable to selected type	String
education	default	Not applicable to selected type	String
housing	loan	Not applicable to selected type	String

- i. On the **Confirm details** form, verify the information matches what was previously populated on the **Basic info**, **Datastore and file selection** and **Settings and preview** forms.
- j. Select **Create** to complete the creation of your dataset.
- k. Select your dataset once it appears in the list.
- l. Review the **Data preview** to ensure you didn't include `day_of_week` then, select **Close**.
- m. Select **Next**.

Configure job

After you load and configure your data, you can set up your experiment. This setup includes experiment design tasks such as, selecting the size of your compute environment and specifying what column you want to predict.

1. Select the **Create new** radio button.
2. Populate the **Configure Job** form as follows:
 - a. Enter this experiment name: `my-1st-automl-experiment`
 - b. Select `y` as the target column, what you want to predict. This column indicates whether the client subscribed to a term deposit or not.

- c. Select **compute cluster** as your compute type.
- d. +**New** to configure your compute target. A compute target is a local or cloud-based resource environment used to run your training script or host your service deployment. For this experiment, we use a cloud-based compute.
- i. Populate the **Select virtual machine** form to set up your compute.

Field	Description	Value for tutorial
Location	Your region that you'd like to run the machine from	West US 2
Virtual machine tier	Select what priority your experiment should have	Dedicated
Virtual machine type	Select the virtual machine type for your compute.	CPU (Central Processing Unit)
Virtual machine size	Select the virtual machine size for your compute. A list of recommended sizes is provided based on your data and experiment type.	Standard_DS12_V2

- ii. Select **Next** to populate the **Configure settings** form.

Field	Description	Value for tutorial
Compute name	A unique name that identifies your compute context.	automl-compute
Min / Max nodes	To profile data, you must specify 1 or more nodes.	Min nodes: 1 Max nodes: 6
Idle seconds before scale down	Idle time before the cluster is automatically scaled down to the minimum node count.	120 (default)
Advanced settings	Settings to configure and authorize a virtual network for your experiment.	None

- iii. Select **Create** to create your compute target.

This takes a couple minutes to complete.

Create compute cluster ⓘ

Virtual Machine		Configure Settings						
		Name	Category	Cores	Available quota	RAM	Storage	Cost/Hour
		Standard_DS12_v2	Memory optimized	4	94 cores	28 GB	56 GB	\$0.37/hr
<input checked="" type="checkbox"/> Settings		Compute name * ⓘ <input type="text" value="automl-compute1"/> Minimum number of nodes * ⓘ  <small>To avoid charges when no jobs are running, set the minimum nodes to 0. This setting allows Azure Machine Learning to de-allocate the compute nodes when idle. Any higher value will result in charges for the number of nodes allocated.</small> Maximum number of nodes * ⓘ  Idle seconds before scale down * ⓘ <input type="text" value="120"/> <input checked="" type="checkbox"/> Enable SSH access ⓘ Advanced settings <input checked="" type="checkbox"/> Enable virtual network ⓘ <input checked="" type="checkbox"/> Assign a managed identity ⓘ						
		<input type="button" value="Back"/>	<input style="background-color: #0072BC; color: white; font-weight: bold; border-radius: 5px; padding: 5px 10px; border: none; font-size: 10pt; margin-right: 10px;" type="button" value="Create"/>	Download a template for automation				
								<input type="button" value="Cancel"/>

iv. After creation, select your new compute target from the drop-down list.

e. Select **Next**.

3. On the **Select task and settings** form, complete the setup for your automated ML experiment by specifying the machine learning task type and configuration settings.

a. Select **Classification** as the machine learning task type.

b. Select **View additional configuration settings** and populate the fields as follows. These settings are to better control the training job. Otherwise, defaults are applied based on experiment selection and data.

Additional configurations	Description	Value for tutorial
Primary metric	Evaluation metric that the machine learning algorithm will be measured by.	AUC_weighted
Explain best model	Automatically shows explainability on the best model created by automated ML.	Enable

Additional configurations	Description	Value for tutorial
Blocked algorithms	Algorithms you want to exclude from the training job	None
Additional classification settings	These settings help improve the accuracy of your model	Positive class label: None
Exit criterion	If a criteria is met, the training job is stopped.	Training job time (hours): 1 Metric score threshold: None
Concurrency	The maximum number of parallel iterations executed per iteration	Max concurrent iterations: 5

Select **Save**.

- c. Select **Next**.
- 4. On the **[Optional] Validate and test** form,
 - a. Select k-fold cross-validation as your **Validation type**.
 - b. Select 2 as your **Number of cross validations**.
- 5. Select **Finish** to run the experiment. The **Job Detail** screen opens with the **Job status** at the top as the experiment preparation begins. This status updates as the experiment progresses. Notifications also appear in the top right corner of the studio to inform you of the status of your experiment.

Important

Preparation takes **10-15 minutes** to prepare the experiment run. Once running, it takes **2-3 minutes more for each iteration**.

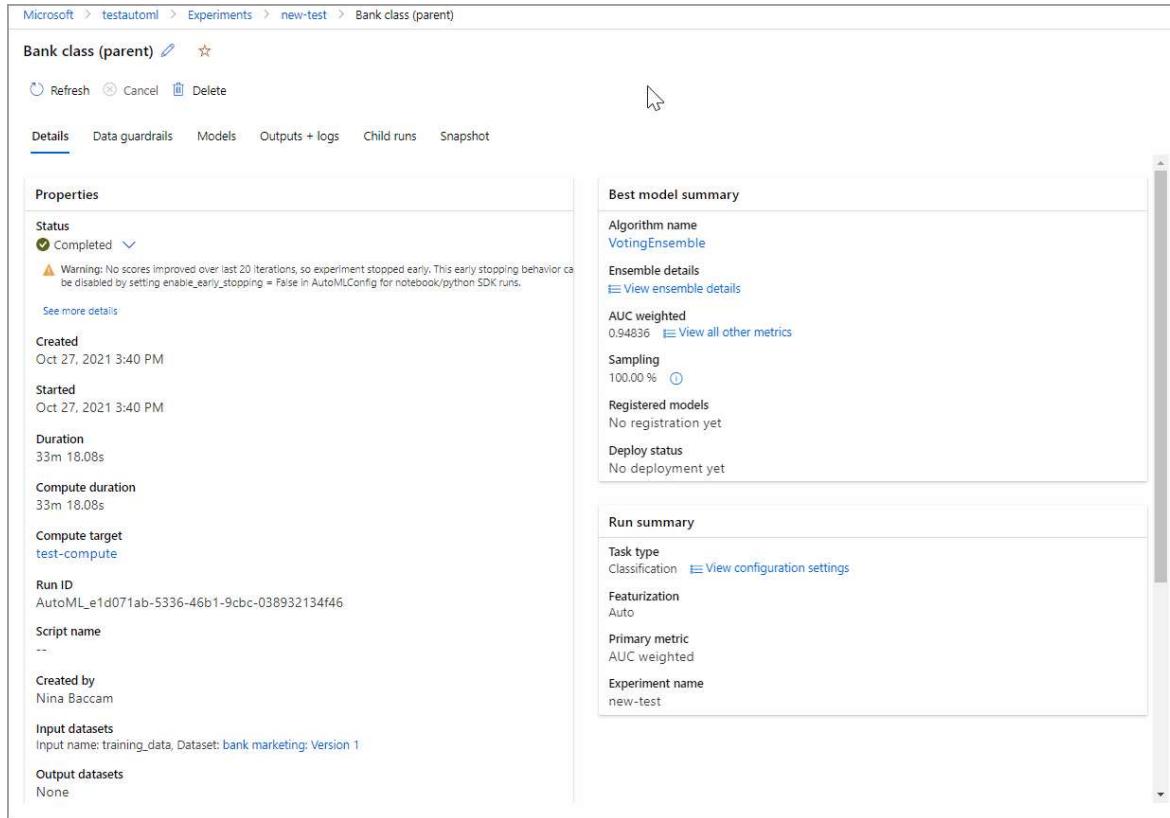
In production, you'd likely walk away for a bit. But for this tutorial, we suggest you start exploring the tested algorithms on the **Models** tab as they complete while the others are still running.

Explore models

Navigate to the **Models** tab to see the algorithms (models) tested. By default, the models are ordered by metric score as they complete. For this tutorial, the model that scores the highest based on the chosen **AUC_weighted** metric is at the top of the list.

While you wait for all of the experiment models to finish, select the **Algorithm name** of a completed model to explore its performance details.

The following navigates through the **Details** and the **Metrics** tabs to view the selected model's properties, metrics, and performance charts.



The screenshot shows the Microsoft AutoML interface with the following details:

- Properties** tab selected.
- Status**: Completed (green circle).
- Warning**: No scores improved over last 20 iterations, so experiment stopped early. This early stopping behavior can be disabled by setting enable_early_stopping = False in AutoMLConfig for notebook/python SDK runs.
- Created**: Oct 27, 2021 3:40 PM.
- Started**: Oct 27, 2021 3:40 PM.
- Duration**: 33m 18.08s.
- Compute duration**: 33m 18.08s.
- Compute target**: test-compute.
- Run ID**: AutoML_e1d071ab-5336-46b1-9cbc-038932134f46.
- Script name**: --.
- Created by**: Nina Baccam.
- Input datasets**: Input name: training_data, Dataset: bank marketing, Version 1.
- Output datasets**: None.
- Best model summary**: Algorithm name: VotingEnsemble, Ensemble details: View ensemble details, AUC weighted: 0.94836, Sampling: 100.00%, Registered models: No registration yet, Deploy status: No deployment yet.
- Run summary**: Task type: Classification, View configuration settings, Featurization: Auto, Primary metric: AUC weighted, Experiment name: new-test.

Model explanations

While you wait for the models to complete, you can also take a look at model explanations and see which data features (raw or engineered) influenced a particular model's predictions.

These model explanations can be generated on demand, and are summarized in the model explanations dashboard that's part of the **Explanations (preview)** tab.

To generate model explanations,

1. Select **Job 1** at the top to navigate back to the **Models** screen.
2. Select the **Models** tab.
3. For this tutorial, select the first **MaxAbsScaler, LightGBM** model.
4. Select the **Explain model** button at the top. On the right, the **Explain model** pane appears.

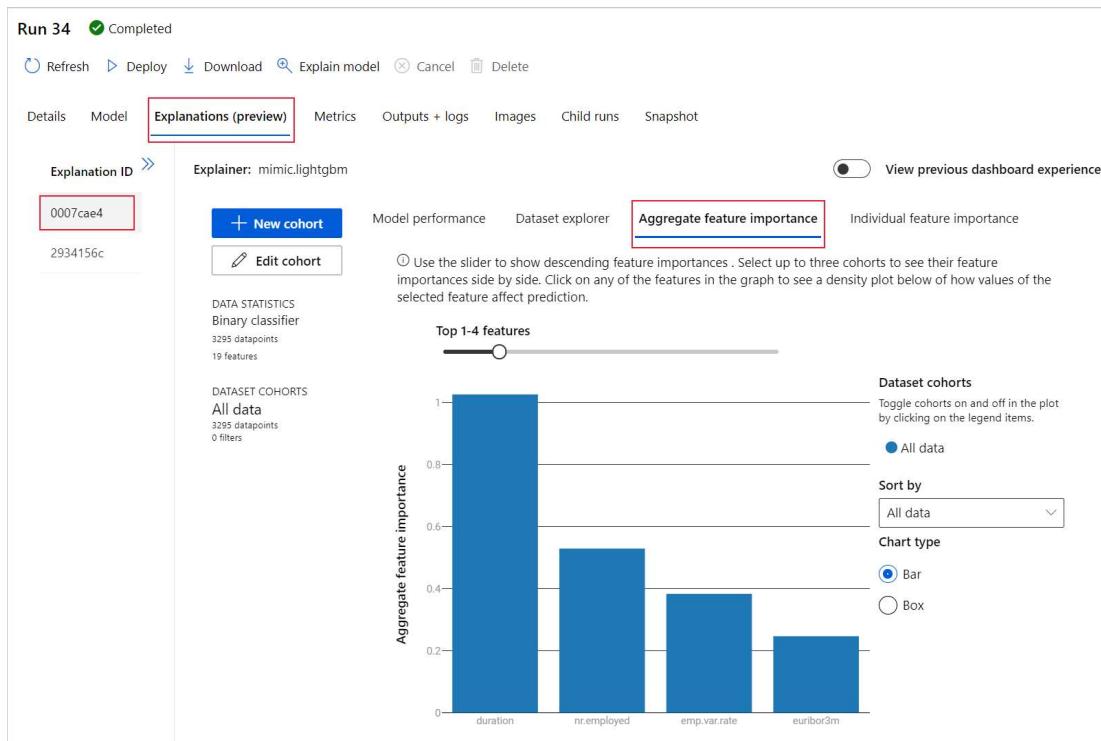
5. Select the **automl-compute** that you created previously. This compute cluster initiates a child job to generate the model explanations.
6. Select **Create** at the bottom. A green success message appears towards the top of your screen.

! Note

The explainability job takes about 2-5 minutes to complete.

7. Select the **Explanations (preview)** button. This tab populates once the explainability run completes.
8. On the left hand side, expand the pane and select the row that says **raw** under **Features**.
9. Select the **Aggregate feature importance** tab on the right. This chart shows which data features influenced the predictions of the selected model.

In this example, the *duration* appears to have the most influence on the predictions of this model.



Deploy the best model

The automated machine learning interface allows you to deploy the best model as a web service in a few steps. Deployment is the integration of the model so it can predict on new data and identify potential areas of opportunity.

For this experiment, deployment to a web service means that the financial institution now has an iterative and scalable web solution for identifying potential fixed term deposit customers.

Check to see if your experiment run is complete. To do so, navigate back to the parent job page by selecting **Job 1** at the top of your screen. A **Completed** status is shown on the top left of the screen.

Once the experiment run is complete, the **Details** page is populated with a **Best model summary** section. In this experiment context, **VotingEnsemble** is considered the best model, based on the **AUC_weighted** metric.

We deploy this model, but be advised, deployment takes about 20 minutes to complete. The deployment process entails several steps including registering the model, generating resources, and configuring them for the web service.

1. Select **VotingEnsemble** to open the model-specific page.
2. Select the **Deploy** menu in the top-left and select **Deploy to web service**.
3. Populate the **Deploy a model** pane as follows:

Field	Value
Deployment name	my-automl-deploy
Deployment description	My first automated machine learning experiment deployment
Compute type	Select Azure Container Instance (ACI)
Enable authentication	Disable.
Use custom deployments	Disable. Allows for the default driver file (scoring script) and environment file to be auto-generated.

For this example, we use the defaults provided in the *Advanced* menu.

4. Select **Deploy**.

A green success message appears at the top of the **Job** screen, and in the **Model summary** pane, a status message appears under **Deploy status**. Select **Refresh** periodically to check the deployment status.

Now you have an operational web service to generate predictions.

Proceed to the [Next Steps](#) to learn more about how to consume your new web service, and test your predictions using Power BI's built in Azure Machine Learning support.

Clean up resources

Deployment files are larger than data and experiment files, so they cost more to store. Delete only the deployment files to minimize costs to your account, or if you want to keep your workspace and experiment files. Otherwise, delete the entire resource group, if you don't plan to use any of the files.

Delete the deployment instance

Delete just the deployment instance from Azure Machine Learning at <https://ml.azure.com/>, if you want to keep the resource group and workspace for other tutorials and exploration.

1. Go to [Azure Machine Learning](#). Navigate to your workspace and on the left under the **Assets** pane, select **Endpoints**.
2. Select the deployment you want to delete and select **Delete**.
3. Select **Proceed**.

Delete the resource group

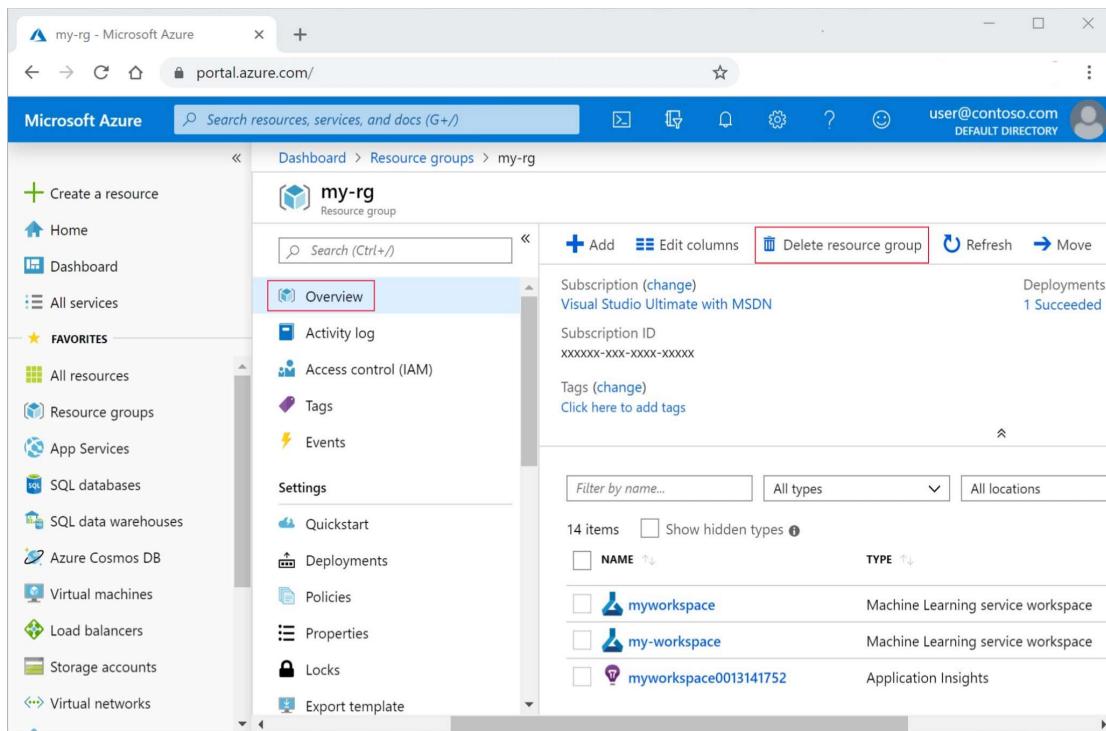
Important

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.

3. Select **Delete resource group**.



The screenshot shows the Microsoft Azure portal interface. The left sidebar lists various service categories like Home, Dashboard, All services, and Favorites. Under Favorites, 'Resource groups' is selected, showing 'my-rg' as the current resource group. The main content area displays the 'Overview' tab for 'my-rg'. At the top right of this section is a red-bordered 'Delete resource group' button. Below it, there's information about the subscription (Visual Studio Ultimate with MSDN) and deployment status (1 Succeeded). A table lists 14 items, including three workspace entries: 'myworkspace', 'my-workspace', and 'myworkspace0013141752', along with their types (Machine Learning service workspace or Application Insights).

4. Enter the resource group name. Then select **Delete**.

Next steps

In this automated machine learning tutorial, you used Azure Machine Learning's automated ML interface to create and deploy a classification model. See these articles for more information and next steps:

Consume a web service

- Learn more about [automated machine learning](#).
- For more information on classification metrics and charts, see the [Understand automated machine learning results](#) article.
- Learn more about [featurization](#).
- Learn more about [data profiling](#).

Note

This Bank Marketing dataset is made available under the [Creative Commons \(CC0: Public Domain\) License](#). Any rights in individual contents of the database are licensed under the [Database Contents License](#) and available on [Kaggle](#). This dataset was originally available within the [UCI Machine Learning Database](#).

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. *Decision Support Systems*, Elsevier, 62:22-31, June 2014.

Tutorial: Forecast demand with no-code automated machine learning in the Azure Machine Learning studio

Article • 10/19/2022 • 10 minutes to read

Learn how to create a [time-series forecasting model](#) without writing a single line of code using automated machine learning in the Azure Machine Learning studio. This model will predict rental demand for a bike sharing service.

You won't write any code in this tutorial, you'll use the studio interface to perform training. You'll learn how to do the following tasks:

- ✓ Create and load a dataset.
- ✓ Configure and run an automated ML experiment.
- ✓ Specify forecasting settings.
- ✓ Explore the experiment results.
- ✓ Deploy the best model.

Also try automated machine learning for these other model types:

- For a no-code example of a classification model, see [Tutorial: Create a classification model with automated ML in Azure Machine Learning](#).
- For a code first example of an object detection model, see the [Tutorial: Train an object detection model with AutoML and Python](#).

Prerequisites

- An Azure Machine Learning workspace. See [Create workspace resources](#).
- Download the [bike-no.csv](#) data file

Sign in to the studio

For this tutorial, you create your automated ML experiment run in Azure Machine Learning studio, a consolidated web interface that includes machine learning tools to perform data science scenarios for data science practitioners of all skill levels. The studio is not supported on Internet Explorer browsers.

1. Sign in to [Azure Machine Learning studio](#).

2. Select your subscription and the workspace you created.
3. Select **Get started**.
4. In the left pane, select **Automated ML** under the **Author** section.
5. Select **+ New automated ML job**.

Create and load dataset

Before you configure your experiment, upload your data file to your workspace in the form of an Azure Machine Learning dataset. Doing so, allows you to ensure that your data is formatted appropriately for your experiment.

1. On the **Select dataset** form, select **From local files** from the **+Create dataset** drop-down.
 - a. On the **Basic info** form, give your dataset a name and provide an optional description. The dataset type should default to **Tabular**, since automated ML in Azure Machine Learning studio currently only supports tabular datasets.
 - b. Select **Next** on the bottom left
 - c. On the **Datastore and file selection** form, select the default datastore that was automatically set up during your workspace creation, **workspaceblobstore (Azure Blob Storage)**. This is the storage location where you'll upload your data file.
 - d. Select **Upload files** from the **Upload** drop-down..
 - e. Choose the **bike-no.csv** file on your local computer. This is the file you downloaded as a [prerequisite](#).
 - f. Select **Next**

When the upload is complete, the Settings and preview form is pre-populated based on the file type.

- g. Verify that the **Settings and preview** form is populated as follows and select **Next**.

Field	Description	Value for tutorial

Field	Description	Value for tutorial
File format	Defines the layout and type of data stored in a file.	Delimited
Delimiter	One or more characters for specifying the boundary between separate, independent regions in plain text or other data streams.	Comma
Encoding	Identifies what bit to character schema table to use to read your dataset.	UTF-8
Column headers	Indicates how the headers of the dataset, if any, will be treated.	Only first file has headers
Skip rows	Indicates how many, if any, rows are skipped in the dataset.	None

- h. The **Schema** form allows for further configuration of your data for this experiment.
- i. For this example, choose to ignore the **casual** and **registered** columns. These columns are a breakdown of the **cnt** column so, therefore we don't include them.
 - ii. Also for this example, leave the defaults for the **Properties** and **Type**.
 - iii. Select **Next**.
 - i. On the **Confirm details** form, verify the information matches what was previously populated on the **Basic info** and **Settings and preview** forms.
 - j. Select **Create** to complete the creation of your dataset.
 - k. Select your dataset once it appears in the list.
 - l. Select **Next**.

Configure job

After you load and configure your data, set up your remote compute target and select which column in your data you want to predict.

1. Populate the **Configure job** form as follows:

- a. Enter an experiment name: `automl-bikeshare`
- b. Select **cnt** as the target column, what you want to predict. This column indicates the number of total bike share rentals.
- c. Select **compute cluster** as your compute type.
- d. Select **+New** to configure your compute target. Automated ML only supports Azure Machine Learning compute.
- i. Populate the **Select virtual machine** form to set up your compute.

Field	Description	Value for tutorial
Virtual machine tier	Select what priority your experiment should have	Dedicated
Virtual machine type	Select the virtual machine type for your compute.	CPU (Central Processing Unit)
Virtual machine size	Select the virtual machine size for your compute. A list of recommended sizes is provided based on your data and experiment type.	Standard_DS12_V2

- ii. Select **Next** to populate the **Configure settings** form.

Field	Description	Value for tutorial
Compute name	A unique name that identifies your compute context.	bike-compute
Min / Max nodes	To profile data, you must specify 1 or more nodes.	Min nodes: 1 Max nodes: 6
Idle seconds before scale down	Idle time before the cluster is automatically scaled down to the minimum node count.	120 (default)
Advanced settings	Settings to configure and authorize a virtual network for your experiment.	None

- iii. Select **Create** to get the compute target.

This takes a couple minutes to complete.

- iv. After creation, select your new compute target from the drop-down list.

e. Select **Next**.

Select forecast settings

Complete the setup for your automated ML experiment by specifying the machine learning task type and configuration settings.

1. On the **Task type and settings** form, select **Time series forecasting** as the machine learning task type.
2. Select **date** as your **Time column** and leave **Time series identifiers** blank.
3. The **Frequency** is how often your historic data is collected. Keep **Autodetect** selected.
- 4.
5. The **forecast horizon** is the length of time into the future you want to predict. Deselect **Autodetect** and type 14 in the field.
6. Select **View additional configuration settings** and populate the fields as follows. These settings are to better control the training job and specify settings for your forecast. Otherwise, defaults are applied based on experiment selection and data.

Additional configurations	Description	Value for tutorial
Primary metric	Evaluation metric that the machine learning algorithm will be measured by.	Normalized root mean squared error
Explain best model	Automatically shows explainability on the best model created by automated ML.	Enable
Blocked algorithms	Algorithms you want to exclude from the training job	Extreme Random Trees
Additional forecasting settings	<p>These settings help improve the accuracy of your model.</p> <p>Forecast target lags: how far back you want to construct the lags of the target variable</p> <p>Target rolling window: specifies the size of the rolling window over which features, such as the <i>max</i>, <i>min</i> and <i>sum</i>, will be generated.</p>	Forecast target lags: None Target rolling window size: None

Additional configurations	Description	Value for tutorial
Exit criterion	If a criteria is met, the training job is stopped.	Training job time (hours): 3 Metric score threshold: None
Concurrency	The maximum number of parallel iterations executed per iteration	Max concurrent iterations: 6

Select **Save**.

7. Select **Next**.

8. On the **[Optional] Validate and test** form,
- Select k-fold cross-validation as your **Validation type**.
 - Select 5 as your **Number of cross validations**.

Run experiment

To run your experiment, select **Finish**. The **Job details** screen opens with the **Job status** at the top next to the job number. This status updates as the experiment progresses. Notifications also appear in the top right corner of the studio, to inform you of the status of your experiment.

Important

Preparation takes **10-15 minutes** to prepare the experiment job. Once running, it takes **2-3 minutes more for each iteration**.

In production, you'd likely walk away for a bit as this process takes time. While you wait, we suggest you start exploring the tested algorithms on the **Models** tab as they complete.

Explore models

Navigate to the **Models** tab to see the algorithms (models) tested. By default, the models are ordered by metric score as they complete. For this tutorial, the model that scores the highest based on the chosen **Normalized root mean squared error** metric is at the top of the list.

While you wait for all of the experiment models to finish, select the **Algorithm name** of a completed model to explore its performance details.

The following example navigates through the **Details** and the **Metrics** tabs to view the selected model's properties, metrics and performance charts.

The screenshot shows the Azure Machine Learning studio interface. The top navigation bar includes 'Microsoft' > 'testautoml' > 'Experiments' > 'new-test' > 'Bank class (parent)'. Below the navigation is a toolbar with 'Refresh', 'Cancel', and 'Delete' buttons. A dropdown menu is open next to the 'Delete' button. The main area has tabs: 'Details' (selected), 'Data guardrails', 'Models', 'Outputs + logs', 'Child runs', and 'Snapshot'. The 'Details' tab contains sections for 'Properties', 'Best model summary', and 'Run summary'. The 'Properties' section includes fields like 'Status' (Completed), 'Created' (Oct 27, 2021 3:40 PM), 'Started' (Oct 27, 2021 3:40 PM), 'Duration' (33m 18.08s), 'Compute duration' (33m 18.08s), 'Compute target' (test-compute), 'Run ID' (AutoML_e1d071ab-5336-46b1-9cbc-038932134f46), 'Script name' (--), 'Created by' (Nina Baccam), 'Input datasets' (Input name: training_data, Dataset: bank marketing, Version 1), and 'Output datasets' (None). The 'Best model summary' section shows 'Algorithm name' (VotingEnsemble), 'Ensemble details' (View ensemble details), 'AUC weighted' (0.94836, View all other metrics), 'Sampling' (100.00 %), 'Registered models' (No registration yet), and 'Deploy status' (No deployment yet). The 'Run summary' section shows 'Task type' (Classification, View configuration settings), 'Featurization' (Auto), 'Primary metric' (AUC weighted), and 'Experiment name' (new-test).

Deploy the model

Automated machine learning in Azure Machine Learning studio allows you to deploy the best model as a web service in a few steps. Deployment is the integration of the model so it can predict on new data and identify potential areas of opportunity.

For this experiment, deployment to a web service means that the bike share company now has an iterative and scalable web solution for forecasting bike share rental demand.

Once the job is complete, navigate back to parent job page by selecting **Job 1** at the top of your screen.

In the **Best model summary** section, the best model in the context of this experiment, is selected based on the **Normalized root mean squared error metric**.

We deploy this model, but be advised, deployment takes about 20 minutes to complete. The deployment process entails several steps including registering the model, generating resources, and configuring them for the web service.

1. Select the **best model** to open the model-specific page.
2. Select the **Deploy** button located in the top-left area of the screen.
3. Populate the **Deploy a model** pane as follows:

Field	Value
Deployment name	bikeshare-deploy
Deployment description	bike share demand deployment
Compute type	Select Azure Compute Instance (ACI)
Enable authentication	Disable.
Use custom deployment assets	Disable. Disabling allows for the default driver file (scoring script) and environment file to be autogenerated.

For this example, we use the defaults provided in the *Advanced* menu.

4. Select **Deploy**.

A green success message appears at the top of the **Job** screen stating that the deployment was started successfully. The progress of the deployment can be found in the **Model summary** pane under **Deploy status**.

Once deployment succeeds, you have an operational web service to generate predictions.

Proceed to the [Next steps](#) to learn more about how to consume your new web service, and test your predictions using Power BI's built in Azure Machine Learning support.

Clean up resources

Deployment files are larger than data and experiment files, so they cost more to store. Delete only the deployment files to minimize costs to your account, or if you want to keep your workspace and experiment files. Otherwise, delete the entire resource group, if you don't plan to use any of the files.

Delete the deployment instance

Delete just the deployment instance from the Azure Machine Learning studio, if you want to keep the resource group and workspace for other tutorials and exploration.

1. Go to the [Azure Machine Learning studio](#). Navigate to your workspace and on the left under the **Assets** pane, select **Endpoints**.
2. Select the deployment you want to delete and select **Delete**.
3. Select **Proceed**.

Delete the resource group

ⓘ Important

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. The URL in the browser is portal.azure.com/. The page title is "my-rg - Microsoft Azure". The top navigation bar includes "Microsoft Azure", a search bar, and user information "user@contoso.com DEFAULT DIRECTORY". On the left, there's a sidebar with "Create a resource", "Home", "Dashboard", "All services", and a "FAVORITES" section containing "All resources", "Resource groups", "App Services", "SQL databases", "SQL data warehouses", "Azure Cosmos DB", "Virtual machines", "Load balancers", "Storage accounts", and "Virtual networks". The main content area shows "Resource groups > my-rg". The "Overview" tab is selected. At the top right of the overview blade are buttons for "Add", "Edit columns", "Delete resource group" (which is highlighted with a red box), "Refresh", and "Move". Below these are sections for "Subscription (change)", "Deployment ID", and "Tags (change)". A table at the bottom lists 14 items, including "myworkspace" (Machine Learning service workspace), "my-workspace" (Machine Learning service workspace), and "myworkspace0013141752" (Application Insights). There are filters for "Filter by name...", "All types", and "All locations".

4. Enter the resource group name. Then select **Delete**.

Next steps

In this tutorial, you used automated ML in the Azure Machine Learning studio to create and deploy a time series forecasting model that predicts bike share rental demand.

See this article for steps on how to create a Power BI supported schema to facilitate consumption of your newly deployed web service:

Consume a web service

- Learn more about [automated machine learning](#).
- For more information on classification metrics and charts, see the [Understand automated machine learning results](#) article.
- Learn more about [featurization](#).
- Learn more about [data profiling](#).

ⓘ Note

This bike share dataset has been modified for this tutorial. This dataset was made available as part of a [Kaggle competition](#) and was originally available via [Capital Bikeshare](#). It can also be found within the [UCI Machine Learning Database](#).

Source: Fanaee-T, Hadi, and Gama, Joao, Event labeling combining ensemble detectors and background knowledge, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

Tutorial: Designer - train a no-code regression model

Article • 02/24/2023 • 13 minutes to read

Train a linear regression model that predicts car prices using the Azure Machine Learning designer. This tutorial is part one of a two-part series.

This tutorial uses the Azure Machine Learning designer, for more information, see [What is Azure Machine Learning designer?](#)

In part one of the tutorial, you learn how to:

- ✓ Create a new pipeline.
- ✓ Import data.
- ✓ Prepare data.
- ✓ Train a machine learning model.
- ✓ Evaluate a machine learning model.

In [part two](#) of the tutorial, you deploy your model as a real-time inferencing endpoint to predict the price of any car based on technical specifications you send it.

ⓘ Note

A completed version of this tutorial is available as a sample pipeline.

To find it, go to the designer in your workspace. In the **New pipeline** section, select **Sample 1 - Regression: Automobile Price Prediction(Basic)**.

ⓘ Important

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Create a new pipeline

Azure Machine Learning pipelines organize multiple machine learning and data processing steps into a single resource. Pipelines let you organize, manage, and reuse complex machine learning workflows across projects and users.

To create an Azure Machine Learning pipeline, you need an Azure Machine Learning workspace. In this section, you learn how to create both these resources.

Create a new workspace

You need an Azure Machine Learning workspace to use the designer. The workspace is the top-level resource for Azure Machine Learning, it provides a centralized place to work with all the artifacts you create in Azure Machine Learning. For instruction on creating a workspace, see [Create workspace resources](#).

ⓘ Note

If your workspace uses a Virtual network, there are additional configuration steps you must use to use the designer. For more information, see [Use Azure Machine Learning studio in an Azure virtual network](#)

Create the pipeline

ⓘ Note

Designer supports two type of components, classic prebuilt components and custom components. These two types of components are not compatible.

Classic prebuilt components provides prebuilt components majorly for data processing and traditional machine learning tasks like regression and classification. This type of component continues to be supported but will not have any new components added.

Custom components allow you to provide your own code as a component. It supports sharing across workspaces and seamless authoring across Studio, CLI, and SDK interfaces.

This article applies to classic prebuilt components.

1. Sign in to ml.azure.com, and select the workspace you want to work with.
2. Select **Designer** -> **Classic prebuilt**

Microsoft Azure Machine Learning Studio

Microsoft > main > Designer

Designer

New pipeline

[Classic prebuilt](#) [Custom](#)

This low-code option uses existing prebuilt components and earlier versions of any new components added.

[Create a new pipeline using classic prebuilt components](#)



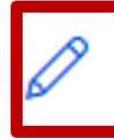
Image Classification using DenseNet

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with various options: Microsoft, New, Home, Author, Notebooks, Automated ML, Designer (which is selected and highlighted with a red box), Assets, Data, Jobs, Components, and Pipelines. The main area is titled 'Designer' and 'New pipeline'. It shows two tabs: 'Classic prebuilt' (selected) and 'Custom'. A descriptive text box explains that this is a low-code option using existing prebuilt components. Below it is a large button with a plus sign and the text 'Create a new pipeline using classic prebuilt components'. To the right, there's a diagram of a neural network architecture for image classification and a link to 'Image Classification using DenseNet'.

3. Select **Create a new pipeline using classic prebuilt components**.

4. Click the pencil icon beside the automatically generated pipeline draft name, rename it to *Automobile price prediction*. The name doesn't need to be unique.

Pipeline-Created-on-10-17-2022



Set the default compute target

A pipeline jobs on a compute target, which is a compute resource that's attached to your workspace. After you create a compute target, you can reuse it for future jobs.

i **Important**

Attached compute is not supported, use [compute instances or clusters](#) instead.

You can set a **Default compute target** for the entire pipeline, which will tell every component to use the same compute target by default. However, you can specify

compute targets on a per-module basis.

1. Select  **Settings** to the right of the canvas to open the **Settings** pane.

2. Select **Create Azure Machine Learning compute instance**.

If you already have an available compute target, you can select it from the **Select Azure Machine Learning compute instance** drop-down to run this pipeline.

3. Enter a name for the compute resource.

4. Select **Create**.

Note

It takes approximately five minutes to create a compute resource. After the resource is created, you can reuse it and skip this wait time for future jobs.

The compute resource autoscales to zero nodes when it's idle to save cost.

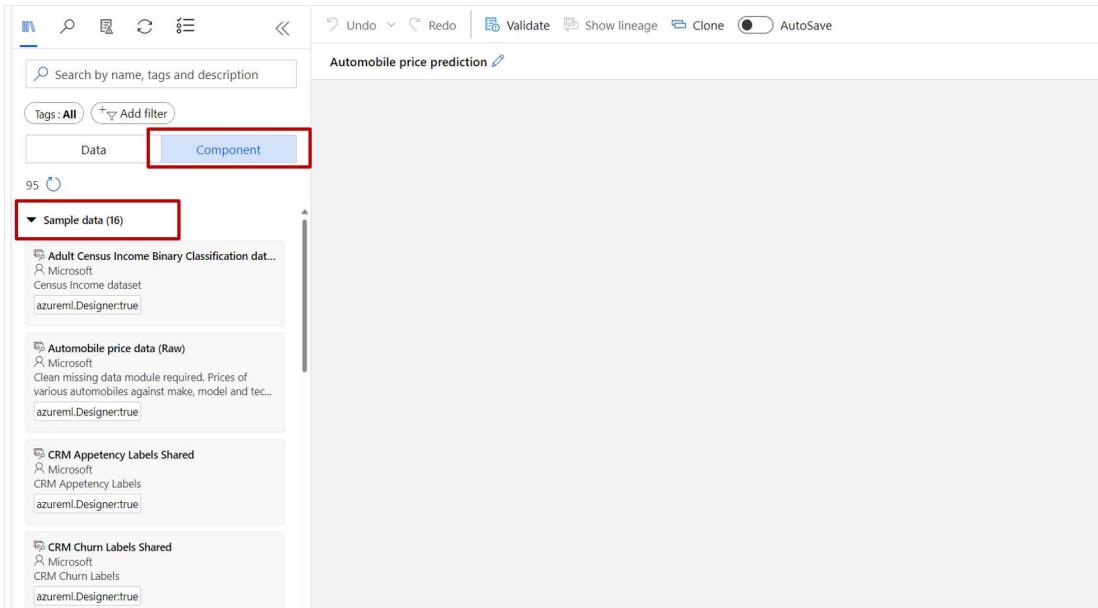
When you use it again after a delay, you might experience approximately five minutes of wait time while it scales back up.

Import data

There are several sample datasets included in the designer for you to experiment with. For this tutorial, use **Automobile price data (Raw)**.

1. To the left of the pipeline canvas is a palette of datasets and components. Select **Component -> Sample data**.

2. Select the dataset **Automobile price data (Raw)**, and drag it onto the canvas.



Visualize the data

You can visualize the data to understand the dataset that you'll use.

1. Right-click the **Automobile price data (Raw)** and select **Preview Data**.
2. Select the different columns in the data window to view information about each one.

Each row represents an automobile, and the variables associated with each automobile appear as columns. There are 205 rows and 26 columns in this dataset.

Prepare data

Datasets typically require some preprocessing before analysis. You might have noticed some missing values when you inspected the dataset. These missing values must be cleaned so that the model can analyze the data correctly.

Remove a column

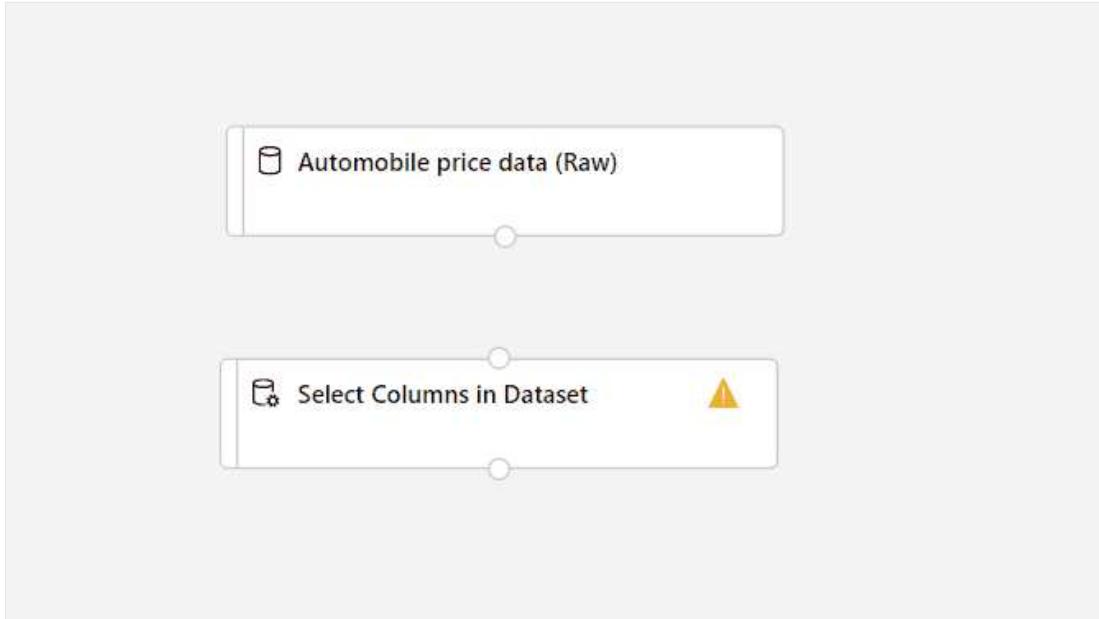
When you train a model, you have to do something about the data that's missing. In this dataset, the **normalized-losses** column is missing many values, so you'll exclude that column from the model altogether.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Select Columns in Dataset** component.

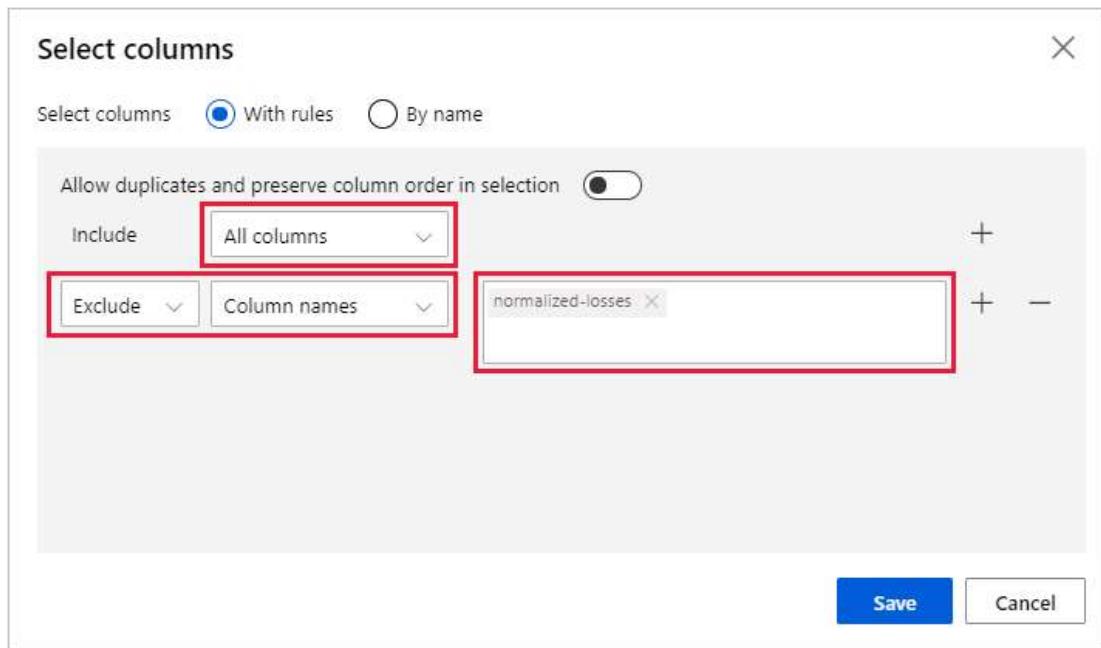
2. Drag the **Select Columns in Dataset** component onto the canvas. Drop the component below the dataset component.
3. Connect the **Automobile price data (Raw)** dataset to the **Select Columns in Dataset** component. Drag from the dataset's output port, which is the small circle at the bottom of the dataset on the canvas, to the input port of **Select Columns in Dataset**, which is the small circle at the top of the component.

 **Tip**

You create a flow of data through your pipeline when you connect the output port of one component to an input port of another.



4. Select the **Select Columns in Dataset** component.
5. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Select Columns in Dataset** component to open the details pane.
6. Select **Edit column** to the right of the pane.
7. Expand the **Column names** drop down next to **Include**, and select **All columns**.
8. Select the **+** to add a new rule.
9. From the drop-down menus, select **Exclude** and **Column names**.
10. Enter *normalized-losses* in the text box.
11. In the lower right, select **Save** to close the column selector.



12. In the **Select Columns in Dataset** component details pane, expand **Node info**.

13. Select the **Comment** text box and enter *Exclude normalized losses*.

Comments will appear on the graph to help you organize your pipeline.

Clean missing data

Your dataset still has missing values after you remove the **normalized-losses** column. You can remove the remaining missing data by using the **Clean Missing Data** component.

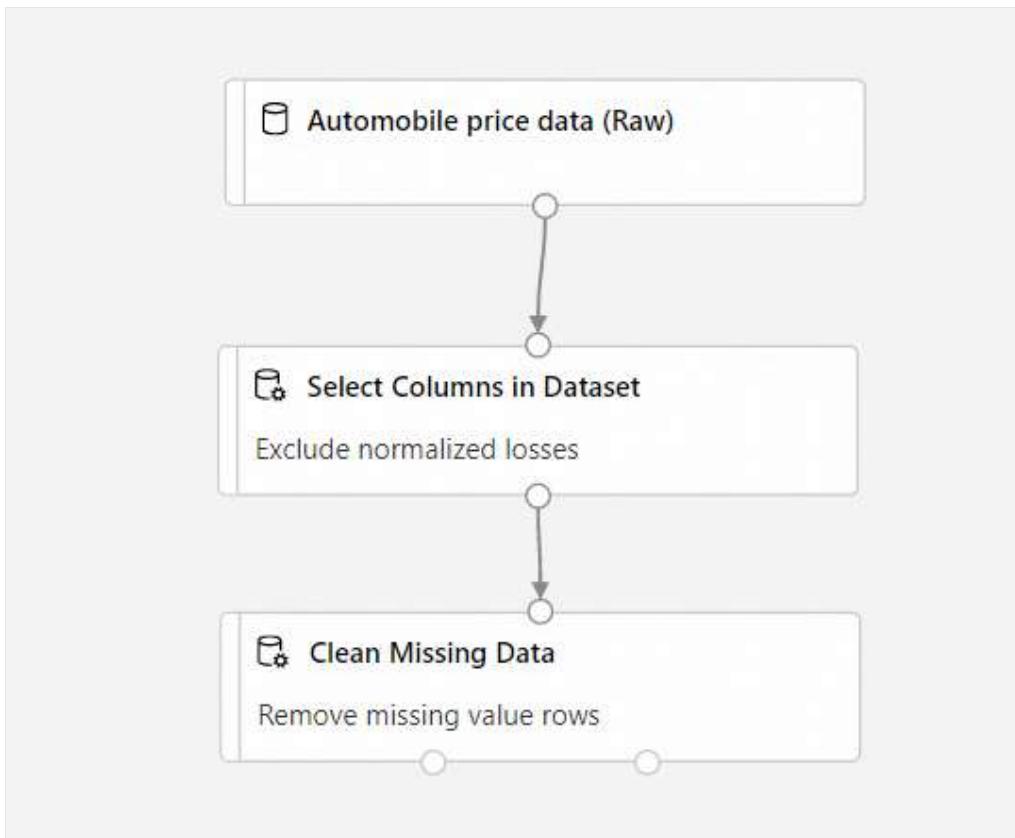
Tip

Cleaning the missing values from input data is a prerequisite for using most of the components in the designer.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Clean Missing Data** component.
2. Drag the **Clean Missing Data** component to the pipeline canvas. Connect it to the **Select Columns in Dataset** component.
3. Select the **Clean Missing Data** component.
4. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Clean Missing Data** component to open the details pane.

5. Select **Edit column** to the right of the pane.
6. In the **Columns to be cleaned** window that appears, expand the drop-down menu next to **Include**. Select, **All columns**
7. Select **Save**
8. In the **Clean Missing Data** component details pane, under **Cleaning mode**, select **Remove entire row**.
9. In the **Clean Missing Data** component details pane, expand **Node info**.
10. Select the **Comment** text box and enter *Remove missing value rows*.

Your pipeline should now look something like this:



Train a machine learning model

Now that you have the components in place to process the data, you can set up the training components.

Because you want to predict price, which is a number, you can use a regression algorithm. For this example, you use a linear regression model.

Split the data

Splitting data is a common task in machine learning. You'll split your data into two separate datasets. One dataset will train the model and the other will test how well the model performed.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Split Data** component.
2. Drag the **Split Data** component to the pipeline canvas.
3. Connect the left port of the **Clean Missing Data** component to the **Split Data** component.

Important

Make sure that the left output port of **Clean Missing Data** connects to **Split Data**. The left port contains the cleaned data. The right port contains the discarded data.

4. Select the **Split Data** component.
5. Click on the arrow icon under **Settings** to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Split Data** component to open the details pane.
6. In the **Split Data** details pane, set the **Fraction of rows in the first output dataset** to 0.7.

This option splits 70 percent of the data to train the model and 30 percent for testing it. The 70 percent dataset will be accessible through the left output port. The remaining data will be available through the right output port.

7. In the **Split Data** details pane, expand **Node info**.
8. Select the **Comment** text box and enter *Split the dataset into training set (0.7) and test set (0.3)*.

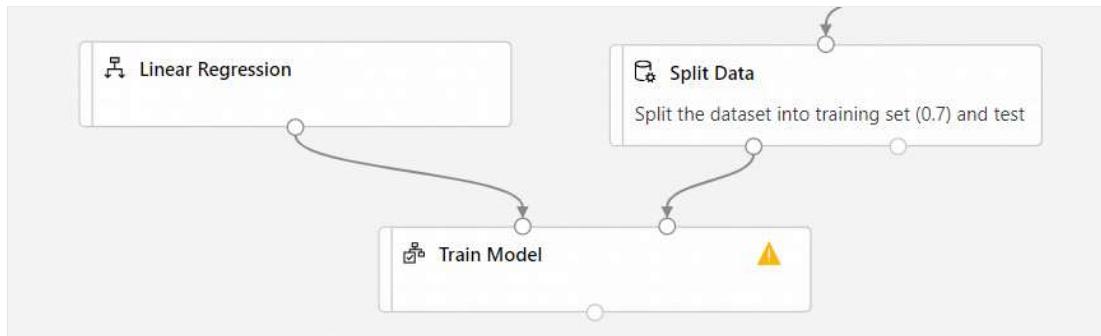
Train the model

Train the model by giving it a dataset that includes the price. The algorithm constructs a model that explains the relationship between the features and the price as presented by the training data.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Linear Regression** component.
2. Drag the **Linear Regression** component to the pipeline canvas.
3. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Train Model** component.
4. Drag the **Train Model** component to the pipeline canvas.
5. Connect the output of the **Linear Regression** component to the left input of the **Train Model** component.
6. Connect the training data output (left port) of the **Split Data** component to the right input of the **Train Model** component.

ⓘ Important

Make sure that the left output port of **Split Data** connects to **Train Model**. The left port contains the training set. The right port contains the test set.

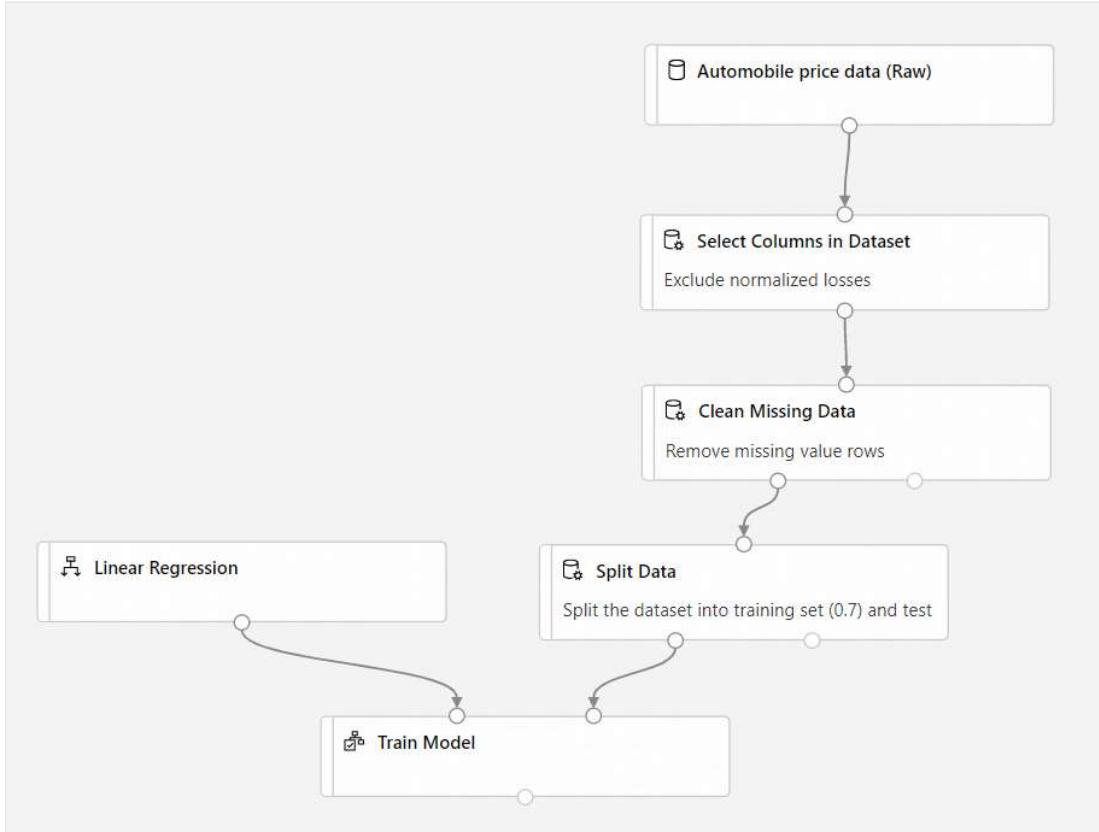


7. Select the **Train Model** component.
8. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Train Model** component to open the details pane.
9. Select **Edit column** to the right of the pane.
10. In the **Label column** window that appears, expand the drop-down menu and select **Column names**.
11. In the text box, enter *price* to specify the value that your model is going to predict.

ⓘ Important

Make sure you enter the column name exactly. Do not capitalize **price**.

Your pipeline should look like this:



Add the Score Model component

After you train your model by using 70 percent of the data, you can use it to score the other 30 percent to see how well your model functions.

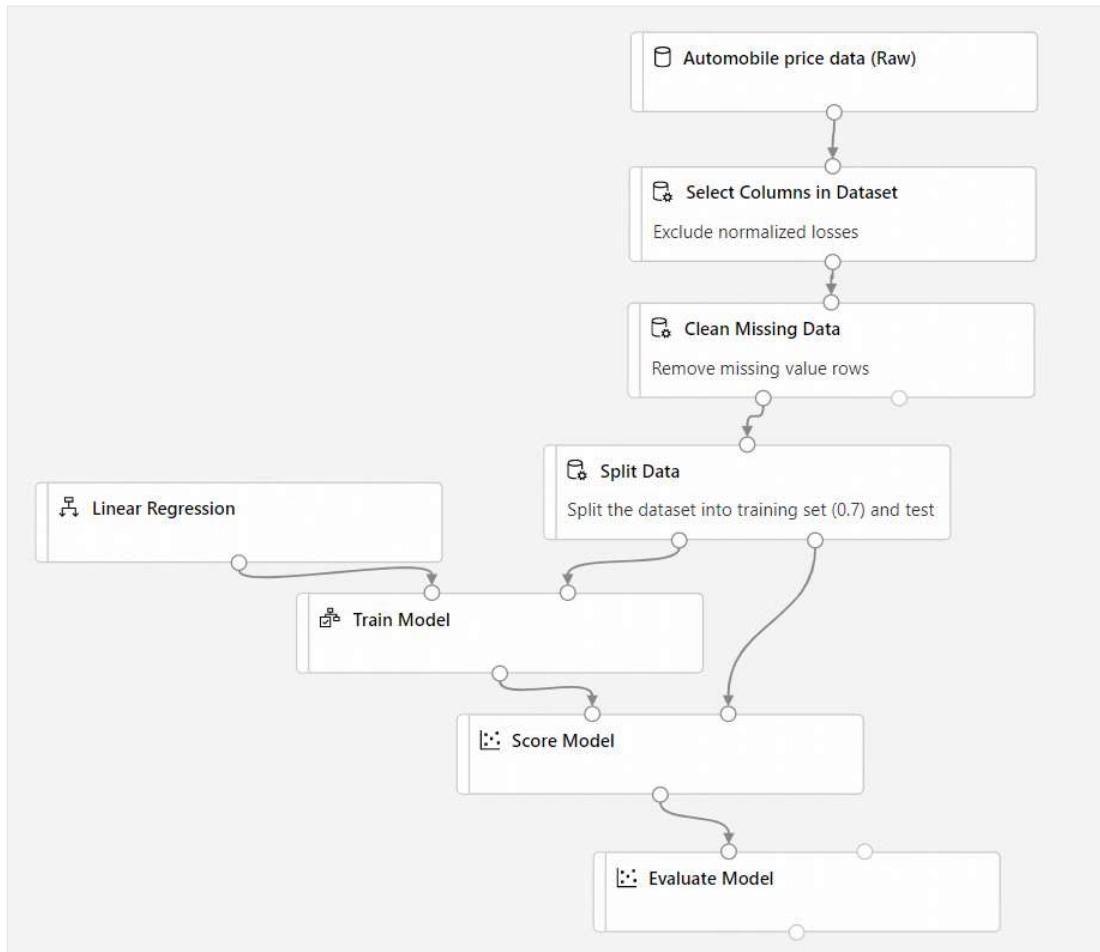
1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Score Model** component.
2. Drag the **Score Model** component to the pipeline canvas.
3. Connect the output of the **Train Model** component to the left input port of **Score Model**. Connect the test data output (right port) of the **Split Data** component to the right input port of **Score Model**.

Add the Evaluate Model component

Use the **Evaluate Model** component to evaluate how well your model scored the test dataset.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Evaluate Model** component.
2. Drag the **Evaluate Model** component to the pipeline canvas.
3. Connect the output of the **Score Model** component to the left input of **Evaluate Model**.

The final pipeline should look something like this:



Submit the pipeline

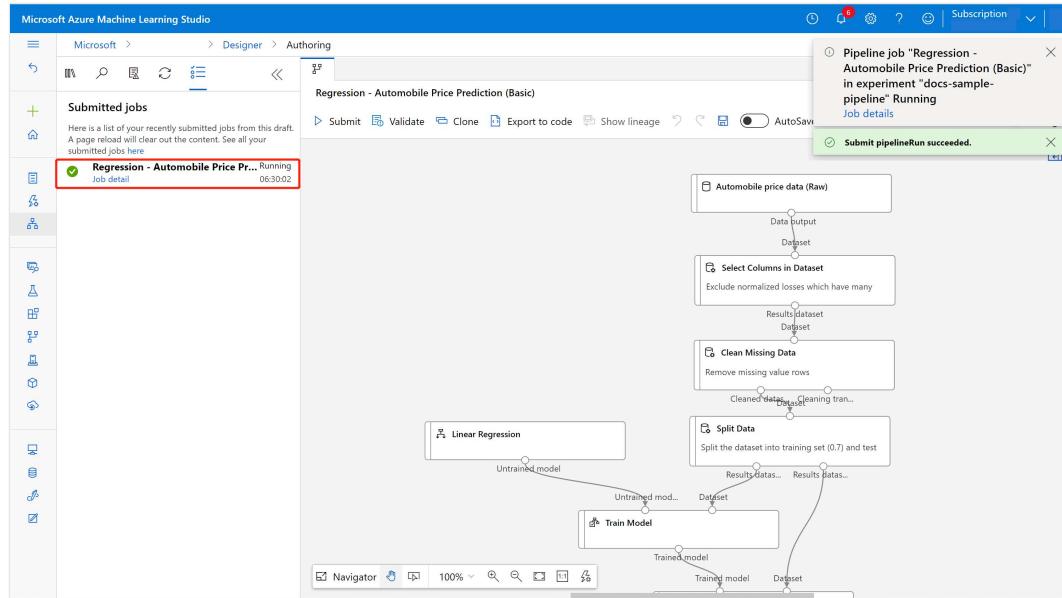
Now that your pipeline is all setup, you can submit a pipeline job to train your machine learning model. You can submit a valid pipeline job at any point, which can be used to review changes to your pipeline during development.

1. At the top of the canvas, select **Submit**.
2. In the **Set up pipeline job** dialog box, select **Create new**.

(!) Note

Experiments group similar pipeline jobs together. If you run a pipeline multiple times, you can select the same experiment for successive jobs.

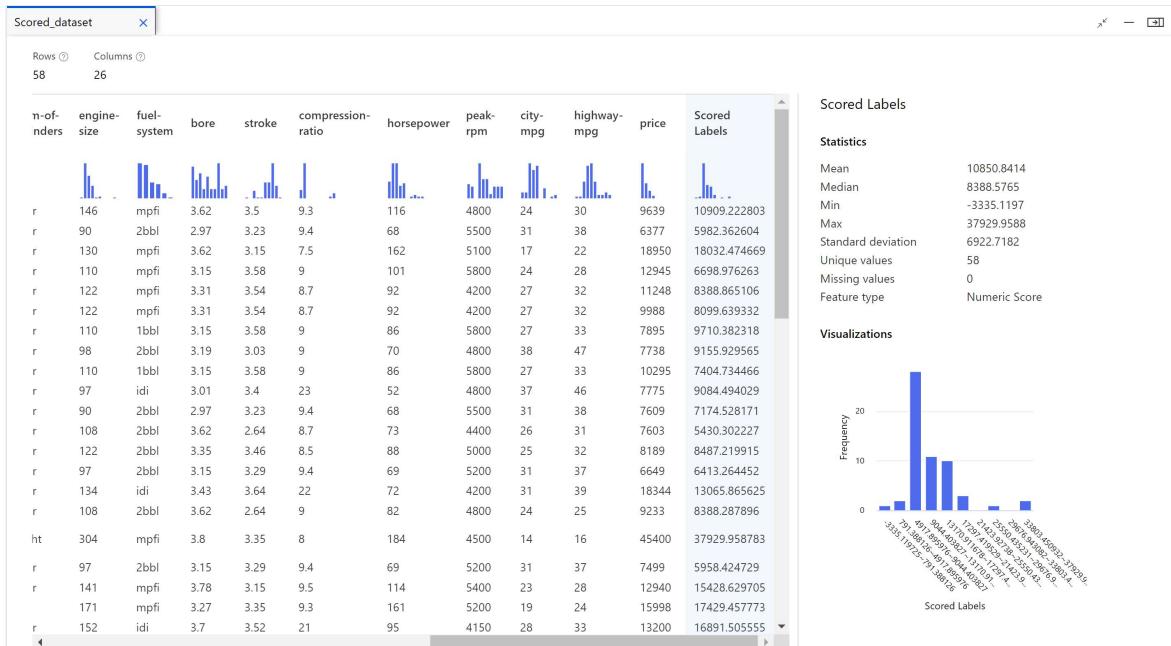
- a. For **New experiment Name**, enter **Tutorial-CarPrices**.
- b. Select **Submit**.
- c. You'll see a submission list in the left pane of the canvas, and a notification will pop up at the top right corner of the page. You can select the **Job detail** link to go to job detail page for debugging.



If this is the first job, it may take up to 20 minutes for your pipeline to finish running. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline jobs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each component to further improve efficiency.

View scored labels

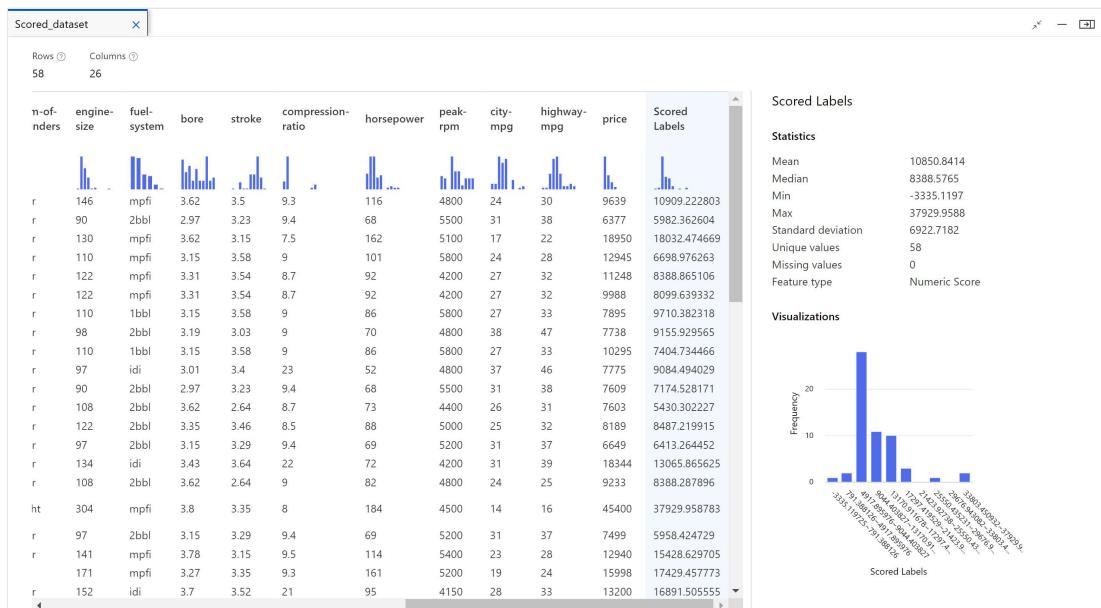
In the job detail page, you can check the pipeline job status, results and logs.



After the job completes, you can view the results of the pipeline job. First, look at the predictions generated by the regression model.

1. Right-click the **Score Model** component, and select **Preview data > Scored dataset** to view its output.

Here you can see the predicted prices and the actual prices from the testing data.



Evaluate models

Use the **Evaluate Model** to see how well the trained model performed on the test dataset.

1. Right-click the **Evaluate Model** component and select **Preview data > Evaluation results** to view its output.

The following statistics are shown for your model:

- **Mean Absolute Error (MAE)**: The average of absolute errors. An error is the difference between the predicted value and the actual value.
- **Root Mean Squared Error (RMSE)**: The square root of the average of squared errors of predictions made on the test dataset.
- **Relative Absolute Error**: The average of absolute errors relative to the absolute difference between actual values and the average of all actual values.
- **Relative Squared Error**: The average of squared errors relative to the squared difference between the actual values and the average of all actual values.
- **Coefficient of Determination**: Also known as the R squared value, this statistical metric indicates how well a model fits the data.

For each of the error statistics, smaller is better. A smaller value indicates that the predictions are closer to the actual values. For the coefficient of determination, the closer its value is to one (1.0), the better the predictions.

Clean up resources

Skip this section if you want to continue on with part 2 of the tutorial, [deploying models](#).

Important

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.

The screenshot shows the Microsoft Azure Resource Groups blade. On the left, there's a list of resource groups under the heading 'Resource groups'. One resource group, 'my-rg', is highlighted with a red box. On the right, the details for 'my-rg' are shown, including its subscription information ('Subscription (change) : documentationteam'), subscription ID ('Subscription ID : aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa'), and tags ('Tags (change) : Click here to add tags'). Below this, there's a list of resources within the group, including 'my-ws', 'myws22', 'myws70', and 'myws74', each with a checkbox next to it.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the **Delete** button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

The screenshot shows the 'Compute' section of the Azure Machine Learning studio. On the left, there's a sidebar with 'Compute' highlighted. The main area displays a table of 'Inference Clusters'. One cluster, 'aks-compute', is selected and has a red box around its 'Delete' button.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the 'Datasets' page in the Azure Machine Learning studio. The 'Datasets' tab is selected in the sidebar. A specific dataset, 'TD-Sample_1: Regression - Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1', is selected and its details are shown. The 'Unregister' button is highlighted with a red box.

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

In part two, you'll learn how to deploy your model as a real-time endpoint.

[Continue to deploying models](#)

Additional resources

Documentation

[Evaluate Model: Component Reference - Azure Machine Learning](#)

Learn how to use the Evaluate Model component in Azure Machine Learning to measure the accuracy of a trained model.

[Example pipelines & datasets for the designer - Azure Machine Learning](#)

Learn how to use samples in Azure Machine Learning designer to jumps-start your machine learning pipelines.

[Linear Regression: Component Reference - Azure Machine Learning](#)

Learn how to use the Linear Regression component in Azure Machine Learning to create a linear regression model for use in a pipeline.

[What is the Azure Machine Learning designer? - Azure Machine Learning](#)

Learn what the Azure Machine Learning designer is and what tasks you can use it for. The drag-and-drop UI enables model training and deployment.

[Avoid overfitting & imbalanced data with AutoML - Azure Machine Learning](#)

Identify and manage common pitfalls of ML models with Azure Machine Learning's automated machine learning solutions.

[Two-Class Support Vector Machine: Component Reference - Azure Machine Learning](#)

Learn how to use the Two-Class Support Vector Machine component in Azure Machine Learning to create a binary classifier.

[What is Responsible AI - Azure Machine Learning](#)

Learn what Responsible AI is and how to use it with Azure Machine Learning to understand models, protect data, and control the model lifecycle.

[Train Model: Component Reference - Azure Machine Learning](#)

Learn how to use the **Train Model** component in Azure Machine Learning to train a classification or regression model.

[Show 5 more](#)

Training

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Tutorial: Designer - deploy a machine learning model

Article • 10/21/2022 • 8 minutes to read

Use the designer to deploy a machine learning model to predict the price of cars. This tutorial is part two of a two-part series.

In [part one of the tutorial](#) you trained a linear regression model on car prices. In part two, you deploy the model to give others a chance to use it. In this tutorial, you:

- ✓ Create a real-time inference pipeline.
- ✓ Create an inferencing cluster.
- ✓ Deploy the real-time endpoint.
- ✓ Test the real-time endpoint.

Prerequisites

Complete [part one of the tutorial](#) to learn how to train and score a machine learning model in the designer.

Important

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Create a real-time inference pipeline

To deploy your pipeline, you must first convert the training pipeline into a real-time inference pipeline. This process removes training components and adds web service inputs and outputs to handle requests.

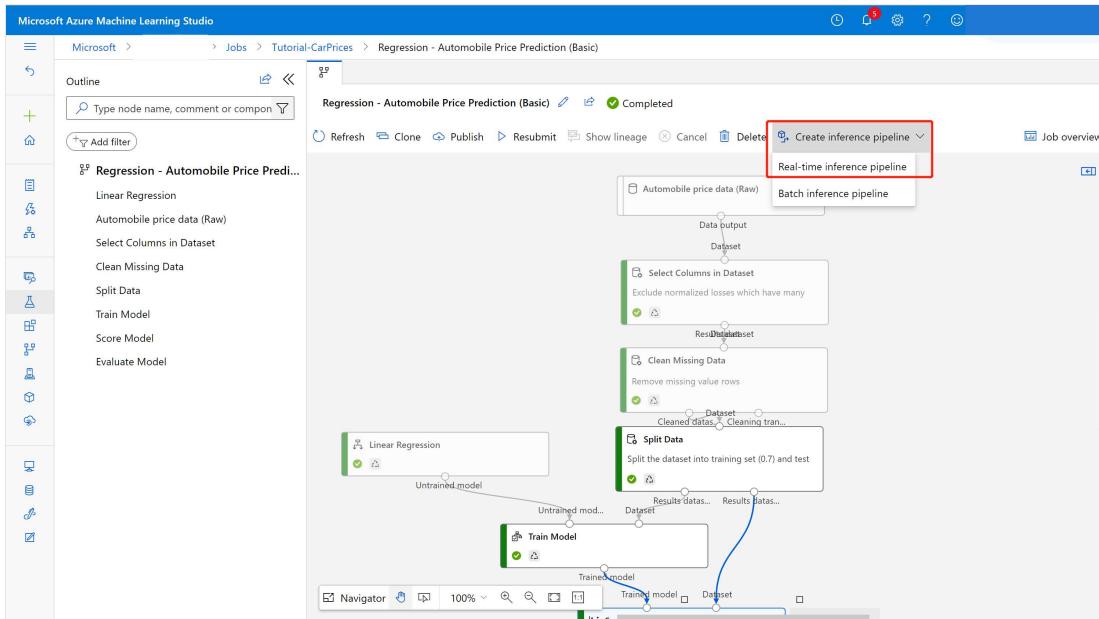
Note

Create inference pipeline only supports training pipelines which contain only the designer built-in components and must have a component like **Train Model** which

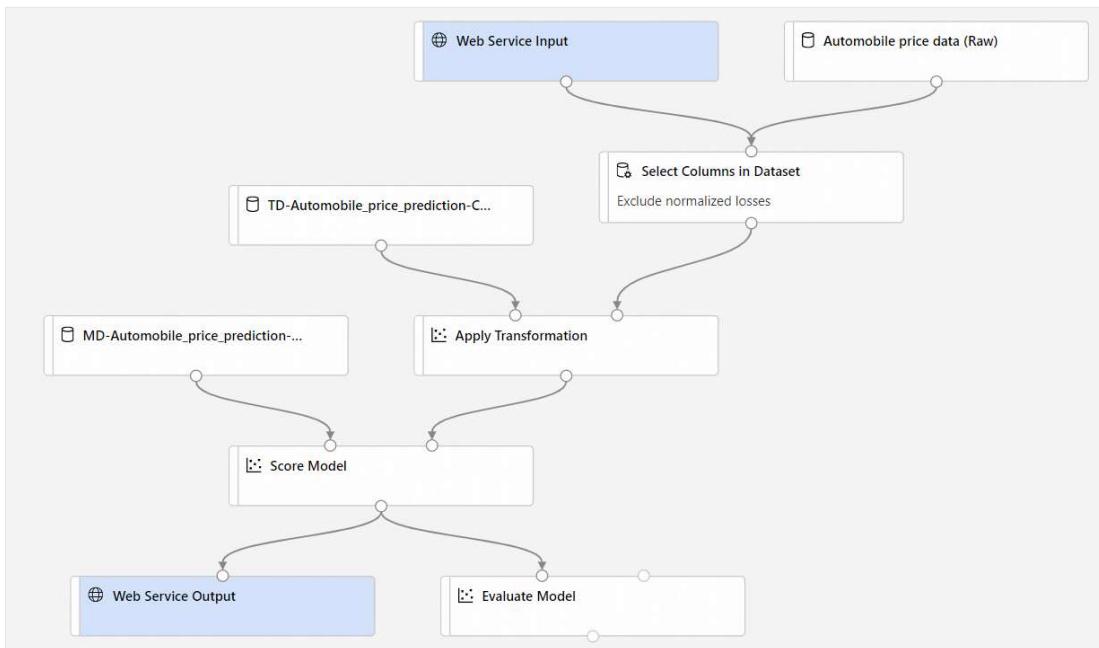
outputs the trained model.

Create a real-time inference pipeline

1. On pipeline job detail page, above the pipeline canvas, select **Create inference pipeline** > **Real-time inference pipeline**.



Your new pipeline will now look like this:



When you select **Create inference pipeline**, several things happen:

- The trained model is stored as a **Dataset** component in the component palette. You can find it under **My Datasets**.

- Training components like **Train Model** and **Split Data** are removed.
- The saved trained model is added back into the pipeline.
- **Web Service Input** and **Web Service Output** components are added. These components show where user data enters the pipeline and where data is returned.

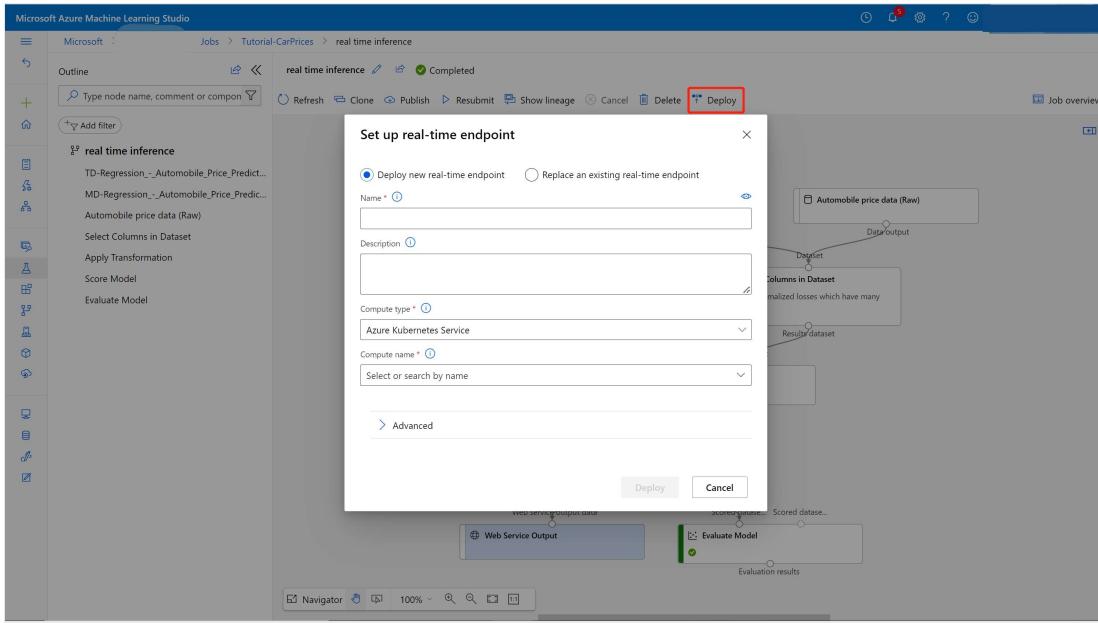
ⓘ Note

By default, the **Web Service Input** will expect the same data schema as the component output data which connects to the same downstream port as it. In this sample, **Web Service Input** and **Automobile price data (Raw)** connect to the same downstream component, hence **Web Service Input** expect the same data schema as **Automobile price data (Raw)** and target variable column `price` is included in the schema. However, usually When you score the data, you won't know the target variable values. For such case, you can remove the target variable column in the inference pipeline using **Select Columns in Dataset** component. Make sure that the output of **Select Columns in Dataset** removing target variable column is connected to the same port as the output of the **Web Service Input** component.

2. Select **Submit**, and use the same compute target and experiment that you used in part one.

If this is the first job, it may take up to 20 minutes for your pipeline to finish running. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline jobs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each component to further improve efficiency.

3. Go to the real-time inference pipeline job detail by selecting **Job detail** link in the left pane.
4. Select **Deploy** in the job detail page.



Create an inferencing cluster

In the dialog box that appears, you can select from any existing Azure Kubernetes Service (AKS) clusters to deploy your model to. If you don't have an AKS cluster, use the following steps to create one.

1. Select **Compute** in the dialog box that appears to go to the **Compute** page.
2. On the navigation ribbon, select **Inference Clusters > + New**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar is open, revealing various service categories like Notebooks, Automated ML, Designer, Data, Jobs, Components, Pipelines, Environments, Models, Endpoints, and Compute. The 'Compute' item is highlighted with a red box. The main content area is titled 'Compute' and shows the 'Inference clusters' tab selected. Below it, there's a toolbar with buttons for '+ New', Refresh, Delete, Detach, Edit columns, and Reset view. A search bar is also present. A table lists a single entry: 'aks-cluster' with a status of 'Creating' and a type of 'Kubernetes service'. The entire interface has a clean, modern design with a light blue header.

3. In the inference cluster pane, configure a new Kubernetes Service.

4. Enter *aks-compute* for the **Compute name**.

5. Select a nearby region that's available for the **Region**.

6. Select **Create**.

ⓘ Note

It takes approximately 15 minutes to create a new AKS service. You can check the provisioning state on the **Inference Clusters** page.

Deploy the real-time endpoint

After your AKS service has finished provisioning, return to the real-time inferencing pipeline to complete deployment.

1. Select **Deploy** above the canvas.

2. Select **Deploy new real-time endpoint**.

3. Select the AKS cluster you created.

Set up real-time endpoint

Deploy new real-time endpoint Replace an existing real-time endpoint

Name *

Description

Compute type *

Compute name *

> Advanced

Deploy **Cancel**

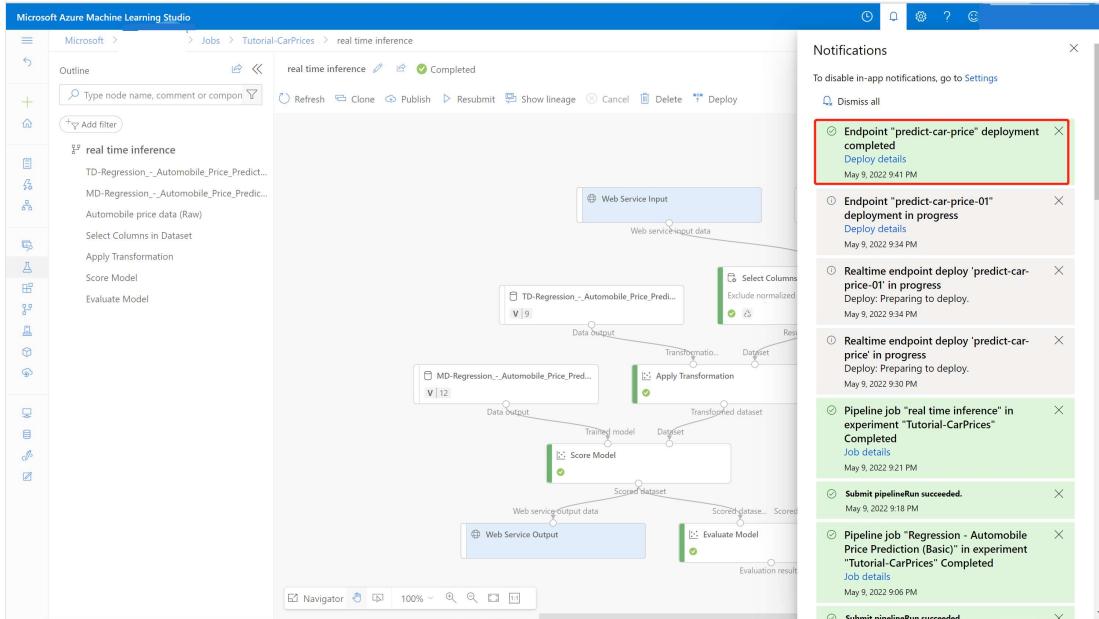
You can also change **Advanced** setting for your real-time endpoint.

Advanced setting	Description
Enable Application Insights diagnostics and data collection	Whether to enable Azure Application Insights to collect data from the deployed endpoints. By default: false.
Scoring timeout	A timeout in milliseconds to enforce for scoring calls to the web service. By default: 60000.
Auto scale enabled	Whether to enable autoscaling for the web service. By default: true.
Min replicas	The minimum number of containers to use when autoscaling this web service. By default: 1.

Advanced setting	Description
Max replicas	The maximum number of containers to use when autoscaling this web service. By default: 10.
Target utilization	The target utilization (in percent out of 100) that the autoscaler should attempt to maintain for this web service. By default: 70.
Refresh period	How often (in seconds) the autoscaler attempts to scale this web service. By default: 1.
CPU reserve capacity	The number of CPU cores to allocate for this web service. By default: 0.1.
Memory reserve capacity	The amount of memory (in GB) to allocate for this web service. By default: 0.5.

4. Select Deploy.

A success notification from the notification center appears after deployment finishes. It might take a few minutes.



Tip

You can also deploy to **Azure Container Instance (ACI)** if you select **Azure Container Instance** for **Compute type** in the real-time endpoint setting box. Azure

Container Instance is used for testing or development. Use ACI for low-scale CPU-based workloads that require less than 48 GB of RAM.

Test the real-time endpoint

After deployment finishes, you can view your real-time endpoint by going to the **Endpoints** page.

1. On the **Endpoints** page, select the endpoint you deployed.

In the **Details** tab, you can see more information such as the REST URI, Swagger definition, status, and tags.

In the **Consume** tab, you can find sample consumption code, security keys, and set authentication methods.

In the **Deployment logs** tab, you can find the detailed deployment logs of your real-time endpoint.

2. To test your endpoint, go to the **Test** tab. From here, you can enter test data and select **Test** verify the output of your endpoint.

Update the real-time endpoint

You can update the online endpoint with new model trained in the designer. On the online endpoint detail page, find your previous training pipeline job and inference pipeline job.

1. You can directly find and modify your training pipeline draft in the designer homepage.
Or you can open the training pipeline job link and then clone it into a new pipeline draft to continue editing.

Microsoft Azure Machine Learning Studio

Endpoints > predict-car-price

predict-car-price

Details Test Consume Deployment logs

Attributes

Service ID
predict-car-price

Description
--

Deployment state
Healthy

Operation state
Succeeded

Compute type
Container instance

Created by
amilstudio

Model ID
amilstudio-predict-car-price:1

Created on
May 9, 2022 9:30 PM

Last updated on
May 9, 2022 9:30 PM

Image ID
--

REST endpoint

Key-based authentication enabled
true

Swagger URI
eastus.azurecontainer.io/swagger.json

Tags

CreatedByAMLStudio
true

Properties

Real-time inference pipeline job

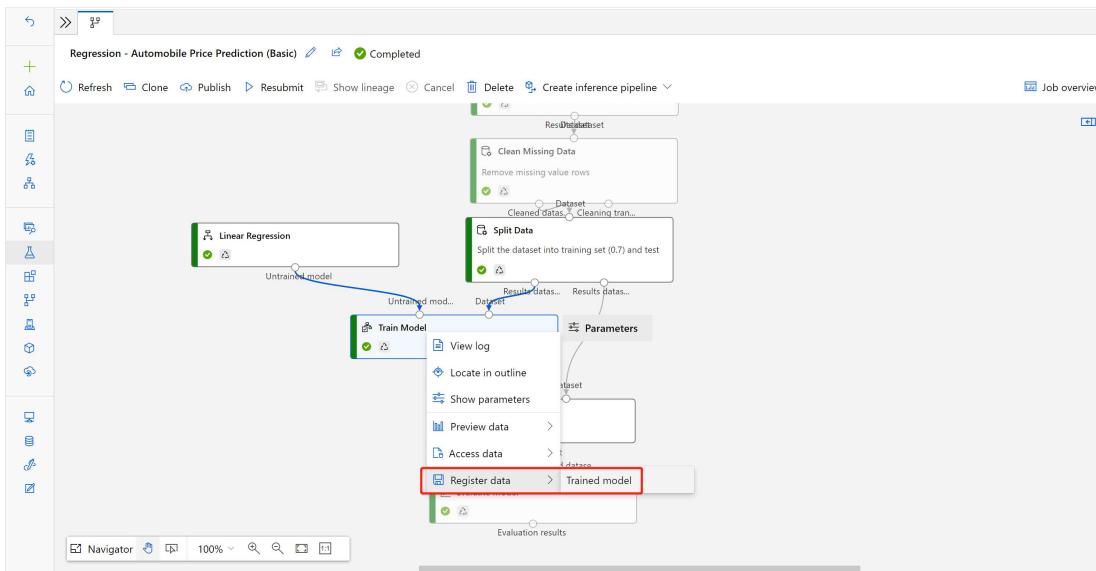
Training pipeline job

hasInferenceSchema
True

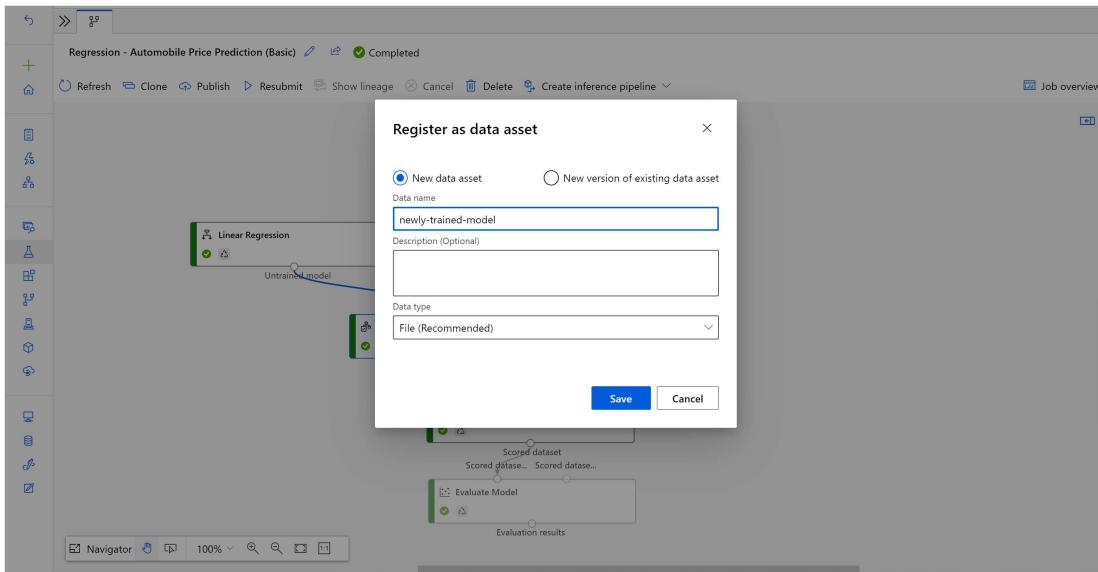
hasHttps
False

2. After you submit the modified training pipeline, go to the job detail page.

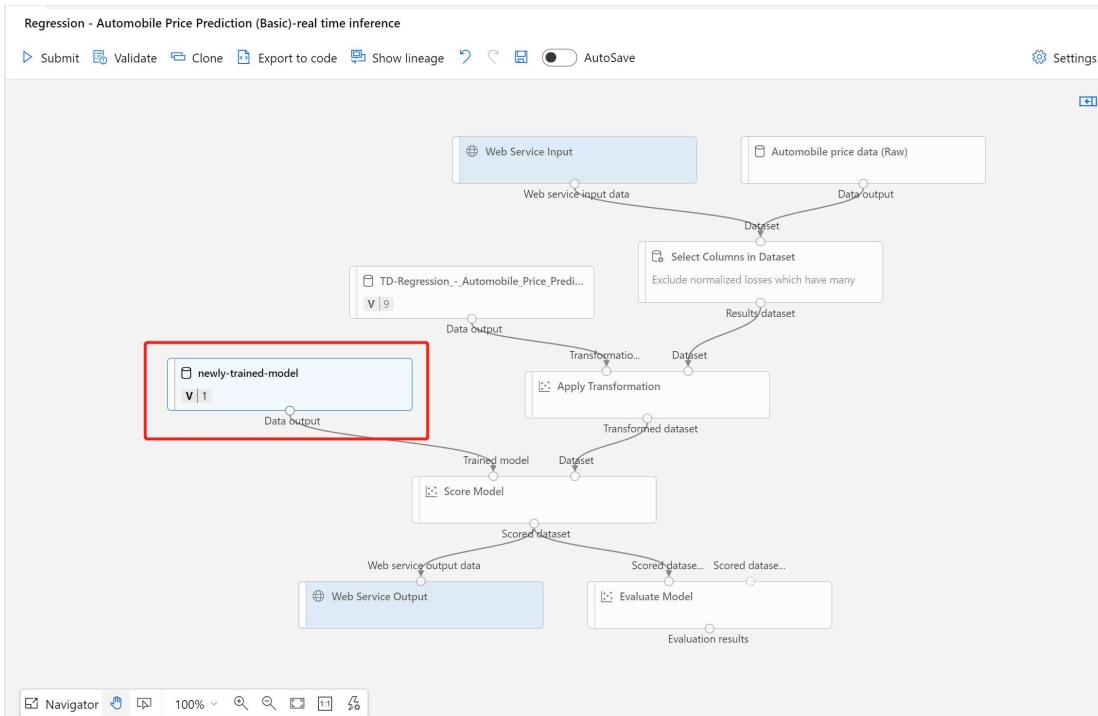
3. When the job completes, right click **Train Model** and select **Register data**.



Input name and select **File** type.



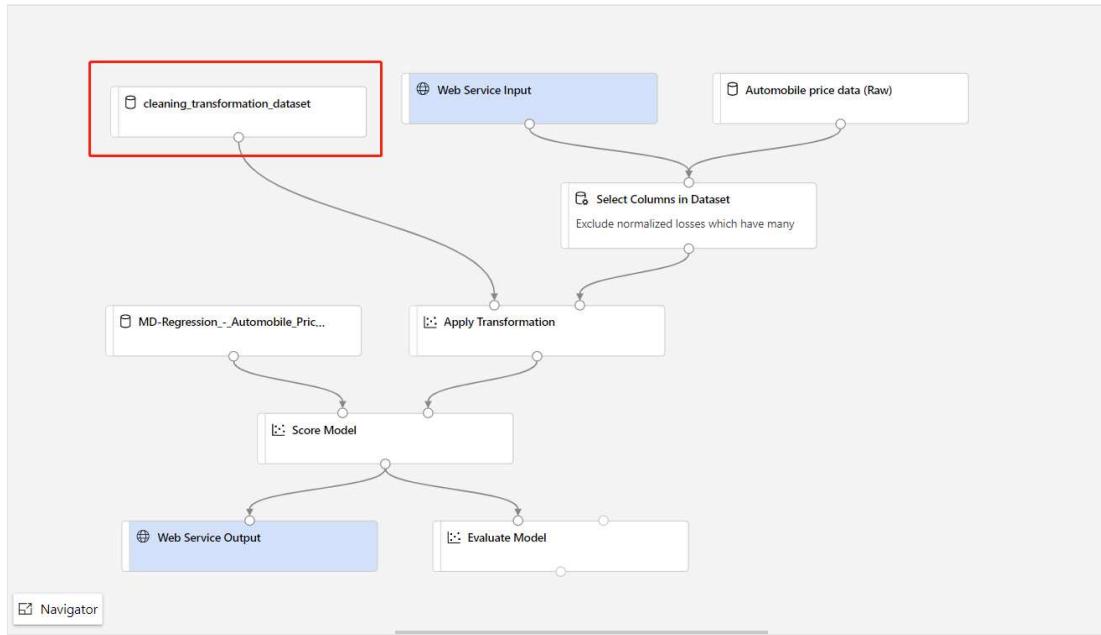
4. After the dataset registers successfully, open your inference pipeline draft, or clone the previous inference pipeline job into a new draft. In the inference pipeline draft, replace the previous trained model shown as **MD-XXXX** node connected to the **Score Model** component with the newly registered dataset.



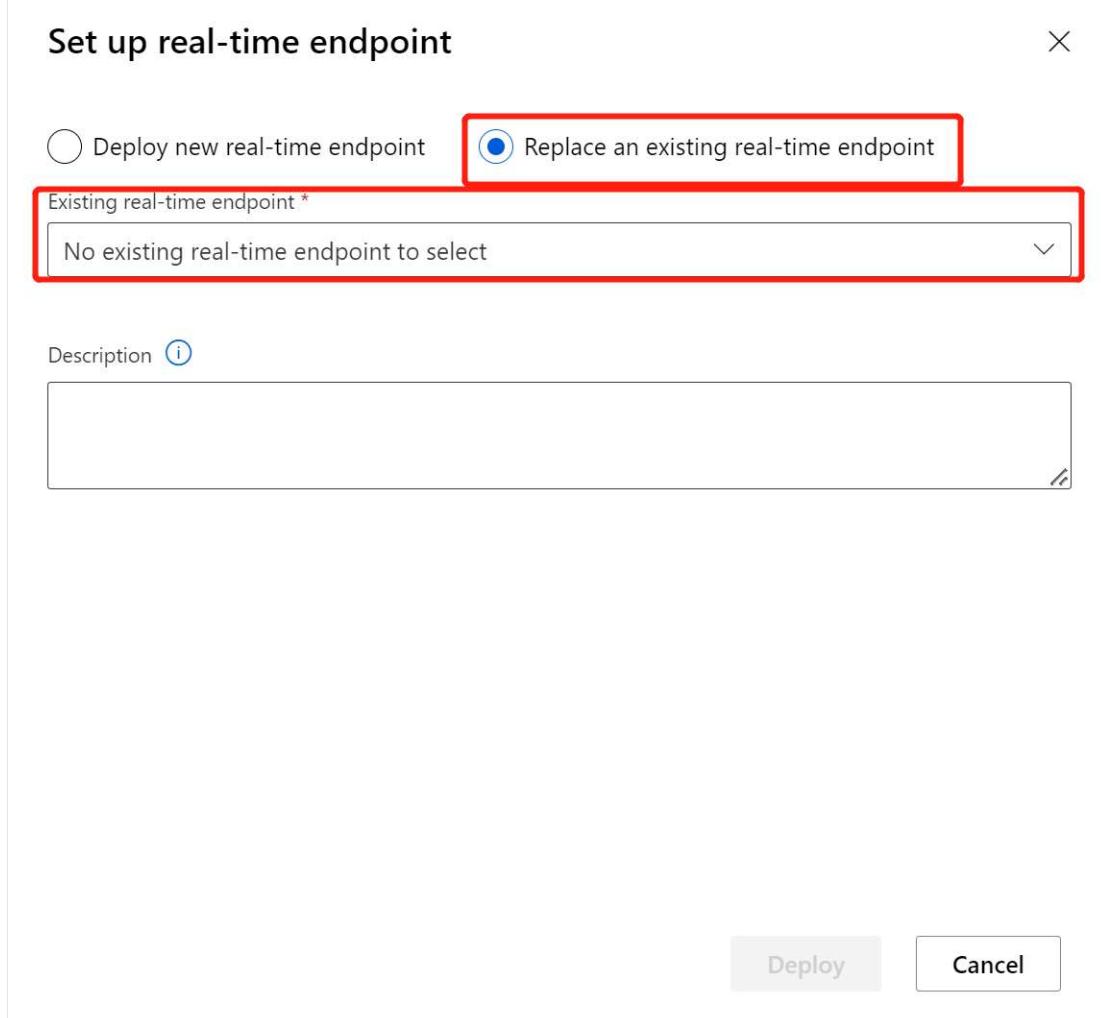
5. If you need to update the data preprocessing part in your training pipeline, and would like to update that into the inference pipeline, the processing is similar as steps above.

You just need to register the transformation output of the transformation component as dataset.

Then manually replace the TD- component in inference pipeline with the registered dataset.



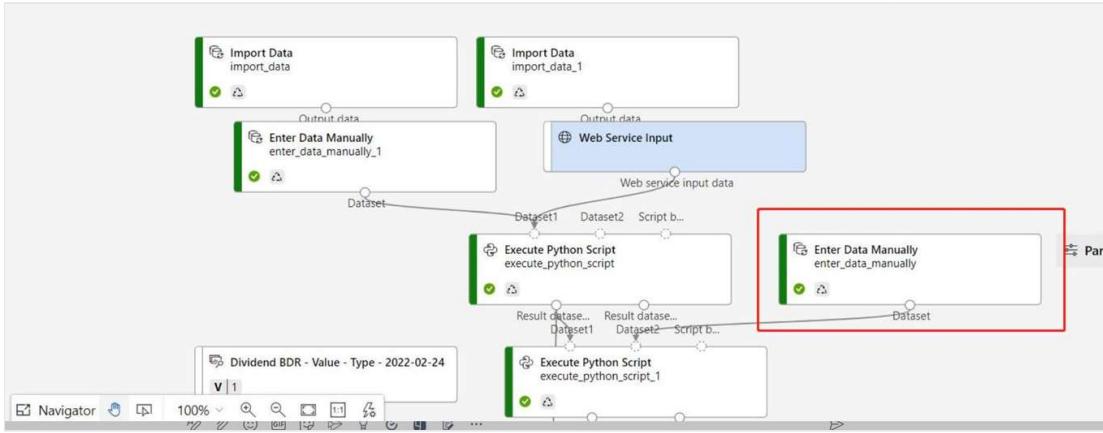
6. After modifying your inference pipeline with the newly trained model or transformation, submit it. When the job is completed, deploy it to the existing online endpoint deployed previously.



Limitations

- Due to datastore access limitation, if your inference pipeline contains **Import Data** or **Export Data** component, they'll be auto-removed when deploy to real-time endpoint.
- If you have datasets in the real-time inference pipeline and want to deploy it to real-time endpoint, currently this flow only supports datasets registered from **Blob** datastore. If you want to use datasets from other type datastores, you can use Select Column to connect with your initial dataset with settings of selecting all columns, register the outputs of Select Column as File dataset and then replace the initial dataset in the real-time inference pipeline with this newly registered dataset.
- If your inference graph contains "Enter Data Manually" component which is not connected to the same port as "Web service Input" component, the "Enter Data Manually" component will not be executed during HTTP call processing. A workaround is to register the outputs of that "Enter Data Manually" component as

dataset, then in the inference pipeline draft, replace the "Enter Data Manually" component with the registered dataset.



Clean up resources

Important

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.

The screenshot shows the Microsoft Azure portal's 'Resource groups' page. The 'my-rg' resource group is selected and highlighted with a red box. On the right side of the screen, there is a detailed view of the resource group, including its subscription information ('Subscription (change) : documentationteam'), subscription ID ('Subscription ID : aaaaaaa-aaa-aaa-aaa-aaaaaaaaaaaa'), and tags ('Tags (change) : Click here to add tags'). The 'Delete resource group' button is also highlighted with a red box. The page includes standard Azure navigation and search features.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the **Delete** button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

Name	Type	Provis...	Cre...	Virtua...
aks-compute	Kubernetes Serv...	Succe...	10/17/...	STAN...

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with navigation links: New, Home, Author, Notebooks, Automated ML, Designer, Assets, Datasets (which is selected and highlighted with a red box), Experiments, Pipelines, Models, Endpoints, Manage, Compute, Environments, Datastores, and Data labeling. The main content area has a title bar: TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1 and a dropdown for Version 1 (latest). Below the title are tabs: Details, Explore, Models, Datasheet, with Details being the active tab. Under Details, there are buttons for Refresh, Generate profile, Unregister (which is highlighted with a red box), and New version. The main content is divided into sections: Attributes, Properties, Description, Datastore, Relative path, Profile, Files in dataset, Current version, and Latest version. The Description section contains: This is a dataset promoted by inference graph generation automatically on 11/12/...; Datastore: workspaceblobstore; Relative path: azureml/4393076b-19ff-4e41-81d9-a1146d905696/Cleaning_transformation; Profile: No profile generated; Files in dataset: 4; Current version: 1; Latest version: 1. To the right of these sections are Tags (CreatedByAMLStudio, true) and Sample usage, which contains a code snippet:

```
# azureml-core of version 1.0.72 or higher is required
from azureml.core import Workspace, Dataset

subscription_id = 'ee85ed72-2b26-48f6-a0e8-cb5bcf98fb9'
resource_group = 'test-like'
workspace_name = 'like_test'

workspace = Workspace(subscription_id, resource_group, workspace_name)

dataset = Dataset.get_by_name(workspace, name='TD-Sample_1:_Regression_-_Aut')
dataset.download(target_path='.', overwrite=False)
```

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

In this tutorial, you learned the key steps in how to create, deploy, and consume a machine learning model in the designer. To learn more about how you can use the designer see the following links:

- [Designer samples](#): Learn how to use the designer to solve other types of problems.
- [Use Azure Machine Learning studio in an Azure virtual network](#).

Train an image classification TensorFlow model using the Azure Machine Learning Visual Studio Code Extension (preview)

Article • 02/24/2023 • 5 minutes to read

APPLIES TO:  Azure CLI ml extension v2 (current)

Learn how to train an image classification model to recognize hand-written numbers using TensorFlow and the Azure Machine Learning Visual Studio Code Extension.

In this tutorial, you learn the following tasks:

- ✓ Understand the code
- ✓ Create a workspace
- ✓ Create a GPU cluster for training
- ✓ Train a model

Prerequisites

- Azure subscription. If you don't have one, sign up to try the [free or paid version of Azure Machine Learning](#). If you're using the free subscription, only CPU clusters are supported.
- Install [Visual Studio Code](#), a lightweight, cross-platform code editor.
- Azure Machine Learning Studio Visual Studio Code extension. For install instructions see the [Setup Azure Machine Learning Visual Studio Code extension guide](#)
- CLI (v2). For installation instructions, see [Install, set up, and use the CLI \(v2\)](#)
- Clone the community driven repository

Bash

```
git clone https://github.com/Azure/azureml-examples.git
```

Understand the code

The code for this tutorial uses TensorFlow to train an image classification machine learning model that categorizes handwritten digits from 0-9. It does so by creating a

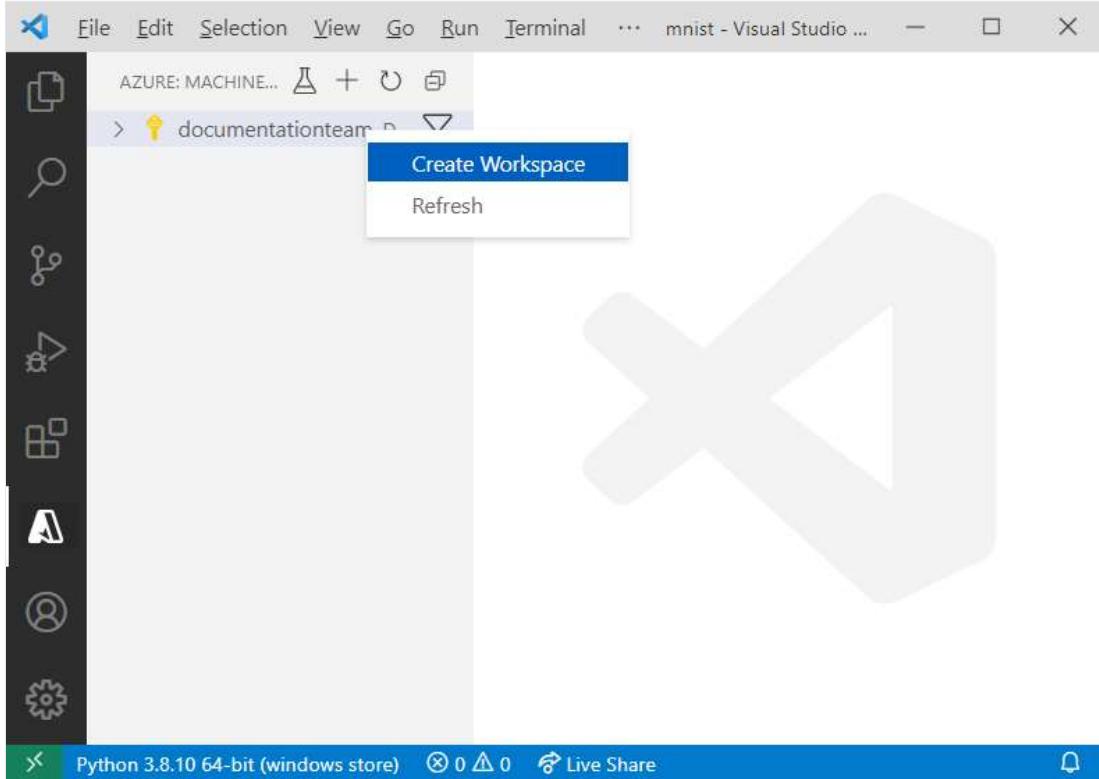
neural network that takes the pixel values of 28 px x 28 px image as input and outputs a list of 10 probabilities, one for each of the digits being classified. Below is a sample of what the data looks like.

8	9	4	7	9	2	9	4	7	8	4	8	8	2	3	2	4	6	1	1	3	8	8	1	3	1	8	4	1	8
8	9	4	7	9	2	9	4	7	8	4	8	8	2	3	2	4	6	1	1	3	8	8	1	3	1	8	4	1	8

Create a workspace

The first thing you have to do to build an application in Azure Machine Learning is to create a workspace. A workspace contains the resources to train models as well as the trained models themselves. For more information, see [what is a workspace](#).

1. Open the `azureml-examples/cli/jobs/single-step/tensorflow/mnist` directory from the community driven repository in Visual Studio Code.
2. On the Visual Studio Code activity bar, select the **Azure** icon to open the Azure Machine Learning view.
3. In the Azure Machine Learning view, right-click your subscription node and select **Create Workspace**.



4. A specification file appears. Configure the specification file with the following options.

```
yml
```

```
$schema: https://azuremlschemas.azureedge.net/latest/workspace.schema.json
name: TeamWorkspace
location: WestUS2
display_name: team-ml-workspace
description: A workspace for training machine learning models
tags:
  purpose: training
  team: ml-team
```

The specification file creates a workspace called `TeamWorkspace` in the `WestUS2` region. The rest of the options defined in the specification file provide friendly naming, descriptions, and tags for the workspace.

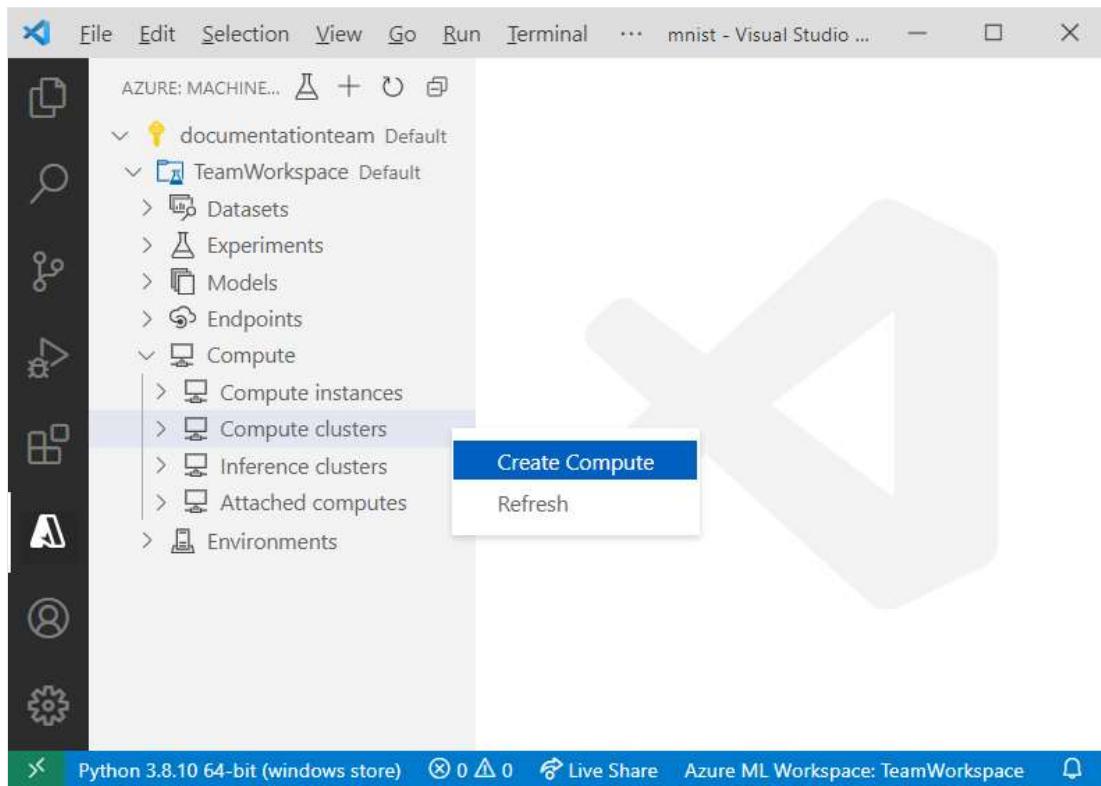
5. Right-click the specification file and select **AzureML: Execute YAML**. Creating a resource uses the configuration options defined in the YAML specification file and submits a job using the CLI (v2). At this point, a request to Azure is made to create a new workspace and dependent resources in your account. After a few minutes, the new workspace appears in your subscription node.
6. Set `TeamWorkspace` as your default workspace. Doing so places resources and jobs you create in the workspace by default. Select the **Set Azure Machine Learning Workspace** button on the Visual Studio Code status bar and follow the prompts to set `TeamWorkspace` as your default workspace.

For more information on workspaces, see [how to manage resources in VS Code](#).

Create a GPU cluster for training

A compute target is the computing resource or environment where you run training jobs. For more information, see the [Azure Machine Learning compute targets documentation](#).

1. In the Azure Machine Learning view, expand your workspace node.
2. Right-click the **Compute clusters** node inside your workspace's **Compute** node and select **Create Compute**



3. A specification file appears. Configure the specification file with the following options.

```
yml

$schema:
https://azurermschemas.azureedge.net/latest/compute.schema.json
name: gpu-cluster
type: amlcompute
size: Standard_NC12

min_instances: 0
max_instances: 3
idle_time_before_scale_down: 120
```

The specification file creates a GPU cluster called `gpu-cluster` with at most 3 Standard_NC12 VM nodes that automatically scales down to 0 nodes after 120 seconds of inactivity.

For more information on VM sizes, see [sizes for Linux virtual machines in Azure](#).

4. Right-click the specification file and select **AzureML: Execute YAML**.

After a few minutes, the new compute target appears in the *Compute > Compute clusters* node of your workspace.

Train image classification model

During the training process, a TensorFlow model is trained by processing the training data and learning patterns embedded within it for each of the respective digits being classified.

Like workspaces and compute targets, training jobs are defined using resource templates. For this sample, the specification is defined in the *job.yml* file which looks like the following:

```
yml  
  
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json  
code: src  
command: >  
    python train.py  
environment: azureml:AzureML-tensorflow-2.4-ubuntu18.04-py37-cuda11-gpu:48  
compute: azureml:gpu-cluster  
experiment_name: tensorflow-mnist-example  
description: Train a basic neural network with TensorFlow on the MNIST dataset.
```

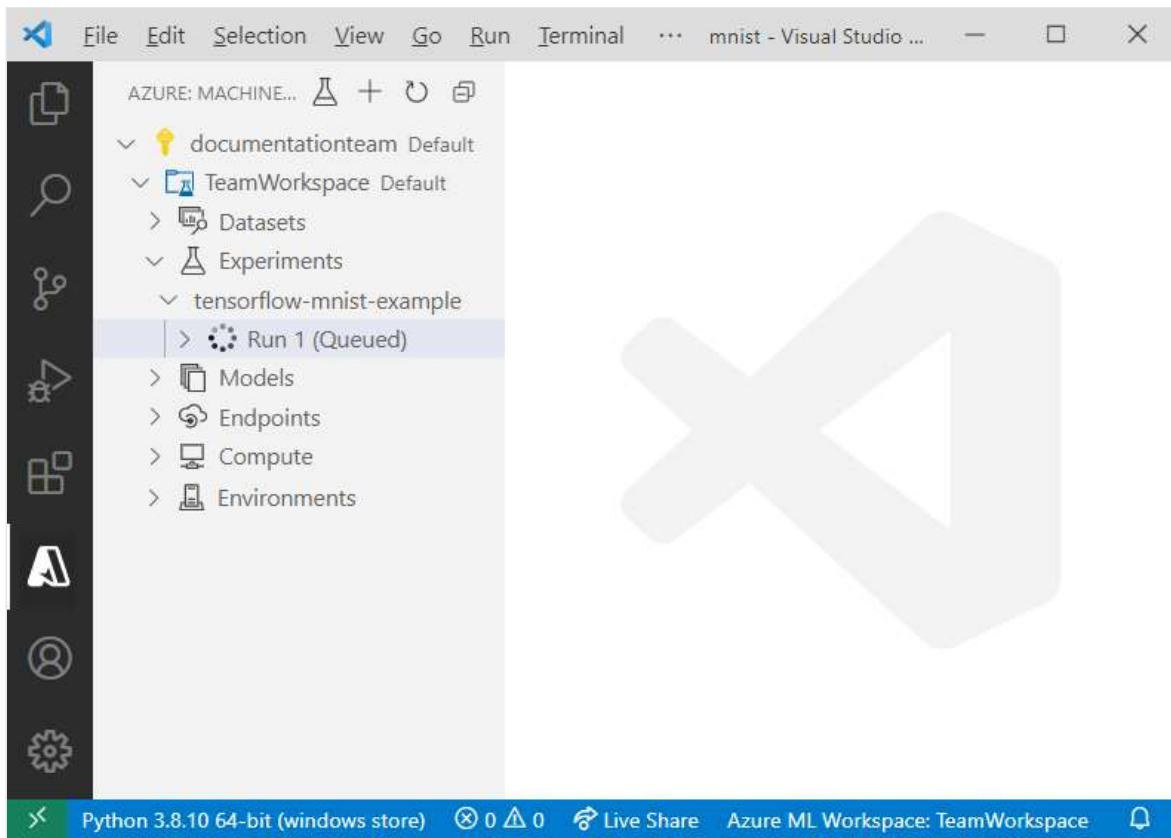
This specification file submits a training job called `tensorflow-mnist-example` to the recently created `gpu-cluster` computer target that runs the code in the *train.py* Python script. The environment used is one of the curated environments provided by Azure Machine Learning which contains TensorFlow and other software dependencies required to run the training script. For more information on curated environments, see [Azure Machine Learning curated environments](#).

To submit the training job:

1. Open the *job.yml* file.
2. Right-click the file in the text editor and select **AzureML: Execute YAML**.

At this point, a request is sent to Azure to run your experiment on the selected compute target in your workspace. This process takes several minutes. The amount of time to run the training job is impacted by several factors like the compute type and training data size. To track the progress of your experiment, right-click the current run node and select **View Job in Azure portal**.

When the dialog requesting to open an external website appears, select **Open**.



When the model is done training, the status label next to the run node updates to "Completed".

Next steps

In this tutorial, you learn the following tasks:

- ✓ Understand the code
- ✓ Create a workspace
- ✓ Create a GPU cluster for training
- ✓ Train a model

For next steps, see:

- [Create and manage Azure Machine Learning resources using Visual Studio Code](#).
- [Connect Visual Studio Code to a compute instance](#) for a full development experience.
- For a walkthrough of how to edit, run, and debug code locally, see the [Python hello-world tutorial](#).
- [Run Jupyter Notebooks in Visual Studio Code](#) using a remote Jupyter server.
- For a walkthrough of how to train with Azure Machine Learning outside of Visual Studio Code, see [Tutorial: Train and deploy a model with Azure Machine Learning](#).

Additional resources

Documentation

[Train deep learning Keras models \(SDK v2\) - Azure Machine Learning](#)

Learn how to train and register a Keras deep neural network classification model running on TensorFlow using Azure Machine Learning SDK (v2).

[Tutorial: AutoML- train object detection model \(v1\) - Azure Machine Learning](#)

Train an object detection model to identify if an image contains certain objects with automated ML and the Azure Machine Learning Python SDK automated ML. (v1)

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Tutorial: Train and deploy an example in Jupyter Notebook - Azure Machine Learning](#)

Use Azure Machine Learning to train and deploy an image classification model with scikit-learn in a cloud-based Python Jupyter Notebook.

[Prepare data for computer vision tasks - Azure Machine Learning](#)

Image data preparation for Azure Machine Learning automated ML to train computer vision models on classification, object detection, and segmentation

[Train deep learning PyTorch models \(SDK v2\) - Azure Machine Learning](#)

Learn how to run your PyTorch training scripts at enterprise scale using Azure Machine Learning SDK (v2).

[What is distributed training? - Azure Machine Learning](#)

Learn what type of distributed training Azure Machine Learning supports and the open source framework integrations available for distributed training.

[Show 5 more](#)

Training

Learning paths and modules

[Train models in Azure Machine Learning with the CLI \(v2\) - Training](#)

Train models in Azure Machine Learning with the CLI (v2)

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Explore Azure Machine Learning with Jupyter Notebooks

Article • 10/12/2022 • 2 minutes to read

APPLIES TO:  Python SDK azure-ai-ml v2 (current) ↗

The [AzureML-Examples](#) ↗ repository includes the latest (v2) Azure Machine Learning Python CLI and SDK samples. For information on the various example types, see the [readme](#) ↗.

This article shows you how to access the repository from the following environments:

- Azure Machine Learning compute instance
- Your own compute resource
- Data Science Virtual Machine

Option 1: Access on Azure Machine Learning compute instance (recommended)

The easiest way to get started with the samples is to complete the [Quickstart: Get started with Azure Machine Learning](#). Once completed, you'll have a dedicated notebook server pre-loaded with the SDK and the Azure Machine Learning Notebooks repository. No downloads or installation necessary.

To view example notebooks: 1. Sign in to [studio](#) ↗ and select your workspace if necessary. 1. Select **Notebooks**. 1. Select the **Samples** tab. Use the **SDK v2** folder for examples using Python SDK v2.

Option 2: Access on your own notebook server

If you'd like to bring your own notebook server for local development, follow these steps on your computer.

1. Use the instructions at [Azure Machine Learning SDK](#) ↗ to install the Azure Machine Learning SDK (v2) for Python
2. Create an [Azure Machine Learning workspace](#).
3. Write a [configuration file](#) file (`aml_config/config.json`).
4. Clone [the AzureML-Examples repository](#) ↗.

```
Bash
```

```
git clone https://github.com/Azure/azureml-examples.git --depth 1
```

5. Start the notebook server from the directory containing your clone.

```
Bash
```

```
jupyter notebook
```

These instructions install the base SDK packages necessary for the quickstart and tutorial notebooks. Other sample notebooks may require you to install extra components. For more information, see [Install the Azure Machine Learning SDK for Python ↗](#).

Option 3: Access on a DSVM

The Data Science Virtual Machine (DSVM) is a customized VM image built specifically for doing data science. If you [create a DSVM](#), the SDK and notebook server are installed and configured for you. However, you'll still need to create a workspace and clone the sample repository.

1. [Create an Azure Machine Learning workspace](#).

2. Download a workspace configuration file:

- Sign in to [Azure Machine Learning studio ↗](#)
- Select your workspace settings in the upper right
- Select [Download config file](#)

ML-docs my-workspace

Directory + Subscription + Workspace

Current directory

Microsoft

Current subscription

ML-docs

Current workspace

my-workspace

Resource Group

sgilley-rg

Subscription ID

xxxx-xxxx-xxxxxxxx-xxxxxx-xxxxxx-xxxx-xxxxxx-x@

Location

eastus2

Download config file

View all properties in Azure Portal

3. From the directory where you added the configuration file, clone the [the AzureML-Examples repository](#).

```
Bash
```

```
git clone https://github.com/Azure/azureml-examples.git --depth 1
```

4. Start the notebook server from the directory, which now contains the clone and the config file.

```
Bash
```

```
jupyter notebook
```

Next steps

Explore the [AzureML-Examples](#) repository to discover what Azure Machine Learning can do.

For more examples of MLOps, see <https://github.com/Azure/mlops-v2>.

Try these tutorials:

- [Train and deploy an image classification model with MNIST](#)
- [Tutorial: Train an object detection model with AutoML and Python](#)

Example pipelines & datasets for Azure Machine Learning designer

Article • 01/25/2023 • 10 minutes to read

Use the built-in examples in Azure Machine Learning designer to quickly get started building your own machine learning pipelines. The Azure Machine Learning designer [GitHub repository](#) contains detailed documentation to help you understand some common machine learning scenarios.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#)
- An Azure Machine Learning workspace

Important

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Use sample pipelines

The designer saves a copy of the sample pipelines to your studio workspace. You can edit the pipeline to adapt it to your needs and save it as your own. Use them as a starting point to jumpstart your projects.

Here's how to use a designer sample:

1. Sign in to [ml.azure.com](#), and select the workspace you want to work with.
2. Select **Designer**.
3. Select a sample pipeline under the **New pipeline** section.

Select **Show more samples** for a complete list of samples.

4. To run a pipeline, you first have to set default compute target to run the pipeline on.

- a. In the **Settings** pane to the right of the canvas, select **Select compute target**.
- b. In the dialog that appears, select an existing compute target or create a new one. Select **Save**.
- c. Select **Submit** at the top of the canvas to submit a pipeline job.

Depending on the sample pipeline and compute settings, jobs may take some time to complete. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline jobs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each component to further improve efficiency.

5. After the pipeline finishes running, you can review the pipeline and view the output for each component to learn more. Use the following steps to view component outputs:

- a. Right-click the component in the canvas whose output you'd like to see.
- b. Select **Visualize**.

Use the samples as starting points for some of the most common machine learning scenarios.

Regression

Explore these built-in regression samples.

Sample title	Description
Regression - Automobile Price Prediction (Basic) ↗	Predict car prices using linear regression.
Regression - Automobile Price Prediction (Advanced) ↗	Predict car prices using decision forest and boosted decision tree regressors. Compare models to find the best algorithm.

Classification

Explore these built-in classification samples. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

Sample title	Description
Binary Classification with Feature Selection - Income Prediction ↗	Predict income as high or low, using a two-class boosted decision tree. Use Pearson correlation to select features.
Binary Classification with custom Python script - Credit Risk Prediction ↗	Classify credit applications as high or low risk. Use the Execute Python Script component to weight your data.
Binary Classification - Customer Relationship Prediction ↗	Predict customer churn using two-class boosted decision trees. Use SMOTE to sample biased data.
Text Classification - Wikipedia SP 500 Dataset ↗	Classify company types from Wikipedia articles with multiclass logistic regression.
Multiclass Classification - Letter Recognition	Create an ensemble of binary classifiers to classify written letters.

Computer vision

Explore these built-in computer vision samples. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

Sample title	Description
Image Classification using DenseNet	Use computer vision components to build image classification model based on PyTorch DenseNet.

Recommender

Explore these built-in recommender samples. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

Sample title	Description
Wide & Deep based Recommendation - Restaurant Rating Prediction	Build a restaurant recommender engine from restaurant/user features and ratings.
Recommendation - Movie Rating Tweets	Build a movie recommender engine from movie/user features and ratings.

Utility

Learn more about the samples that demonstrate machine learning utilities and features.

You can learn more about the samples by opening the samples and viewing the component comments in the designer.

Sample title	Description
Binary Classification using Vowpal Wabbit Model - Adult Income Prediction	Vowpal Wabbit is a machine learning system which pushes the frontier of machine learning with techniques such as online, hashing, allreduce, reductions, learning2search, active, and interactive learning. This sample shows how to use Vowpal Wabbit model to build binary classification model.
Use custom R script - Flight Delay Prediction	Use customized R script to predict if a scheduled passenger flight will be delayed by more than 15 minutes.
Cross Validation for Binary Classification - Adult Income Prediction	Use cross validation to build a binary classifier for adult income.
Permutation Feature Importance	Use permutation feature importance to compute importance scores for the test dataset.
Tune Parameters for Binary Classification - Adult Income Prediction	Use Tune Model Hyperparameters to find optimal hyperparameters to build a binary classifier.

Datasets

When you create a new pipeline in Azure Machine Learning designer, a number of sample datasets are included by default. These sample datasets are used by the sample pipelines in the designer homepage.

The sample datasets are available under **Datasets-Samples** category. You can find this in the component palette to the left of the canvas in the designer. You can use any of these datasets in your own pipeline by dragging it to the canvas.

Dataset name	Dataset description

Dataset name	Dataset description
Adult Census Income Binary Classification dataset	<p>A subset of the 1994 Census database, using working adults over the age of 16 with an adjusted income index of > 100.</p> <p>Usage: Classify people using demographics to predict whether a person earns over 50K a year.</p> <p>Related Research: Kohavi, R., Becker, B., (1996). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science</p>
Automobile price data (Raw)	<p>Information about automobiles by make and model, including the price, features such as the number of cylinders and MPG, as well as an insurance risk score.</p> <p>The risk score is initially associated with auto price. It is then adjusted for actual risk in a process known to actuaries as symboling. A value of +3 indicates that the auto is risky, and a value of -3 that it is probably safe.</p> <p>Usage: Predict the risk score by features, using regression or multivariate classification.</p> <p>Related Research: Schlimmer, J.C. (1987). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>
CRM Appetency Labels Shared	<p>Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train_appetency.labels).</p>
CRM Churn Labels Shared	<p>Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train_churn.labels).</p>
CRM Dataset Shared	<p>This data comes from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train.data.zip).</p> <p>The dataset contains 50K customers from the French Telecom company Orange. Each customer has 230 anonymized features, 190 of which are numeric and 40 are categorical. The features are very sparse.</p>
CRM Upselling Labels Shared	<p>Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_large_train_upselling.labels).</p>
Flight Delays Data	<p>Passenger flight on-time performance data taken from the TranStats data collection of the U.S. Department of Transportation (On-Time).</p> <p>The dataset covers the time period April-October 2013. Before uploading to the designer, the dataset was processed as follows:</p> <ul style="list-style-type: none"> - The dataset was filtered to cover only the 70 busiest airports in the continental US - Canceled flights were labeled as delayed by more than 15 minutes - Diverted flights were filtered out - The following columns were selected: Year, Month, DayofMonth, DayOfWeek, Carrier, OriginAirportID, DestAirportID, CRSDepTime, DepDelay, DepDel15, CRSArrTime, ArrDelay, ArrDel15, Canceled

Dataset name	Dataset description
German Credit Card UCI dataset	<p>The UCI Statlog (German Credit Card) dataset (Statlog+German+Credit+Data ↗), using the german.data file.</p> <p>The dataset classifies people, described by a set of attributes, as low or high credit risks. Each example represents a person. There are 20 features, both numerical and categorical, and a binary label (the credit risk value). High credit risk entries have label = 2, low credit risk entries have label = 1. The cost of misclassifying a low risk example as high is 1, whereas the cost of misclassifying a high risk example as low is 5.</p>
IMDB Movie Titles	<p>The dataset contains information about movies that were rated in Twitter tweets: IMDB movie ID, movie name, genre, and production year. There are 17K movies in the dataset. The dataset was introduced in the paper "S. Dooms, T. De Pessemier and L. Martens. MovieTweetings: a Movie Rating Dataset Collected From Twitter. Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013."</p>
Movie Ratings	<p>The dataset is an extended version of the Movie Tweetings dataset. The dataset has 170K ratings for movies, extracted from well-structured tweets on Twitter. Each instance represents a tweet and is a tuple: user ID, IMDB movie ID, rating, timestamp, number of favorites for this tweet, and number of retweets of this tweet. The dataset was made available by A. Said, S. Dooms, B. Loni and D. Tikk for Recommender Systems Challenge 2014.</p>
Weather Dataset	<p>Hourly land-based weather observations from NOAA (merged data from 201304 to 201310 ↗).</p> <p>The weather data covers observations made from airport weather stations, covering the time period April-October 2013. Before uploading to the designer, the dataset was processed as follows:</p> <ul style="list-style-type: none"> - Weather station IDs were mapped to corresponding airport IDs - Weather stations not associated with the 70 busiest airports were filtered out - The Date column was split into separate Year, Month, and Day columns - The following columns were selected: AirportID, Year, Month, Day, Time, TimeZone, SkyCondition, Visibility, WeatherType, DryBulbFarenheit, DryBulbCelsius, WetBulbFarenheit, WetBulbCelsius, DewPointFarenheit, DewPointCelsius, RelativeHumidity, WindSpeed, WindDirection, ValueForWindCharacter, StationPressure, PressureTendency, PressureChange, SeaLevelPressure, RecordType, HourlyPrecip, Altimeter
Wikipedia SP 500 Dataset	<p>Data is derived from Wikipedia (https://www.wikipedia.org/ ↗) based on articles of each S&P 500 company, stored as XML data.</p> <p>Before uploading to the designer, the dataset was processed as follows:</p> <ul style="list-style-type: none"> - Extract text content for each specific company - Remove wiki formatting - Remove non-alphanumeric characters - Convert all text to lowercase - Known company categories were added <p>Note that for some companies an article could not be found, so the number of records is less than 500.</p>

Dataset name	Dataset description
Restaurant Feature Data	<p>A set of metadata about restaurants and their features, such as food type, dining style, and location.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>
Restaurant Ratings	<p>Contains ratings given by users to restaurants on a scale from 0 to 2.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>
Restaurant Customer Data	<p>A set of metadata about customers, including demographics and preferences.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>

Clean up resources

i Important

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.

The screenshot shows the Microsoft Azure Resource Groups blade. On the left, there's a list of resource groups under the heading 'Resource groups'. One resource group, 'my-rg', is highlighted with a red box. On the right, the details for 'my-rg' are shown, including its subscription information ('Subscription (change) : documentationteam'), subscription ID ('Subscription ID : aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaaaa'), and tags ('Tags (change) : Click here to add tags'). Below this, there's a list of resources within the group, including 'my-ws', 'myws22', 'myws70', and 'myws74', each with a checkbox next to it.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the **Delete** button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

The screenshot shows the 'Compute' blade in the Azure Machine Learning studio. The left sidebar has sections for Home, Author, Automated ML, Designer, Notebooks, Assets, and Manage. Under Manage, the 'Compute' button is highlighted with a red box. The main area is titled 'Compute' and shows tabs for Compute Instances, Training Clusters, Inference Clusters (which is selected and underlined), and Attached Compute. Below these tabs is a toolbar with New, Refresh, Delete (also highlighted with a red box), and an ellipsis. A search bar says 'Search to filter items...'. The main content area displays a table with columns Name, Type, Provisioning Status, Creation Date, and Status. One row is selected, showing 'aks-compute' as a 'Kubernetes Service'. Navigation arrows for 'Prev' and 'Next' are at the bottom.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the 'Datasets' blade in the Azure Machine Learning studio. The left sidebar has sections for New, Home, Author, Notebooks, Automated ML, Designer, Assets, and Datasets (which is selected and highlighted with a red box). The main area shows a dataset named 'TD-Sample_1: Regression - Automobile Price Prediction (Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1'. The 'Details' tab is selected. At the top right, there is a dropdown for 'Version 1 (latest)'. Below it are buttons for Refresh, Generate profile, Unregister (highlighted with a red box), and New version. The dataset details include Attributes (Properties, File, Description, Datastore, Relative path, Profile), Tags (CreatedByAMLStudio, true), and Sample usage (a code snippet in Python).

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

Learn the fundamentals of predictive analytics and machine learning with [Tutorial: Predict automobile price with the designer](#)

Additional resources

Documentation

[Assess ML models' fairness in Python \(preview\) - Azure Machine Learning](#)

Learn how to assess and mitigate the fairness of your machine learning models using Fairlearn and the Azure Machine Learning Python SDK.

[Create and explore datasets with labels - Azure Machine Learning](#)

Learn how to export data labels from your Azure Machine Learning labeling projects and use them for machine learning tasks.

[Create Python Model: Component reference - Azure Machine Learning](#)

Learn how to use the Create Python Model component in Azure Machine Learning to create a custom modeling or data processing component.

[Score Model: Component Reference - Azure Machine Learning](#)

Learn how to use the Score Model component in Azure Machine Learning to generate predictions using a trained classification or regression model.

[Model explainability in automated ML \(preview\) - Azure Machine Learning](#)

Learn how to get explanations for how your automated ML model determines feature importance and makes predictions when using the Azure Machine Learning SDK.

[Execute Python Script in the designer - Azure Machine Learning](#)

Learn how to use the Execute Python Script model in Azure Machine Learning designer to run custom operations written in Python.

[azureml.train.automl.automlconfig.AutoMLConfig class - Azure Machine Learning Python](#)

Represents configuration for submitting an automated ML experiment in Azure Machine Learning. This configuration object contains and persists the parameters for configuring the experiment run, as well as the training data to be used at run time. For guidance on selecting your settings, see...

[Set up AutoML with the studio UI - Azure Machine Learning](#)

Learn how to set up AutoML training jobs without a single line of code with Azure Machine Learning automated ML in the Azure Machine Learning studio.

[Show 5 more](#)

What is Azure Machine Learning CLI & Python SDK v2?

Article • 02/24/2023 • 4 minutes to read

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) ↗

Azure Machine Learning CLI v2 and Azure Machine Learning Python SDK v2 introduce a consistency of features and terminology across the interfaces. In order to create this consistency, the syntax of commands differs, in some cases significantly, from the first versions (v1).

Azure Machine Learning CLI v2

The Azure Machine Learning CLI v2 (CLI v2) is the latest extension for the [Azure CLI](#). The CLI v2 provides commands in the format `az ml <noun> <verb> <options>` to create and maintain Azure Machine Learning assets and workflows. The assets or workflows themselves are defined using a YAML file. The YAML file defines the configuration of the asset or workflow – what is it, where should it run, and so on.

A few examples of CLI v2 commands:

- `az ml job create --file my_job_definition.yaml`
- `az ml environment update --name my-env --file my_updated_env_definition.yaml`
- `az ml model list`
- `az ml compute show --name my_compute`

Use cases for CLI v2

The CLI v2 is useful in the following scenarios:

- On board to Azure Machine Learning without the need to learn a specific programming language

The YAML file defines the configuration of the asset or workflow – what is it, where should it run, and so on. Any custom logic/IP used, say data preparation, model training, model scoring can remain in script files, which are referred to in the YAML, but not part of the YAML itself. Azure Machine Learning supports script files in python, R, Java, Julia or C#. All you need to learn is YAML format and command lines to use Azure Machine Learning. You can stick with script files of your choice.

- Ease of deployment and automation

The use of command-line for execution makes deployment and automation simpler, since workflows can be invoked from any offering/platform, which allows users to call the command line.

- Managed inference deployments

Azure Machine Learning offers [endpoints](#) to streamline model deployments for both real-time and batch inference deployments. This functionality is available only via CLI v2 and SDK v2.

- Reusable components in pipelines

Azure Machine Learning introduces [components](#) for managing and reusing common logic across pipelines. This functionality is available only via CLI v2 and SDK v2.

Azure Machine Learning Python SDK v2

Azure Machine Learning Python SDK v2 is an updated Python SDK package, which allows users to:

- Submit training jobs
- Manage data, models, environments
- Perform managed inferencing (real time and batch)
- Stitch together multiple tasks and production workflows using Azure Machine Learning pipelines

The SDK v2 is on par with CLI v2 functionality and is consistent in how assets (nouns) and actions (verbs) are used between SDK and CLI. For example, to list an asset, the `list` action can be used in both CLI and SDK. The same `list` action can be used to list a compute, model, environment, and so on.

Use cases for SDK v2

The SDK v2 is useful in the following scenarios:

- Use Python functions to build a single step or a complex workflow

SDK v2 allows you to build a single command or a chain of commands like Python functions - the command has a name, parameters, expects input, and returns output.

- Move from simple to complex concepts incrementally

SDK v2 allows you to:

- Construct a single command.
- Add a hyperparameter sweep on top of that command,
- Add the command with various others into a pipeline one after the other.

This construction is useful, given the iterative nature of machine learning.

- Reusable components in pipelines

Azure Machine Learning introduces [components](#) for managing and reusing common logic across pipelines. This functionality is available only via CLI v2 and SDK v2.

- Managed inferencing

Azure Machine Learning offers [endpoints](#) to streamline model deployments for both real-time and batch inference deployments. This functionality is available only via CLI v2 and SDK v2.

Should I use v1 or v2?

CLI v2

The Azure Machine Learning CLI v1 has been deprecated. We recommend you to use CLI v2 if:

- You were a CLI v1 user
- You want to use new features like - reusable components, managed inferencing
- You don't want to use a Python SDK - CLI v2 allows you to use YAML with scripts in python, R, Java, Julia or C#
- You were a user of R SDK previously - Azure Machine Learning won't support an SDK in `R`. However, the CLI v2 has support for `R` scripts.
- You want to use command line based automation/deployments
- You don't need Spark Jobs. This feature is currently available in preview in CLI v2.

SDK v2

The Azure Machine Learning Python SDK v1 doesn't have a planned deprecation date. If you have significant investments in Python SDK v1 and don't need any new features

offered by SDK v2, you can continue to use SDK v1. However, you should consider using SDK v2 if:

- You want to use new features like - reusable components, managed inferencing
- You're starting a new workflow or pipeline - all new features and future investments will be introduced in v2
- You want to take advantage of the improved usability of the Python SDK v2 - ability to compose jobs and pipelines using Python functions, easy evolution from simple to complex tasks etc.
- You don't need Spark Jobs. This feature is currently available in preview in SDK v2.

Next steps

- [How to upgrade from v1 to v2](#)
- Get started with CLI v2
 - [Install and set up CLI \(v2\)](#)
 - [Train models with the CLI \(v2\)](#)
 - [Deploy and score models with online endpoints](#)
- Get started with SDK v2
 - [Install and set up SDK \(v2\) ↗](#)
 - [Train models with the Azure Machine Learning Python SDK v2](#)
 - [Tutorial: Create production ML pipelines with Python SDK v2 in a Jupyter notebook](#)

Additional resources

Documentation

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[CLI \(v2\) mltable YAML schema - Azure Machine Learning](#)

Reference documentation for the CLI (v2) MLTable YAML schema.

[Tutorial: Upload data and train a model \(SDK v1\) - Azure Machine Learning](#)

How to upload and use your own data in a remote training job, with SDK v1. This is part 3 of a three-part getting-started series.

[azureml.core.model.InferenceConfig class - Azure Machine Learning Python](#)

Represents configuration settings for a custom environment used for deployment. Inference configuration is an input parameter for Model deployment-related actions: deploy profile package

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[How to do hyperparameter sweep in pipeline - Azure Machine Learning](#)

How to use sweep to do hyperparameter tuning in Azure Machine Learning pipeline using CLI v2 and Python SDK

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Show 5 more](#)

[Training](#)

Learning paths and modules

[Train models in Azure Machine Learning with the CLI \(v2\) - Training](#)

Train models in Azure Machine Learning with the CLI (v2)

Data concepts in Azure Machine Learning

Article • 02/24/2023 • 5 minutes to read

With Azure Machine Learning, you can bring data from a local machine or an existing cloud-based storage. In this article, you'll learn the main Azure Machine Learning data concepts.

URI

A Uniform Resource Identifier (URI) represents a storage location on your local computer, Azure storage, or a publicly available http(s) location. These examples show URIs for different storage options:

Storage location	URI examples
Local computer	<code>./home/username/data/my_data</code>
Public http(s) server	<code>https://raw.githubusercontent.com/pandas-dev/pandas/main/doc/data/titanic.csv</code>
Blob storage	<code>wasbs://<containername>@<accountname>.blob.core.windows.net/<folder>/</code>
Azure Data Lake (gen2)	<code>abfss://<file_system>@<account_name>.dfs.core.windows.net/<folder>/<file>.csv</code>
Azure Data Lake (gen1)	<code>adl://<accountname>.azuredatalakestore.net/<folder1>/<folder2></code>
Azure Machine Learning Datastore	<code>azureml://datastores/<data_store_name>/paths/<folder1>/<folder2>/<folder3>/<file>.parquet</code>

An Azure Machine Learning job maps URIs to the compute target filesystem. This mapping means that in a command that consumes or produces a URI, that URI works like a file or a folder. A URI uses **identity-based authentication** to connect to storage services,

with either your Azure Active Directory ID (default), or Managed Identity. Azure Machine Learning [Datastore](#) URIs can apply either identity-based authentication, or **credential-based** (for example, Service Principal, SAS token, account key) without exposure of secrets.

A URI can serve as either *input* or an *output* to an Azure Machine Learning job, and it can map to the compute target filesystem with one of four different *mode* options:

- **Read-only mount (`ro_mount`)**: The URI represents a storage location that is *mounted* to the compute target filesystem. The mounted data location supports read-only output exclusively.
- **Read-write mount (`rw_mount`)**: The URI represents a storage location that is *mounted* to the compute target filesystem. The mounted data location supports both read output from it *and* data writes to it.
- **Download (`download`)**: The URI represents a storage location containing data that is *downloaded* to the compute target filesystem.
- **Upload (`upload`)**: All data written to a compute target location is *uploaded* to the storage location represented by the URI.

Additionally, you can pass in the URI as a job input string with the **direct** mode. This table summarizes the combination of modes available for inputs and outputs:

Job Input or Output	<code>upload</code>	<code>download</code>	<code>ro_mount</code>	<code>rw_mount</code>	<code>direct</code>
Input		✓	✓		✓
Output	✓			✓	

Read [Access data in a job](#) for more information.

Data types

A URI (storage location) can reference a file, a folder, or a data table. A machine learning job input and output definition requires one of the following three data types:

Type	V2 API	V1 API	Canonical Scenarios	V2/V1 API Difference

Type	V2 API	V1 API	Canonical Scenarios	V2/V1 API Difference
File Reference a single file	<code>uri_file</code>	<code>FileDataset</code>	Read/write a single file - the file can have any format.	A type new to V2 APIs. In V1 APIs, files always mapped to a folder on the compute target filesystem; this mapping required an <code>os.path.join</code> . In V2 APIs, the single file is mapped. This way, you can refer to that location in your code.
Folder Reference a single folder	<code>uri_folder</code>	<code>FileDataset</code>	You must read/write a folder of parquet/CSV files into Pandas/Spark.	In V1 APIs, <code>FileDataset</code> had an associated engine that could take a file sample from a folder. In V2 APIs, a Folder is a simple mapping to the compute target filesystem.
Table Reference a data table	<code>mltable</code>	<code>TabularDataset</code>	Deep-learning with images, text, audio, video files located in a folder.	In V1 APIs, the Azure Machine Learning back-end stored the data materialization blueprint. As a result, <code>TabularDataset</code> only worked if you had an Azure Machine Learning workspace. <code>mltable</code> stores the data materialization blueprint in <i>your</i> storage. This storage location means you can use it <i>disconnected to AzureML</i> - for example, local, on-premises. In V2 APIs, you'll find it easier to transition from local to remote jobs. Read Working with tables in Azure Machine Learning for more information.

Data runtime capability

Azure Machine Learning uses its own *data runtime* for mounts/uploads/downloads, to map storage URIs to the compute target filesystem, or to materialize tabular data into pandas/spark with Azure Machine Learning tables (`mltable`). The Azure Machine Learning

data runtime is designed for machine learning task *high speed and high efficiency*. Its key benefits include:

- ✓ [Rust](#) language architecture. The Rust language is known for high speed and high memory efficiency.
- ✓ Light weight; the Azure Machine Learning data runtime has *no* dependencies on other technologies - JVM, for example - so the runtime installs quickly on compute targets.
- ✓ Multi-process (parallel) data loading.
- ✓ Data pre-fetches operate as background task on the CPU(s), to enhance utilization of the GPU(s) in deep-learning operations.
- ✓ Seamless authentication to cloud storage.

Datastore

An Azure Machine Learning datastore serves as a *reference* to an *existing* Azure storage account. The benefits of Azure Machine Learning datastore creation and use include:

1. A common, easy-to-use API that interacts with different storage types (Blob/Files/ADLS).
2. Easier discovery of useful datastores in team operations.
3. For credential-based access (service principal/SAS/key), Azure Machine Learning datastore secures connection information. This way, you won't need to place that information in your scripts.

When you create a datastore with an existing Azure storage account, you can choose between two different authentication methods:

- **Credential-based** - authenticate data access with a service principal, shared access signature (SAS) token, or account key. Users with *Reader* workspace access can access the credentials.
- **Identity-based** - use your Azure Active Directory identity or managed identity to authenticate data access.

The following table summarizes the Azure cloud-based storage services that an Azure Machine Learning datastore can create. Additionally, the table summarizes the authentication types that can access those services:

Supported storage service	Credential-based authentication	Identity-based authentication
Azure Blob Container	✓	✓
Azure File Share	✓	

Supported storage service	Credential-based authentication	Identity-based authentication
Azure Data Lake Gen1	✓	✓
Azure Data Lake Gen2	✓	✓

Read [Create datastores](#) for more information about datastores.

Data asset

An Azure Machine Learning data asset resembles web browser bookmarks (favorites). Instead of remembering long storage paths (URLs) that point to your most frequently used data, you can create a data asset, and then access that asset with a friendly name.

Data asset creation also creates a *reference* to the data source location, along with a copy of its metadata. Because the data remains in its existing location, you incur no extra storage cost, and you don't risk data source integrity. You can create Data assets from Azure Machine Learning datastores, Azure Storage, public URLs, or local files.

Read [Create data assets](#) for more information about data assets.

Next steps

- [Install and set up the CLI \(v2\)](#)
- [Create datastores](#)
- [Create data assets](#)
- [Access data in a job](#)
- [Data administration](#)

Additional resources

Documentation

[Hyperparameter tuning a model \(v1\) - Azure Machine Learning](#)

Automate hyperparameter tuning for deep learning and machine learning models using Azure Machine Learning.(v1)

[Create and explore datasets with labels - Azure Machine Learning](#)

Learn how to export data labels from your Azure Machine Learning labeling projects and use them for machine learning tasks.

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[Run batch predictions using Azure Machine Learning designer - Azure Machine Learning](#)

Learn how to create a batch prediction pipeline. Deploy the pipeline as a parameterized web service, and trigger it from any HTTP library.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Troubleshooting local model deployment - Azure Machine Learning](#)

Try a local model deployment as a first step in troubleshooting model deployment errors.

[Train scikit-learn machine learning models \(v2\) - Azure Machine Learning](#)

Learn how Azure Machine Learning enables you to scale out a scikit-learn training job using elastic cloud compute resources (v2).

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Show 5 more](#)

Training

Learning certificate

[Microsoft Certified: Azure Data Engineer Associate - Certifications](#)

Azure data engineers help stakeholders understand the data through exploration, and they build and maintain secure and compliant data processing pipelines by using different tools and techniques.

What is "human data" and why is it important to source responsibly?

Article • 01/04/2023 • 5 minutes to read

APPLIES TO:  [Python SDK azure-ai-ml v2 \(current\)](#) 

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Human data is data collected directly from, or about, people. Human data may include personal data such as names, age, images, or voice clips and sensitive data such as genetic data, biometric data, gender identity, religious beliefs, or political affiliations.

Collecting this data can be important to building AI systems that work for all users. But certain practices should be avoided, especially ones that can cause physical and psychological harm to data contributors.

The best practices in this article will help you conduct manual data collection projects from volunteers where everyone involved is treated with respect, and potential harms—especially those faced by vulnerable groups—are anticipated and mitigated. This means that:

- People contributing data are not coerced or exploited in any way, and they have control over what personal data is collected.
- People collecting and labeling data have adequate training.

These practices can also help ensure more-balanced and higher-quality datasets and better stewardship of human data.

These are emerging practices, and we are continually learning. The best practices below are a starting point as you begin your own responsible human data collections. These best practices are provided for informational purposes only and should not be treated as legal advice. All human data collections should undergo specific privacy and legal reviews.

General best practices

We suggest the following best practices for manually collecting human data directly from people.

Best Practice

Why?

Obtain voluntary informed consent.

- Participants should understand and consent to data collection and how their data will be used.
 - Data should only be stored, processed, and used for purposes that are part of the original documented informed consent.
 - Consent documentation should be properly stored and associated with the collected data.
-

Compensate data contributors appropriately.

- Data contributors should not be pressured or coerced into data collections and should be fairly compensated for their time and data.
 - Inappropriate compensation can be exploitative or coercive.
-

Let contributors self-identify demographic information.

- Demographic information that is not self-reported by data contributors but assigned by data collectors may 1) result in inaccurate metadata and 2) be disrespectful to data contributors.
-

Anticipate harms when recruiting vulnerable groups.

- Collecting data from vulnerable population groups introduces risk to data contributors and your organization.
-

Treat data contributors with respect.

- Improper interactions with data contributors at any phase of the data collection can negatively impact data quality, as well as the overall data collection experience for data contributors and data collectors.
-

Qualify external suppliers carefully.

- Data collections with unqualified suppliers may result in low quality data, poor data management, unprofessional practices, and potentially harmful outcomes for data contributors and data collectors (including violations of human rights).
 - Annotation or labeling work (e.g., audio transcription, image tagging) with unqualified suppliers may result in low quality or biased datasets, insecure data management, unprofessional practices, and potentially harmful outcomes for data contributors (including violations of human rights).
-

Communicate expectations clearly in the Statement of Work (SOW) (contracts or agreements) with suppliers.

- A contract which lacks requirements for responsible data collection work may result in low-quality or poorly collected data.
-

Qualify geographies carefully.

- When applicable, collecting data in areas of high geopolitical risk and/or unfamiliar geographies may result in unusable or low-quality data and may impact the safety of involved parties.
-

Be a good steward of your datasets.

- Improper data management and poor documentation can result in data misuse.
-

Note

This article focuses on recommendations for human data, including personal data and sensitive data such as biometric data, health data, racial or ethnic data, data collected manually from the general public or company employees, as well as metadata relating to human characteristics, such as age, ancestry, and gender identity, that may be created via annotation or labeling.

[Download the full recommendations here ↗](#)

Best practices for collecting age, ancestry, and gender identity

In order for AI systems to work well for everyone, the datasets used for training and evaluation should reflect the diversity of people who will use or be affected by those systems. In many cases, age, ancestry, and gender identity can help approximate the range of factors that might affect how well a product performs for a variety of people; however, collecting this information requires special consideration.

If you do collect this data, always let data contributors self-identify (choose their own responses) instead of having data collectors make assumptions, which might be incorrect. Also include a “prefer not to answer” option for each question. These practices will show respect for the data contributors and yield more balanced and higher-quality data.

These best practices have been developed based on three years of research with intended stakeholders and collaboration with many teams at Microsoft: [fairness and inclusiveness working groups ↗](#), [Global Diversity & Inclusion ↗](#), [Global Readiness ↗](#), [Office of Responsible AI ↗](#), and others.

To enable people to self-identify, consider using the following survey questions.

Age

How old are you?

Select your age range

[Include appropriate age ranges as defined by project purpose, geographical region, and guidance from domain experts]

- # to #
- # to #
- # to #
- Prefer not to answer

Ancestry

Please select the categories that best describe your ancestry

May select multiple

[Include appropriate categories as defined by project purpose, geographical region, and guidance from domain experts]

- Ancestry group
- Ancestry group
- Ancestry group
- Multiple (multiracial, mixed Ancestry)
- Not listed, I describe myself as: _____
- Prefer not to answer

Gender identity

How do you identify?

May select multiple

[Include appropriate gender identities as defined by project purpose, geographical region, and guidance from domain experts]

- Gender identity
- Gender identity
- Gender identity

- Prefer to self-describe: _____
- Prefer not to answer

Caution

In some parts of the world, there are laws that criminalize specific gender categories, so it may be dangerous for data contributors to answer this question honestly. Always give people a way to opt out. And work with regional experts and attorneys to conduct a careful review of the laws and cultural norms of each place where you plan to collect data, and if needed, avoid asking this question entirely.

[Download the full guidance here.](#) ↗

Next steps

For more information on how to work with your data:

- [Secure data access in Azure Machine Learning](#)
- [Data ingestion options for Azure Machine Learning workflows](#)
- [Optimize data processing with Azure Machine Learning](#)
- [Use differential privacy with Azure Machine Learning SDK](#)

Follow these how-to guides to work with your data after you've collected it:

- [Set up image labeling](#)
- [Label images and text](#)

Additional resources

Documentation

[Example Jupyter Notebooks \(v1\) - Azure Machine Learning](#)

Learn how to find and use the Jupyter Notebooks designed to help you explore the SDK (v1) and serve as models for your own machine learning projects.

[Search for assets \(preview\) - Azure Machine Learning](#)

Find your Azure Machine Learning assets with search

[mltable.mltable module - Azure Machine Learning Python](#)

Contains functionality to create and interact with MLTable objects

[mltable.MLTable class - Azure Machine Learning Python](#)

Represents a MLTable. A MLTable defines a series of lazily-evaluated, immutable operations to load data from the data source. Data is not loaded from the source until MLTable is asked to deliver data.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[azureml.core.webservice.aci.AciWebservice class - Azure Machine Learning Python](#)

Represents a machine learning model deployed as a web service endpoint on Azure Container Instances. A deployed service is created from a model, script, and associated files. The resulting web service is a load-balanced, HTTP endpoint with a REST API. You can send data to this API and receiv...

[How to do hyperparameter sweep in pipeline - Azure Machine Learning](#)

How to use sweep to do hyperparameter tuning in Azure Machine Learning pipeline using CLI v2 and Python SDK

[Show 4 more](#)

Export or delete your Machine Learning service workspace data

Article • 02/24/2023 • 2 minutes to read

In Azure Machine Learning, you can export or delete your workspace data using either the portal graphical interface or the Python SDK. This article describes both options.

ⓘ Note

For information about viewing or deleting personal data, see [Azure Data Subject Requests for the GDPR](#). For more information about GDPR, see the [GDPR section of the Microsoft Trust Center](#) and the [GDPR section of the Service Trust portal](#).

ⓘ Note

This article provides steps about how to delete personal data from the device or service and can be used to support your obligations under the GDPR. For general information about GDPR, see the [GDPR section of the Microsoft Trust Center](#) and the [GDPR section of the Service Trust portal](#).

Control your workspace data

In-product data stored by Azure Machine Learning is available for export and deletion. You can export and delete data with Azure Machine Learning studio, the CLI, and the SDK. Additionally, you can access telemetry data through the Azure Privacy portal.

In Azure Machine Learning, personal data consists of user information in job history documents.

Delete high-level resources using the portal

When you create a workspace, Azure creates several resources within the resource group:

- The workspace itself
- A storage account
- A container registry

- An Applications Insights instance
- A key vault

To delete these resources, selecting them from the list and choose **Delete**:

ⓘ Important

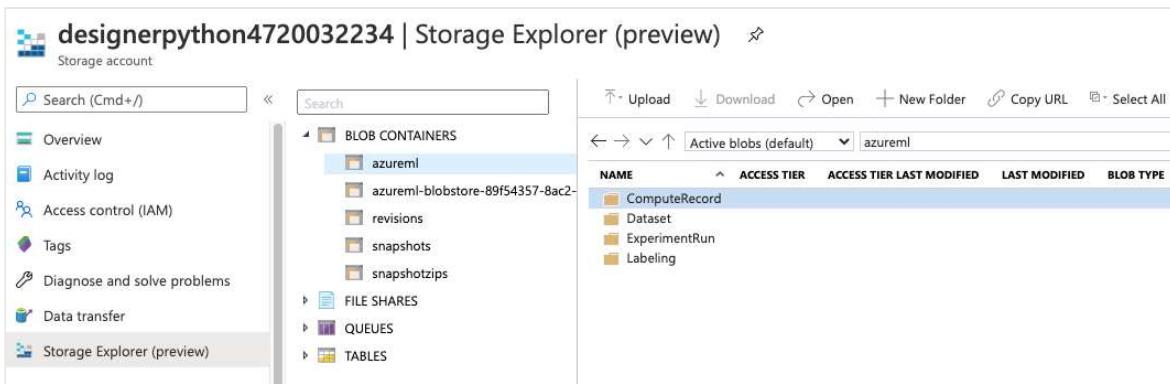
If the resource is configured for soft delete, the data won't actually delete unless you optionally select to delete the resource permanently. For more information, see the following articles:

- [Workspace soft-deletion.](#)
- [Soft delete for blobs.](#)
- [Soft delete in Azure Container Registry.](#)
- [Azure log analytics workspace.](#)
- [Azure Key Vault soft-delete.](#)

The screenshot shows the Azure portal interface for a resource group named 'my-resource-group'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Events, Settings (with sub-options like Quickstart, Deployments, Policies, Properties, Locks, Export template), Cost Management (Cost analysis, Cost alerts (preview), Budgets, Advisor recommendations), and Monitoring (Insights (preview), Alerts, Metrics, Diagnostic settings). The main content area displays the resource group details: Subscription (change) : mysubscription, Subscription ID : abcdeabcd-0f67-45d2-b838-b8f373d61234, Tags (change) : Click here to add tags. Below this is a table listing resources: Name, Type, and Location. The table shows five items: designerpython (Machine Learning, East US), designernpytha32138ef (Container registry, East US), designernpython3584489984 (Application Insights, East US), designernpython4720032234 (Storage account, East US), and designernpython6289077319 (Key vault, East US). The 'Delete' button in the top right of the main content area is highlighted with a red box.

Name	Type	Location
designerpython	Machine Learning	East US
designernpytha32138ef	Container registry	East US
designernpython3584489984	Application Insights	East US
designernpython4720032234	Storage account	East US
designernpython6289077319	Key vault	East US

Job history documents, which may contain personal user information, are stored in the storage account in blob storage, in `/azureml` subfolders. You can download and delete the data from the portal.



Export and delete machine learning resources using Azure Machine Learning studio

Azure Machine Learning studio provides a unified view of your machine learning resources - for example, notebooks, data assets, models, and jobs. Azure Machine Learning studio emphasizes preservation of a record of your data and experiments. You can delete computational resources such as pipelines and compute resources with the browser. For these resources, navigate to the resource in question and choose **Delete**.

You can unregister data assets and archive jobs, but these operations don't delete the data. To entirely remove the data, data assets and job data require deletion at the storage level. Storage level deletion happens in the portal, as described earlier. Azure Machine Learning Studio can handle individual deletion. Job deletion deletes the data of that job.

Azure Machine Learning Studio can handle training artifact downloads from experimental jobs. Choose the relevant **Job**. Choose **Output + logs**, and navigate to the specific artifacts you wish to download. Choose ... and **Download**, or select **Download all**.

To download a registered model, navigate to the **Model** and choose **Download**.

my-workspace > Models > test:1

test:1

Refresh Deploy Download

Next steps

Learn more about [Managing a workspace](#).

Additional resources

Documentation

[Tutorial: Upload data and train a model \(SDK v1\) - Azure Machine Learning](#)

How to upload and use your own data in a remote training job, with SDK v1. This is part 3 of a three-part getting-started series.

[Track, monitor, and analyze runs - Azure Machine Learning](#)

Learn how to start, monitor, and track your machine learning experiment runs with the Azure Machine Learning Python SDK.

[Profile model memory and CPU usage \(v1\) - Azure Machine Learning](#)

Use CLI (v1) or SDK (v1) to profile your model before deployment. Profiling determines the memory and CPU usage of your model.

[azureml.data.azure_storage_datastore.AzureFileDatastore class - Azure Machine Learning Python](#)

Represents a datastore that saves connection information to Azure File storage. You should not work with this class directly. To create a datastore of this type, use the register_azure_file_share method of Datastore. Note: When using a datastore to access data, you must have permission to access that...

[Tutorial: Get started with a Python script \(v1\) - Azure Machine Learning](#)

Get started with your first Python script in Azure Machine Learning, with SDK v1. This is part 1 of a three-part getting-started series.

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[Show 4 more](#)

Train models with Azure Machine Learning

Article • 01/27/2023 • 5 minutes to read

APPLIES TO:  Python SDK azure-ai-ml v2 (current) ↗

Azure Machine Learning provides several ways to train your models, from code-first solutions using the SDK to low-code solutions such as automated machine learning and the visual designer. Use the following list to determine which training method is right for you:

- [Azure Machine Learning SDK for Python](#): The Python SDK provides several ways to train models, each with different capabilities.

Training method	Description
command()	A typical way to train models is to submit a command() that includes a training script, environment, and compute information.
Automated machine learning	Automated machine learning allows you to train models without extensive data science or programming knowledge . For people with a data science and programming background, it provides a way to save time and resources by automating algorithm selection and hyperparameter tuning. You don't have to worry about defining a job configuration when using automated machine learning.
Machine learning pipeline	Pipelines are not a different training method, but a way of defining a workflow using modular, reusable steps that can include training as part of the workflow. Machine learning pipelines support using automated machine learning and run configuration to train models. Since pipelines are not focused specifically on training, the reasons for using a pipeline are more varied than the other training methods. Generally, you might use a pipeline when: <ul style="list-style-type: none">* You want to schedule unattended processes such as long running training jobs or data preparation.* Use multiple steps that are coordinated across heterogeneous compute resources and storage locations.* Use the pipeline as a reusable template for specific scenarios, such as retraining or batch scoring.* Track and version data sources, inputs, and outputs for your workflow.* Your workflow is implemented by different teams that work on specific steps independently. Steps can then be joined together in a pipeline to implement the workflow.

- **Designer:** Azure Machine Learning designer provides an easy entry-point into machine learning for building proof of concepts, or for users with little coding experience. It allows you to train models using a drag and drop web-based UI. You can use Python code as part of the design, or train models without writing any code.
- **Azure CLI:** The machine learning CLI provides commands for common tasks with Azure Machine Learning, and is often used for **scripting and automating tasks**. For example, once you've created a training script or pipeline, you might use the Azure CLI to start a training job on a schedule or when the data files used for training are updated. For training models, it provides commands that submit training jobs. It can submit jobs using run configurations or pipelines.

Each of these training methods can use different types of compute resources for training. Collectively, these resources are referred to as **compute targets**. A compute target can be a local machine or a cloud resource, such as an Azure Machine Learning Compute, Azure HDInsight, or a remote virtual machine.

Python SDK

The Azure Machine Learning SDK for Python allows you to build and run machine learning workflows with Azure Machine Learning. You can interact with the service from an interactive Python session, Jupyter Notebooks, Visual Studio Code, or other IDE.

- [Install/update the SDK](#)
- [Configure a development environment for Azure Machine Learning](#)

Submit a command

A generic training job with Azure Machine Learning can be defined using the [command\(\)](#). The command is then used, along with your training script(s) to train a model on the specified compute target.

You may start with a command for your local computer, and then switch to one for a cloud-based compute target as needed. When changing the compute target, you only change the compute parameter in the command that you use. A run also logs information about the training job, such as the inputs, outputs, and logs.

- [Tutorial: Train your first ML model](#)
- [Examples: Jupyter Notebook and Python examples of training models ↗](#)

Automated Machine Learning

Define the iterations, hyperparameter settings, featurization, and other settings. During training, Azure Machine Learning tries different algorithms and parameters in parallel. Training stops once it hits the exit criteria you defined.

💡 Tip

In addition to the Python SDK, you can also use Automated ML through [Azure Machine Learning studio](#).

- [What is automated machine learning?](#)
- [Tutorial: Create your first classification model with automated machine learning](#)
- [How to: Configure automated ML experiments in Python](#)
- [How to: Create, explore, and deploy automated machine learning experiments with Azure Machine Learning studio](#)

Machine learning pipeline

Machine learning pipelines can use the previously mentioned training methods. Pipelines are more about creating a workflow, so they encompass more than just the training of models.

- [What are ML pipelines in Azure Machine Learning?](#)
- [Tutorial: Create production ML pipelines with Python SDK v2 in a Jupyter notebook](#)

Understand what happens when you submit a training job

The Azure training lifecycle consists of:

1. Zipping the files in your project folder and upload to the cloud.

💡 Tip

To prevent unnecessary files from being included in the snapshot, make an ignore file (`.gitignore` or `.amlignore`) in the directory. Add the files and directories to exclude to this file. For more information on the syntax to use inside this file, see [syntax and patterns](#) for `.gitignore`. The `.amlignore` file uses the same syntax. *If both files exist, the `.amlignore` file is used and the `.gitignore` file is unused.*

2. Scaling up your compute cluster
3. Building or downloading the dockerfile to the compute node
 - a. The system calculates a hash of:
 - The base image
 - Custom docker steps (see [Deploy a model using a custom Docker base image](#))
 - The conda definition YAML (see [Manage Azure Machine Learning environments with the CLI \(v2\)](#)))
 - b. The system uses this hash as the key in a lookup of the workspace Azure Container Registry (ACR)
 - c. If it is not found, it looks for a match in the global ACR
 - d. If it is not found, the system builds a new image (which will be cached and registered with the workspace ACR)
4. Downloading your zipped project file to temporary storage on the compute node
5. Unzipping the project file
6. The compute node executing `python <entry script> <arguments>`
7. Saving logs, model files, and other files written to `./outputs` to the storage account associated with the workspace
8. Scaling down compute, including removing temporary storage

Azure Machine Learning designer

The designer lets you train models using a drag and drop interface in your web browser.

- [What is the designer?](#)
- [Tutorial: Predict automobile price](#)

Azure CLI

The machine learning CLI is an extension for the Azure CLI. It provides cross-platform CLI commands for working with Azure Machine Learning. Typically, you use the CLI to automate tasks, such as training a machine learning model.

- [Use the CLI extension for Azure Machine Learning](#)
- [MLOps on Azure ↗](#)

- Train models

VS Code

You can use the VS Code extension to run and manage your training jobs. See the [VS Code resource management how-to guide](#) to learn more.

Next steps

Learn how to [Tutorial: Create production ML pipelines with Python SDK v2 in a Jupyter notebook](#).

Additional resources

Documentation

[azureml.core.webservice.aci.AciWebservice class - Azure Machine Learning Python](#)

Represents a machine learning model deployed as a web service endpoint on Azure Container Instances. A deployed service is created from a model, script, and associated files. The resulting web service is a load-balanced, HTTP endpoint with a REST API. You can send data to this API and receiv...

[Deploy MLflow models as web services - Azure Machine Learning](#)

Set up MLflow with Azure Machine Learning to deploy your ML models as an Azure web service.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Hyperparameter for AutoML computer vision tasks - Azure Machine Learning](#)

Learn which hyperparameters are available for computer vision tasks with automated ML.

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Train scikit-learn machine learning models \(v2\) - Azure Machine Learning](#)

Learn how Azure Machine Learning enables you to scale out a scikit-learn training job using elastic cloud compute resources (v2).

[What is distributed training? - Azure Machine Learning](#)

Learn what type of distributed training Azure Machine Learning supports and the open source framework integrations available for distributed training.

[From artifacts to models in MLflow - Azure Machine Learning](#)

Learn about how MLflow uses the concept of models instead of artifacts to represent your trained

models and enable a streamlined path to deployment.

[Show 5 more](#)

Training

Learning paths and modules

[Train models in Azure Machine Learning with the CLI \(v2\) - Training](#)

Train models in Azure Machine Learning with the CLI (v2)

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Distributed training with Azure Machine Learning

Article • 10/20/2022 • 2 minutes to read

In this article, you learn about distributed training and how Azure Machine Learning supports it for deep learning models.

In distributed training the workload to train a model is split up and shared among multiple mini processors, called worker nodes. These worker nodes work in parallel to speed up model training. Distributed training can be used for traditional ML models, but is better suited for compute and time intensive tasks, like [deep learning](#) for training deep neural networks.

Deep learning and distributed training

There are two main types of distributed training: [data parallelism](#) and [model parallelism](#). For distributed training on deep learning models, the [Azure Machine Learning SDK in Python](#) supports integrations with popular frameworks, PyTorch and TensorFlow. Both frameworks employ data parallelism for distributed training, and can leverage [horovod](#) for optimizing compute speeds.

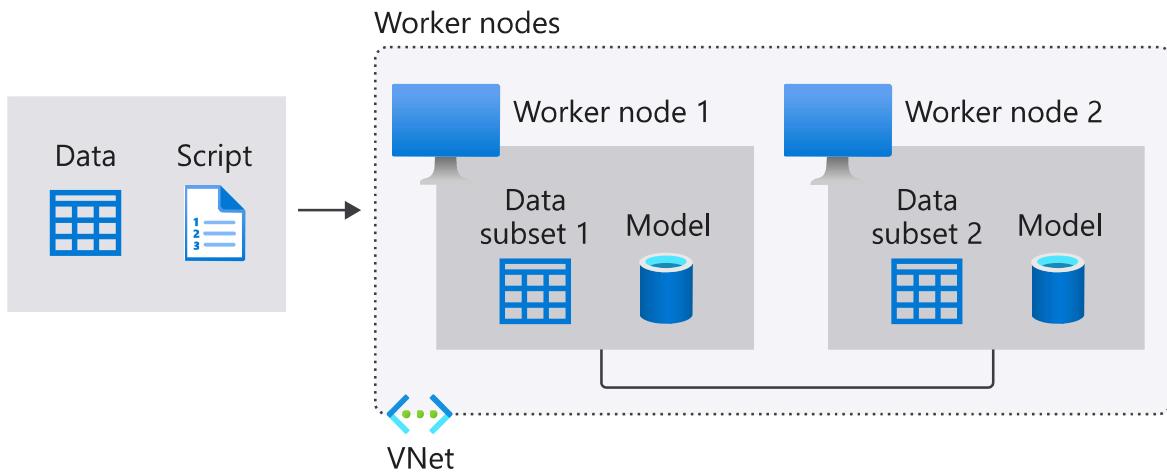
- [Distributed training with PyTorch](#)
- [Distributed training with TensorFlow](#)

For ML models that don't require distributed training, see [train models with Azure Machine Learning](#) for the different ways to train models using the Python SDK.

Data parallelism

Data parallelism is the easiest to implement of the two distributed training approaches, and is sufficient for most use cases.

In this approach, the data is divided into partitions, where the number of partitions is equal to the total number of available nodes, in the compute cluster. The model is copied in each of these worker nodes, and each worker operates on its own subset of the data. Keep in mind that each node has to have the capacity to support the model that's being trained, that is the model has to entirely fit on each node. The following diagram provides a visual demonstration of this approach.



Each node independently computes the errors between its predictions for its training samples and the labeled outputs. In turn, each node updates its model based on the errors and must communicate all of its changes to the other nodes to update their corresponding models. This means that the worker nodes need to synchronize the model parameters, or gradients, at the end of the batch computation to ensure they are training a consistent model.

Model parallelism

In model parallelism, also known as network parallelism, the model is segmented into different parts that can run concurrently in different nodes, and each one will run on the same data. The scalability of this method depends on the degree of task parallelization of the algorithm, and it is more complex to implement than data parallelism.

In model parallelism, worker nodes only need to synchronize the shared parameters, usually once for each forward or backward-propagation step. Also, larger models aren't a concern since each node operates on a subsection of the model on the same training data.

Next steps

- For a technical example, see the [reference architecture scenario](#).
- Find tips for MPI, TensorFlow, and PyTorch in the [Distributed GPU training guide](#)

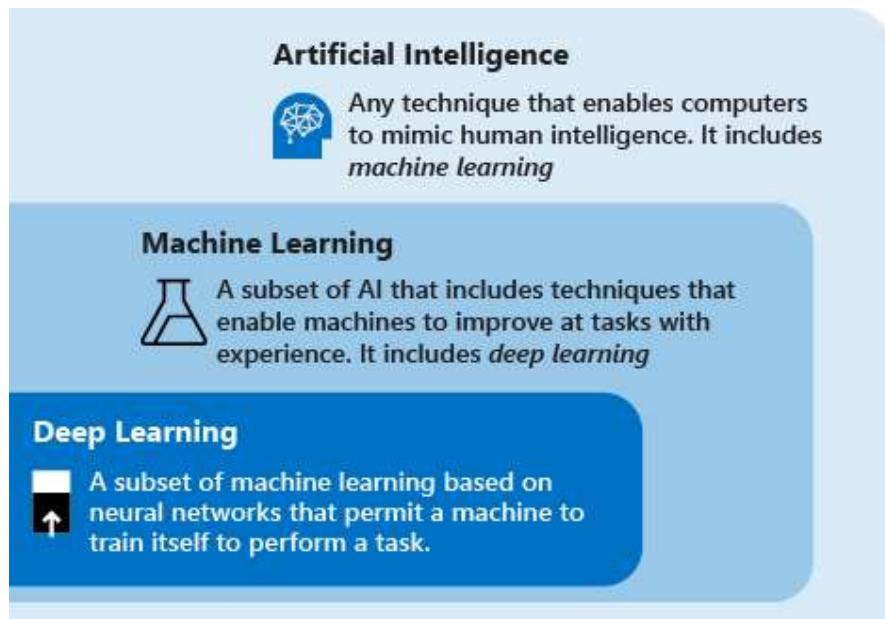
Deep learning vs. machine learning in Azure Machine Learning

Article • 01/20/2023 • 9 minutes to read

This article explains deep learning vs. machine learning and how they fit into the broader category of artificial intelligence. Learn about deep learning solutions you can build on Azure Machine Learning, such as fraud detection, voice and facial recognition, sentiment analysis, and time series forecasting.

For guidance on choosing algorithms for your solutions, see the [Machine Learning Algorithm Cheat Sheet](#).

Deep learning, machine learning, and AI



Consider the following definitions to understand deep learning vs. machine learning vs. AI:

- **Deep learning** is a subset of machine learning that's based on artificial neural networks. The *learning process* is *deep* because the structure of artificial neural networks consists of multiple input, output, and hidden layers. Each layer contains units that transform the input data into information that the next layer can use for a certain predictive task. Thanks to this structure, a machine can learn through its own data processing.
- **Machine learning** is a subset of artificial intelligence that uses techniques (such as deep learning) that enable machines to use experience to improve at tasks. The

learning process is based on the following steps:

1. Feed data into an algorithm. (In this step you can provide additional information to the model, for example, by performing feature extraction.)
 2. Use this data to train a model.
 3. Test and deploy the model.
 4. Consume the deployed model to do an automated predictive task. (In other words, call and use the deployed model to receive the predictions returned by the model.)
- **Artificial intelligence (AI)** is a technique that enables computers to mimic human intelligence. It includes machine learning.

By using machine learning and deep learning techniques, you can build computer systems and applications that do tasks that are commonly associated with human intelligence. These tasks include image recognition, speech recognition, and language translation.

Techniques of deep learning vs. machine learning

Now that you have the overview of machine learning vs. deep learning, let's compare the two techniques. In machine learning, the algorithm needs to be told how to make an accurate prediction by consuming more information (for example, by performing feature extraction). In deep learning, the algorithm can learn how to make an accurate prediction through its own data processing, thanks to the artificial neural network structure.

The following table compares the two techniques in more detail:

	All machine learning	Only deep learning
Number of data points	Can use small amounts of data to make predictions.	Needs to use large amounts of training data to make predictions.
Hardware dependencies	Can work on low-end machines. It doesn't need a large amount of computational power.	Depends on high-end machines. It inherently does a large number of matrix multiplication operations. A GPU can efficiently optimize these operations.
Featurization process	Requires features to be accurately identified and created by users.	Learns high-level features from data and creates new features by itself.

	All machine learning	Only deep learning
Learning approach	Divides the learning process into smaller steps. It then combines the results from each step into one output.	Moves through the learning process by resolving the problem on an end-to-end basis.
Execution time	Takes comparatively little time to train, ranging from a few seconds to a few hours.	Usually takes a long time to train because a deep learning algorithm involves many layers.
Output	The output is usually a numerical value, like a score or a classification.	The output can have multiple formats, like a text, a score or a sound.

What is transfer learning?

Training deep learning models often requires large amounts of training data, high-end compute resources (GPU, TPU), and a longer training time. In scenarios when you don't have any of these available to you, you can shortcut the training process using a technique known as *transfer learning*.

Transfer learning is a technique that applies knowledge gained from solving one problem to a different but related problem.

Due to the structure of neural networks, the first set of layers usually contains lower-level features, whereas the final set of layers contains higher-level features that are closer to the domain in question. By repurposing the final layers for use in a new domain or problem, you can significantly reduce the amount of time, data, and compute resources needed to train the new model. For example, if you already have a model that recognizes cars, you can repurpose that model using transfer learning to also recognize trucks, motorcycles, and other kinds of vehicles.

Learn how to apply transfer learning for image classification using an open-source framework in Azure Machine Learning : [Train a deep learning PyTorch model using transfer learning](#).

Deep learning use cases

Because of the artificial neural network structure, deep learning excels at identifying patterns in unstructured data such as images, sound, video, and text. For this reason, deep learning is rapidly transforming many industries, including healthcare, energy,

finance, and transportation. These industries are now rethinking traditional business processes.

Some of the most common applications for deep learning are described in the following paragraphs. In Azure Machine Learning, you can use a model from you build from an open-source framework or build the model using the tools provided.

Named-entity recognition

Named-entity recognition is a deep learning method that takes a piece of text as input and transforms it into a pre-specified class. This new information could be a postal code, a date, a product ID. The information can then be stored in a structured schema to build a list of addresses or serve as a benchmark for an identity validation engine.

Object detection

Deep learning has been applied in many object detection use cases. Object detection comprises two parts: image classification and then image localization. *Image classification* identifies the image's objects, such as cars or people. *Image localization* provides the specific location of these objects.

Object detection is already used in industries such as gaming, retail, tourism, and self-driving cars.

Image caption generation

Like image recognition, in image captioning, for a given image, the system must generate a caption that describes the contents of the image. When you can detect and label objects in photographs, the next step is to turn those labels into descriptive sentences.

Usually, image captioning applications use convolutional neural networks to identify objects in an image and then use a recurrent neural network to turn the labels into consistent sentences.

Machine translation

Machine translation takes words or sentences from one language and automatically translates them into another language. Machine translation has been around for a long time, but deep learning achieves impressive results in two specific areas: automatic

translation of text (and translation of speech to text) and automatic translation of images.

With the appropriate data transformation, a neural network can understand text, audio, and visual signals. Machine translation can be used to identify snippets of sound in larger audio files and transcribe the spoken word or image as text.

Text analytics

Text analytics based on deep learning methods involves analyzing large quantities of text data (for example, medical documents or expenses receipts), recognizing patterns, and creating organized and concise information out of it.

Companies use deep learning to perform text analysis to detect insider trading and compliance with government regulations. Another common example is insurance fraud: text analytics has often been used to analyze large amounts of documents to recognize the chances of an insurance claim being fraud.

Artificial neural networks

Artificial neural networks are formed by layers of connected nodes. Deep learning models use neural networks that have a large number of layers.

The following sections explore most popular artificial neural network typologies.

Feedforward neural network

The feedforward neural network is the most simple type of artificial neural network. In a feedforward network, information moves in only one direction from input layer to output layer. Feedforward neural networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, and each layer is fully connected to all neurons in the layer before. The last fully connected layer (the output layer) represents the generated predictions.

Recurrent neural network (RNN)

Recurrent neural networks are a widely used artificial neural network. These networks save the output of a layer and feed it back to the input layer to help predict the layer's outcome. Recurrent neural networks have great learning abilities. They're widely used for complex tasks such as time series forecasting, learning handwriting, and recognizing language.

Convolutional neural network (CNN)

A convolutional neural network is a particularly effective artificial neural network, and it presents a unique architecture. Layers are organized in three dimensions: width, height, and depth. The neurons in one layer connect not to all the neurons in the next layer, but only to a small region of the layer's neurons. The final output is reduced to a single vector of probability scores, organized along the depth dimension.

Convolutional neural networks have been used in areas such as video recognition, image recognition, and recommender systems.

Generative adversarial network (GAN)

Generative adversarial networks are generative models trained to create realistic content such as images. It is made up of two networks known as generator and discriminator. Both networks are trained simultaneously. During training, the generator uses random noise to create new synthetic data that closely resembles real data. The discriminator takes the output from the generator as input and uses real data to determine whether the generated content is real or synthetic. Each network is competing with each other. The generator is trying to generate synthetic content that is indistinguishable from real content and the discriminator is trying to correctly classify inputs as real or synthetic. The output is then used to update the weights of both networks to help them better achieve their respective goals.

Generative adversarial networks are used to solve problems like image to image translation and age progression.

Transformers

Transformers are a model architecture that is suited for solving problems containing sequences such as text or time-series data. They consist of [encoder and decoder layers](#). The encoder takes an input and maps it to a numerical representation containing information such as context. The decoder uses information from the encoder to produce an output such as translated text. What makes transformers different from other architectures containing encoders and decoders are the attention sub-layers. Attention is the idea of focusing on specific parts of an input based on the importance of their context in relation to other inputs in a sequence. For example, when summarizing a news article, not all sentences are relevant to describe the main idea. By focusing on key words throughout the article, summarization can be done in a single sentence, the headline.

Transformers have been used to solve natural language processing problems such as translation, text generation, question answering, and text summarization.

Some well-known implementations of transformers are:

- Bidirectional Encoder Representations from Transformers (BERT)
- Generative Pre-trained Transformer 2 (GPT-2)
- Generative Pre-trained Transformer 3 (GPT-3)

Next steps

The following articles show you more options for using open-source deep learning models in [Azure Machine Learning](#):

- [Classify handwritten digits by using a TensorFlow model](#)
 - [Classify handwritten digits by using a TensorFlow estimator and Keras](#)
-

Additional resources

Documentation

[Local inference using ONNX for AutoML image - Azure Machine Learning](#)

Use ONNX with Azure Machine Learning automated ML to make predictions on computer vision models for classification, object detection, and instance segmentation.

[Train and deploy a reinforcement learning model \(preview\) - Azure Machine Learning](#)

Learn how to use Azure Machine Learning reinforcement learning (preview) to train an RL agent to play Pong.

[Tutorial: AutoML- train no-code classification models - Azure Machine Learning](#)

Train a classification model without writing a single line of code using Azure Machine Learning automated ML in the studio UI.

[Avoid overfitting & imbalanced data with AutoML - Azure Machine Learning](#)

Identify and manage common pitfalls of ML models with Azure Machine Learning's automated machine learning solutions.

[Set up AutoML for NLP - Azure Machine Learning](#)

Set up Azure Machine Learning automated ML to train natural language processing models with the Azure Machine Learning Python SDK or the Azure Machine Learning CLI.

[How to select a machine learning algorithm - Azure Machine Learning](#)

How to select Azure Machine Learning algorithms for supervised and unsupervised learning in clustering, classification, or regression experiments.

Tutorial: AutoML- train object detection model - Azure Machine Learning

Train an object detection model to identify if an image contains certain objects with automated ML and the Azure Machine Learning CLI v2 and Python SDK v2.

What is automated ML? AutoML - Azure Machine Learning

Learn how Azure Machine Learning can automatically generate a model by using the parameters and criteria you provide with automated machine learning.

[Show 5 more](#)

Training

Learning paths and modules

[Train and evaluate deep learning models - Training](#)

Train and evaluate deep learning models

From artifacts to models in MLflow

Article • 02/24/2023 • 6 minutes to read

The following article explains the differences between an artifact and a model in MLflow and how to transition from one to the other. It also explains how Azure Machine Learning uses the MLflow model's concept to enable streamlined deployment workflows.

What's the difference between an artifact and a model?

If you are not familiar with MLflow, you may not be aware of the difference between logging artifacts or files vs. logging MLflow models. There are some fundamental differences between the two:

Artifacts

Any file generated (and captured) from an experiment's run or job is an artifact. It may represent a model serialized as a Pickle file, the weights of a PyTorch or TensorFlow model, or even a text file containing the coefficients of a linear regression. Other artifacts can have nothing to do with the model itself, but they can contain configuration to run the model, pre-processing information, sample data, etc. As you can see, an artifact can come in any format.

You may have been logging artifacts already:

```
Python

filename = 'model.pkl'
with open(filename, 'wb') as f:
    pickle.dump(model, f)

mlflow.log_artifact(filename)
```

Models

A model in MLflow is also an artifact. However, we make stronger assumptions about this type of artifacts. Such assumptions provide a clear contract between the saved files and what they mean. When you log your models as artifacts (simple files), you need to know what the model builder meant for each of them in order to know how to load the

model for inference. On the contrary, MLflow models can be loaded using the contract specified in the [The MLModel format](#).

In Azure Machine Learning, logging models has the following advantages:

- ✓ You can deploy them on real-time or batch endpoints without providing an scoring script nor an environment.
- ✓ When deployed, Model's deployments have a Swagger generated automatically and the **Test** feature can be used in Azure Machine Learning studio.
- ✓ Models can be used as pipelines inputs directly.
- ✓ You can use the [Responsible AI dashboard \(preview\)](#).

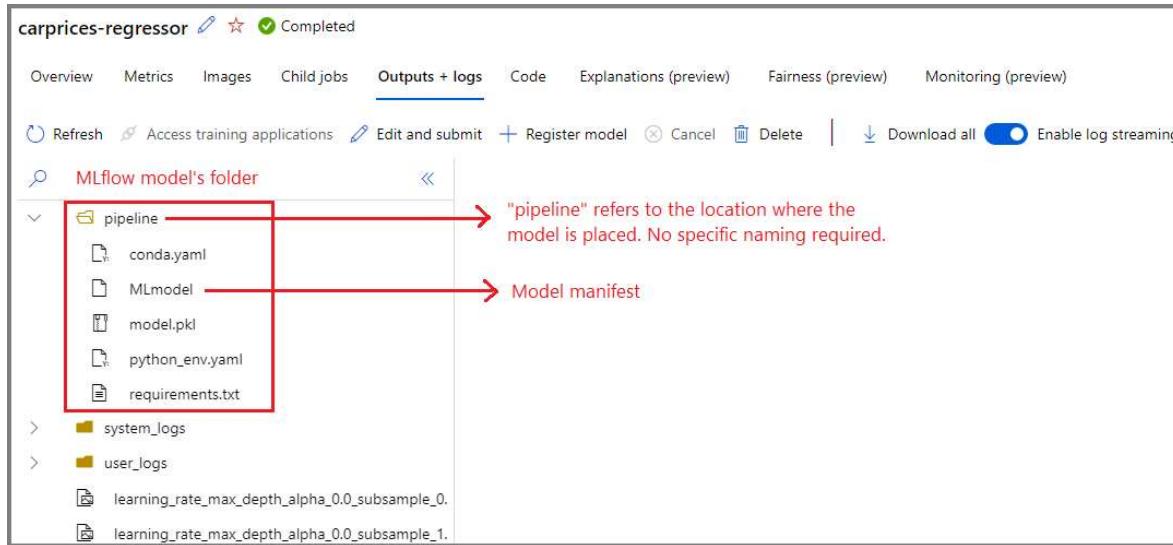
Models can get logged by using MLflow SDK:

Python

```
import mlflow
mlflow.sklearn.log_model(sklearn_estimator, "classifier")
```

The MLmodel format

MLflow adopts the MLmodel format as a way to create a contract between the artifacts and what they represent. The MLmodel format stores assets in a folder. Among them, there is a particular file named `MLmodel`. This file is the single source of truth about how a model can be loaded and used.



The following example shows how the `MLmodel` file for a computer vision model trained with `fastai` may look like:

`MLmodel`

YAML

```
artifact_path: classifier
flavors:
  fastai:
    data: model.fastai
    fastai_version: 2.4.1
  python_function:
    data: model.fastai
    env: conda.yaml
    loader_module: mlflow.fastai
    python_version: 3.8.12
model_uuid: e694c68eba484299976b06ab9058f636
run_id: e13da8ac-b1e6-45d4-a9b2-6a0a5cfac537
signature:
  inputs: '[{"type": "tensor",
            "tensor-spec":
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}
            }'
  outputs: '[{"type": "tensor",
             "tensor-spec":
               {"dtype": "float32", "shape": [-1, 2]}
             }]'
```

The model's flavors

Considering the variety of machine learning frameworks available to use, MLflow introduced the concept of flavor as a way to provide a unique contract to work across all of them. A flavor indicates what to expect for a given model created with a specific framework. For instance, TensorFlow has its own flavor, which specifies how a TensorFlow model should be persisted and loaded. Because each model flavor indicates how they want to persist and load models, the MLModel format doesn't enforce a single serialization mechanism that all the models need to support. Such decision allows each flavor to use the methods that provide the best performance or best support according to their best practices - without compromising compatibility with the MLModel standard.

The following is an example of the `flavors` section for an `fastai` model.

YAML

```
flavors:
  fastai:
    data: model.fastai
    fastai_version: 2.4.1
  python_function:
    data: model.fastai
    env: conda.yaml
```

```
loader_module: mlflow.fastai  
python_version: 3.8.12
```

Signatures

Model signatures in MLflow [↗](#) are an important part of the model specification, as they serve as a data contract between the model and the server running our models. They are also important for parsing and enforcing model's input's types at deployment time. MLflow enforces types when data is submitted to your model if a signature is available [↗](#).

Signatures are indicated when the model gets logged and persisted in the `MLmodel` file, in the `signature` section. Autolog's feature in MLflow automatically infers signatures in a best effort way. However, it may be required to [log the models manually if the signatures inferred are not the ones you need ↗](#).

There are two types of signatures:

- **Column-based signature** corresponding to signatures that operate to tabular data. For models with this signature, MLflow supplies `pandas.DataFrame` objects as inputs.
- **Tensor-based signature**: corresponding to signatures that operate with n-dimensional arrays or tensors. For models with this signature, MLflow supplies `numpy.ndarray` as inputs (or a dictionary of `numpy.ndarray` in the case of named-tensors).

The following example corresponds to a computer vision model trained with `fastai`.

This model receives a batch of images represented as tensors of shape `(300, 300, 3)` with the RGB representation of them (unsigned integers). It outputs batches of predictions (probabilities) for two classes.

MLmodel

YAML

```
signature:  
  inputs: '[{"type": "tensor",  
            "tensor-spec":  
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}  
          }]  
  outputs: '[{"type": "tensor",  
             "tensor-spec":  
               {"dtype": "float32", "shape": [-1, 2]}  
           }]'
```

💡 Tip

Azure Machine Learning generates Swagger for model's deployment in MLflow format with a signature available. This makes easier to test deployed endpoints using the Azure Machine Learning studio.

Model's environment

Requirements for the model to run are specified in the `conda.yaml` file. Dependencies can be automatically detected by MLflow or they can be manually indicated when you call `mlflow.<flavor>.log_model()` method. The latter can be needed in cases that the libraries included in your environment are not the ones you intended to use.

The following is an example of an environment used for a model created with `fastai` framework:

conda.yaml

YAML

```
channels:
- conda-forge
dependencies:
- python=3.8.5
- pip
- pip:
  - mlflow
  - astunparse==1.6.3
  - cffi==1.15.0
  - configparser==3.7.4
  - defusedxml==0.7.1
  - fastai==2.4.1
  - google-api-core==2.7.1
  - ipython==8.2.0
  - psutil==5.9.0
name: mlflow-env
```

⚠ Note

MLflow environments and Azure Machine Learning environments are different concepts. While the former operates at the level of the model, the latter operates at the level of the workspace (for registered environments) or jobs/deployments (for anonymous environments). When you deploy MLflow models in Azure Machine Learning, the model's environment is built and used for deployment.

Alternatively, you can override this behaviour with the [Azure Machine Learning CLI v2](#) and deploy MLflow models using a specific Azure Machine Learning environments.

Model's predict function

All MLflow models contain a `predict` function. **This function is the one that is called when a model is deployed using a no-code-deployment experience.** What the `predict` function returns (classes, probabilities, a forecast, etc.) depend on the framework (i.e. flavor) used for training. Read the documentation of each flavor to know what they return.

In some cases, you may need to customize this function to change the way inference is executed. On those cases, you will need to [log models with a different behavior in the predict method](#) or [log a custom model's flavor](#).

Loading MLflow models back

Models created as MLflow models can be loaded back directly from the run where they were logged, from the file system where they are saved or from the model registry where they are registered. MLflow provides a consistent way to load those models regardless of the location.

There are two workflows available for loading models:

- **Loading back the same object and types that were logged:** You can load models using MLflow SDK and obtain an instance of the model with types belonging to the training library. For instance, an ONNX model will return a `ModelProto` while a decision tree trained with Scikit-Learn model will return a `DecisionTreeClassifier` object. Use `mlflow.<flavor>.load_model()` to do so.
- **Loading back a model for running inference:** You can load models using MLflow SDK and obtain a wrapper where MLflow guarantees there will be a `predict` function. It doesn't matter which flavor you are using, every MLflow model needs to implement this contract. Furthermore, MLflow guarantees that this function can be called using arguments of type `pandas.DataFrame`, `numpy.ndarray` or `dict[string, numpyndarray]` (depending on the signature of the model). MLflow handles the type conversion to the input type the model actually expects. Use `mlflow.pyfunc.load_model()` to do so.

Start logging models

We recommend starting taking advantage of MLflow models in Azure Machine Learning. There are different ways to start using the model's concept with MLflow. Read [How to log MLFlow models](#) to a comprehensive guide.

Additional resources

Documentation

[Using MLflow models in batch deployments - Azure Machine Learning](#)

Learn how to deploy MLflow models in batch deployments

[Train with MLflow Projects \(Preview\) - Azure Machine Learning](#)

Set up MLflow with Azure Machine Learning to log metrics and artifacts from ML models

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Configure MLflow for Azure Machine Learning - Azure Machine Learning](#)

Connect MLflow to Azure Machine Learning workspaces to log metrics, artifacts and deploy models.

[Manage models registries in Azure Machine Learning with MLflow - Azure Machine Learning](#)

Explains how to use MLflow for managing models in Azure Machine Learning

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Machine Learning registries \(preview\) - Azure Machine Learning](#)

Learn what are Azure Machine Learning registries and how to use to for MLOps

[Show 5 more](#)

Training

Learning paths and modules

[Use MLflow with Azure Machine Learning jobs submitted with CLI \(v2\) - Training](#)

Use MLflow with Azure Machine Learning jobs submitted with CLI (v2)

What is automated machine learning (AutoML)?

Article • 02/24/2023 • 10 minutes to read

APPLIES TO:  Python SDK azure-ai-ml v2 (current) ↗

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. Automated ML in Azure Machine Learning is based on a breakthrough from our [Microsoft Research division](#) ↗.

- For code-experienced customers, [Azure Machine Learning Python SDK](#) ↗ . Get started with [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#).

How does AutoML work?

During training, Azure Machine Learning creates a number of 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 will stop once it hits the exit criteria defined in the experiment.

Using [Azure Machine Learning](#), you can design and run your automated ML training experiments with these steps:

1. **Identify the ML problem** to be solved: classification, forecasting, regression, computer vision or NLP.
2. **Choose whether you want a code-first experience or a no-code studio web experience:** Users who prefer a code-first experience can use the [Azure Machine Learning SDKv2](#) or the [Azure Machine Learning CLIV2](#). Get started with [Tutorial: Train an object detection model with AutoML and Python](#). Users who prefer a limited/no-code experience can use the [web interface](#) in Azure Machine Learning studio at <https://ml.azure.com> ↗ . Get started with [Tutorial: Create a classification model with automated ML in Azure Machine Learning](#).

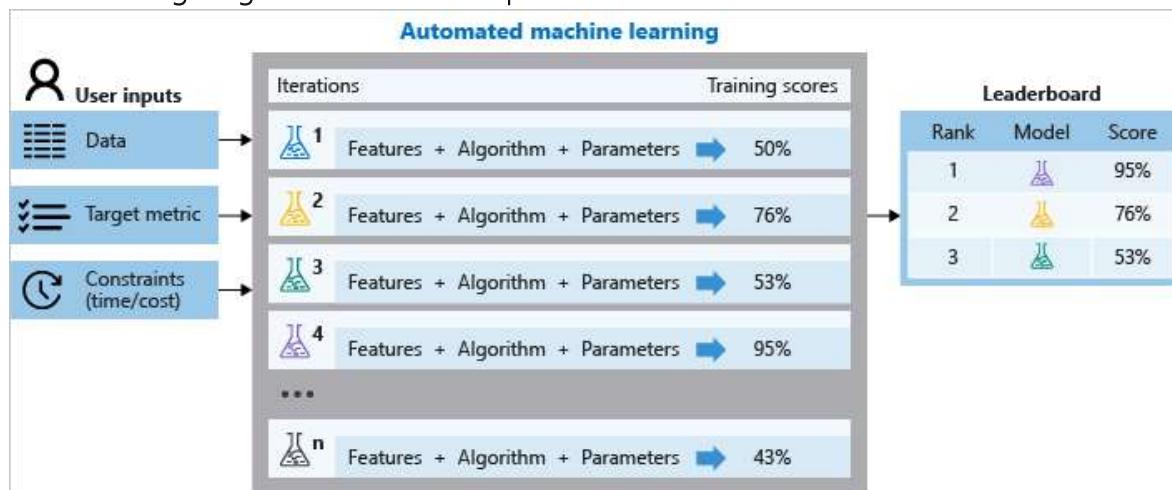
3. Specify the source of the labeled training data: You can bring your data to Azure Machine Learning in [many different ways](#).

4. Configure the automated machine learning parameters that determine how many iterations over different models, hyperparameter settings, advanced preprocessing/featurization, and what metrics to look at when determining the best model.

5. Submit the training job.

6. Review the results

The following diagram illustrates this process.



You can also inspect the logged job information, which [contains metrics](#) gathered during the job. The training job produces a Python serialized object (`.pk1` file) that contains the model and data preprocessing.

While model building is automated, you can also [learn how important or relevant features are](#) to the generated models.

When to use AutoML: classification, regression, forecasting, computer vision & NLP

Apply automated ML when you want Azure Machine Learning to train and tune a model for you using the target metric you specify. Automated ML democratizes the machine learning model development process, and empowers its users, no matter their data science expertise, to identify an end-to-end machine learning pipeline for any problem.

ML professionals and developers across industries can use automated ML to:

- Implement ML solutions without extensive programming knowledge

- Save time and resources
- Leverage data science best practices
- Provide agile problem-solving

Classification

Classification is a type of supervised learning in which models learn using training data, and apply those learnings to new data. Azure Machine Learning offers featurizations specifically for these tasks, such as deep neural network text featurizers for classification. Learn more about [featurization options](#). You can also find the list of algorithms supported by AutoML [here](#).

The main goal of classification models is to predict which categories new data will fall into based on learnings from its training data. Common classification examples include fraud detection, handwriting recognition, and object detection.

See an example of classification and automated machine learning in this Python notebook: [Bank Marketing](#).

Regression

Similar to classification, regression tasks are also a common supervised learning task. Azure Machine Learning offers featurization specific to regression problems. Learn more about [featurization options](#). You can also find the list of algorithms supported by AutoML [here](#).

Different from classification where predicted output values are categorical, regression models predict numerical output values based on independent predictors. In regression, the objective is to help establish the relationship among those independent predictor variables by estimating how one variable impacts the others. For example, automobile price based on features like, gas mileage, safety rating, etc.

See an example of regression and automated machine learning for predictions in these Python notebooks: [Hardware Performance](#).

Time-series forecasting

Building forecasts is an integral part of any business, whether it's revenue, inventory, sales, or customer demand. You can use automated ML to combine techniques and approaches and get a recommended, high-quality time-series forecast. You can find the list of algorithms supported by AutoML [here](#).

An automated time-series experiment is treated as a multivariate regression problem. Past time-series values are "pivoted" to become additional dimensions for the regressor together with other predictors. This approach, unlike classical time series methods, has an advantage of naturally incorporating multiple contextual variables and their relationship to one another during training. Automated ML learns a single, but often internally branched model for all items in the dataset and prediction horizons. More data is thus available to estimate model parameters and generalization to unseen series becomes possible.

Advanced forecasting configuration includes:

- holiday detection and featurization
- time-series and DNN learners (Auto-ARIMA, Prophet, ForecastTCN)
- many models support through grouping
- rolling-origin cross validation
- configurable lags
- rolling window aggregate features

See an example of forecasting and automated machine learning in this Python notebook: [Energy Demand ↗](#).

Computer vision

Support for computer vision tasks allows you to easily generate models trained on image data for scenarios like image classification and object detection.

With this capability you can:

- Seamlessly integrate with the [Azure Machine Learning data labeling](#) capability
- Use labeled data for generating image models
- Optimize model performance by specifying the model algorithm and tuning the hyperparameters.
- Download or deploy the resulting model as a web service in Azure Machine Learning.
- Operationalize at scale, leveraging Azure Machine Learning [MLOps](#) and [ML Pipelines](#) capabilities.

Authoring AutoML models for vision tasks is supported via the Azure Machine Learning Python SDK. The resulting experimentation jobs, models, and outputs can be accessed from the Azure Machine Learning studio UI.

Learn how to [set up AutoML training for computer vision models](#).

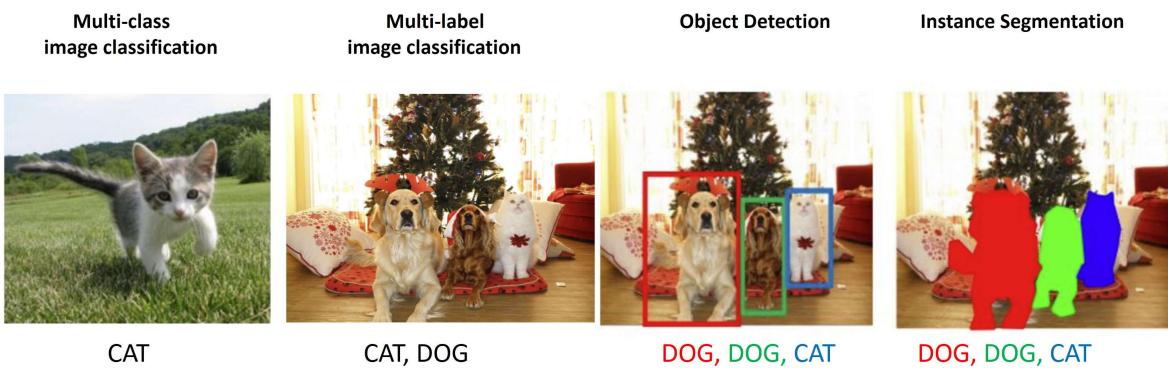


Image from: http://cs231n.stanford.edu/slides/2021/lecture_15.pdf

Automated ML for images supports the following computer vision tasks:

Task	Description
Multi-class image classification	Tasks where an image is classified with only a single label from a set of classes - e.g. each image is classified as either an image of a 'cat' or a 'dog' or a 'duck'
Multi-label image classification	Tasks where an image could have one or more labels from a set of labels - e.g. an image could be labeled with both 'cat' and 'dog'
Object detection	Tasks to identify objects in an image and locate each object with a bounding box e.g. locate all dogs and cats in an image and draw a bounding box around each.
Instance segmentation	Tasks to identify objects in an image at the pixel level, drawing a polygon around each object in the image.

Natural language processing: NLP

Support for natural language processing (NLP) tasks in automated ML allows you to easily generate models trained on text data for text classification and named entity recognition scenarios. Authoring automated ML trained NLP models is supported via the Azure Machine Learning Python SDK. The resulting experimentation jobs, models, and outputs can be accessed from the Azure Machine Learning studio UI.

The NLP capability supports:

- End-to-end deep neural network NLP training with the latest pre-trained BERT models
- Seamless integration with [Azure Machine Learning data labeling](#)
- Use labeled data for generating NLP models
- Multi-lingual support with 104 languages
- Distributed training with Horovod

Learn how to [set up AutoML training for NLP models](#).

Training, validation and test data

With automated ML you provide the **training data** to train ML models, and you can specify what type of model validation to perform. Automated ML performs model validation as part of training. That is, automated ML uses **validation data** to tune model hyperparameters based on the applied algorithm to find the combination that best fits the training data. However, the same validation data is used for each iteration of tuning, which introduces model evaluation bias since the model continues to improve and fit to the validation data.

To help confirm that such bias isn't applied to the final recommended model, automated ML supports the use of **test data** to evaluate the final model that automated ML recommends at the end of your experiment. When you provide test data as part of your AutoML experiment configuration, this recommended model is tested by default at the end of your experiment (preview).

Important

Testing your models with a test dataset to evaluate generated models is a preview feature. This capability is an [experimental](#) preview feature, and may change at any time.

Learn how to [configure AutoML experiments to use test data \(preview\)](#) with the SDK or with the [Azure Machine Learning studio](#).

Feature engineering

Feature engineering is the process of using domain knowledge of the data to create features that help ML algorithms learn better. In Azure Machine Learning, scaling and normalization techniques are applied to facilitate feature engineering. Collectively, these techniques and feature engineering are referred to as featurization.

For automated machine learning experiments, featurization is applied automatically, but can also be customized based on your data. [Learn more about what featurization is included](#) and how AutoML helps [prevent over-fitting and imbalanced data](#) in your models.

Note

Automated machine learning featurization steps (feature normalization, handling missing data, converting text to numeric, etc.) become part of the underlying model. When using the model for predictions, the same featurization steps applied during training are applied to your input data automatically.

Customize featurization

Additional feature engineering techniques such as, encoding and transforms are also available.

Enable this setting with:

- Azure Machine Learning studio: Enable **Automatic featurization** in the **View additional configuration** section [with these steps](#).
- Python SDK: Specify featurization in your [AutoML Job](#) object. Learn more about [enabling featurization](#).

Ensemble models

Automated machine learning supports ensemble models, which are enabled by default. Ensemble learning improves machine learning results and predictive performance by combining multiple models as opposed to using single models. The ensemble iterations appear as the final iterations of your job. Automated machine learning uses both voting and stacking ensemble methods for combining models:

- **Voting:** predicts based on the weighted average of predicted class probabilities (for classification tasks) or predicted regression targets (for regression tasks).
- **Stacking:** stacking combines heterogenous models and trains a meta-model based on the output from the individual models. The current default meta-models are LogisticRegression for classification tasks and ElasticNet for regression/forecasting tasks.

The [Caruana ensemble selection algorithm](#) with sorted ensemble initialization is used to decide which models to use within the ensemble. At a high level, this algorithm initializes the ensemble with up to five models with the best individual scores, and verifies that these models are within 5% threshold of the best score to avoid a poor initial ensemble. Then for each ensemble iteration, a new model is added to the existing ensemble and the resulting score is calculated. If a new model improved the existing ensemble score, the ensemble is updated to include the new model.

See the [AutoML package](#) for changing default ensemble settings in automated machine learning.

AutoML & ONNX

With Azure Machine Learning, you can use automated ML to build a Python model and have it converted to the ONNX format. Once the models are in the ONNX format, they can be run on a variety of platforms and devices. Learn more about [accelerating ML models with ONNX](#).

See how to convert to ONNX format [in this Jupyter notebook example](#). Learn which [algorithms are supported in ONNX](#).

The ONNX runtime also supports C#, so you can use the model built automatically in your C# apps without any need for recoding or any of the network latencies that REST endpoints introduce. Learn more about [using an AutoML ONNX model in a .NET application with ML.NET](#) and [inferencing ONNX models with the ONNX runtime C# API](#).

Next steps

There are multiple resources to get you up and running with AutoML.

Tutorials/ how-tos

Tutorials are end-to-end introductory examples of AutoML scenarios.

- **For a code first experience**, follow the [Tutorial: Train an object detection model with AutoML and Python](#)
- **For a low or no-code experience**, see the [Tutorial: Train a classification model with no-code AutoML in Azure Machine Learning studio](#).

How-to articles provide additional detail into what functionality automated ML offers.

For example,

- Configure the settings for automatic training experiments
 - [Without code in the Azure Machine Learning studio](#).
 - [With the Python SDK](#).
- Learn how to [train computer vision models with Python](#).
- Learn how to [view the generated code from your automated ML models](#).

Jupyter notebook samples

Review detailed code examples and use cases in the [GitHub notebook repository for automated machine learning samples](<https://github.com/Azure/azureml-examples/tree/main/sdk/python/jobs/automl-standalone-jobs>) .

Python SDK reference

Deepen your expertise of SDK design patterns and class specifications with the [AutoML Job class reference documentation](#).

Note

Automated machine learning capabilities are also available in other Microsoft solutions such as, [ML.NET](#), [HDInsight](#), [Power BI](#) and [SQL Server](#)

Additional resources

Documentation

[Avoid overfitting & imbalanced data with AutoML - Azure Machine Learning](#)

Identify and manage common pitfalls of ML models with Azure Machine Learning's automated machine learning solutions.

[Machine learning fairness - Azure Machine Learning](#)

Learn about machine learning fairness and how the Fairlearn Python package can help you assess and mitigate unfairness.

[Tutorial: AutoML- train no-code classification models - Azure Machine Learning](#)

Train a classification model without writing a single line of code using Azure Machine Learning automated ML in the studio UI.

[What is Responsible AI - Azure Machine Learning](#)

Learn what Responsible AI is and how to use it with Azure Machine Learning to understand models, protect data, and control the model lifecycle.

[Example pipelines & datasets for the designer - Azure Machine Learning](#)

Learn how to use samples in Azure Machine Learning designer to jumps-start your machine learning pipelines.

[AutoML-train regression model \(SDK v1\) - Azure Machine Learning](#)

Train a regression model to predict NYC taxi fares with the Azure Machine Learning Python SDK using Azure Machine Learning automated ML SDK (v1).

[Evaluate AutoML experiment results - Azure Machine Learning](#)

Learn how to view and evaluate charts and metrics for each of your automated machine learning experiment jobs.

[Set up AutoML with the studio UI - Azure Machine Learning](#)

Learn how to set up AutoML training jobs without a single line of code with Azure Machine Learning automated ML in the Azure Machine Learning studio.

[Show 5 more](#)

Training

Learning paths and modules

[Use AutoML to train a labeled dataset and develop a production model - Training](#)

Use AutoML to train a labeled dataset and develop a production model.

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Overview of forecasting methods in AutoML

Article • 02/24/2023 • 11 minutes to read

This article focuses on the methods that AutoML uses to prepare time series data and build forecasting models. Instructions and examples for training forecasting models in AutoML can be found in our [set up AutoML for time series forecasting](#) article.

AutoML uses several methods to forecast time series values. These methods can be roughly assigned to two categories:

1. Time series models that use historical values of the target quantity to make predictions into the future.
2. Regression, or explanatory, models that use predictor variables to forecast values of the target.

As an example, consider the problem of forecasting daily demand for a particular brand of orange juice from a grocery store. Let y_t represent the demand for this brand on day t . A **time series model** predicts demand at $t + 1$ using some function of historical demand,

$$y_{t+1} = f(y_t, y_{t-1}, \dots, y_{t-s}).$$

The function f often has parameters that we tune using observed demand from the past. The amount of history that f uses to make predictions, s , can also be considered a parameter of the model.

The time series model in the orange juice demand example may not be accurate enough since it only uses information about past demand. There are many other factors that likely influence future demand such as price, day of the week, and whether it's a holiday or not. Consider a **regression model** that uses these predictor variables,

$$y = g(\text{price}, \text{day of week}, \text{holiday}).$$

Again, g generally has a set of parameters, including those governing regularization, that AutoML tunes using past values of the demand and the predictors. We omit t from the expression to emphasize that the regression model uses correlational patterns between *contemporaneously* defined variables to make predictions. That is, to predict y_{t+1} from g , we must know which day of the week $t + 1$ falls on, whether it's a holiday, and the orange juice price on day $t + 1$. The first two pieces of information are always easily found by consulting a calendar. A retail price is usually set in advance, so the price of

orange juice is likely also known one day ahead. However, the price may not be known 10 days into the future! It's important to understand that the utility of this regression is limited by how far into the future we need forecasts, also called the **forecast horizon**, and to what degree we know the future values of the predictors.

ⓘ Important

AutoML's forecasting regression models assume that all features provided by the user are known into the future, at least up to the forecast horizon.

AutoML's forecasting regression models can also be augmented to use historical values of the target and predictors. The result is a hybrid model with characteristics of a time series model and a pure regression model. Historical quantities are additional predictor variables in the regression and we refer to them as **lagged quantities**. The *order* of the lag refers to how far back the value is known. For example, the current value of an order-two lag of the target for our orange juice demand example is the observed juice demand from two days ago.

Another notable difference between the time series models and the regression models is in the way they generate forecasts. Time series models are generally defined by recursion relations and produce forecasts one-at-a-time. To forecast many periods into the future, they iterate up-to the forecast horizon, feeding previous forecasts back into the model to generate the next one-period-ahead forecast as needed. In contrast, the regression models are so-called **direct forecasters** that generate *all* forecasts up to the horizon in one go. Direct forecasters can be preferable to recursive ones because recursive models compound prediction error when they feed previous forecasts back into the model. When lag features are included, AutoML makes some important modifications to the training data so that the regression models can function as direct forecasters. See the [lag features article](#) for more details.

Forecasting models in AutoML

The following table lists the forecasting models implemented in AutoML and what category they belong to:

Time Series Models	Regression Models
Naive, Seasonal Naive, Average, Seasonal Average [↗] , ARIMA(X) [↗] , Exponential Smoothing [↗]	Linear SGD [↗] , LARS LASSO [↗] , Elastic Net [↗] , Prophet [↗] , K Nearest Neighbors [↗] , Decision Tree [↗] , Random Forest [↗] , Extremely Randomized Trees [↗] , Gradient Boosted Trees [↗] , LightGBM [↗] , XGBoost [↗] , ForecastTCN

The models in each category are listed roughly in order of the complexity of patterns they're able to incorporate, also known as the **model capacity**. A Naive model, which simply forecasts the last observed value, has low capacity while the Temporal Convolutional Network (ForecastTCN), a deep neural network with potentially millions of tunable parameters, has high capacity.

Importantly, AutoML also includes **ensemble** models that create weighted combinations of the best performing models to further improve accuracy. For forecasting, we use a [soft voting ensemble](#) where composition and weights are found via the [Caruana Ensemble Selection Algorithm](#).

ⓘ Note

There are two important caveats for forecast model ensembles:

1. The TCN cannot currently be included in ensembles.
2. AutoML by default disables another ensemble method, the **stack ensemble**, which is included with default regression and classification tasks in AutoML.

The stack ensemble fits a meta-model on the best model forecasts to find ensemble weights. We've found in internal benchmarking that this strategy has an increased tendency to over fit time series data. This can result in poor generalization, so the stack ensemble is disabled by default. However, it can be enabled if desired in the AutoML configuration.

How AutoML uses your data

AutoML accepts time series data in tabular, "wide" format; that is, each variable must have its own corresponding column. AutoML requires one of the columns to be the time axis for the forecasting problem. This column must be parsable into a datetime type. The simplest time series data set consists of a **time column** and a numeric **target column**. The target is the variable one intends to predict into the future. The following is an example of the format in this simple case:

timestamp	quantity
2012-01-01	100
2012-01-02	97
2012-01-03	106

timestamp	quantity
...	...
2013-12-31	347

In more complex cases, the data may contain other columns aligned with the time index.

timestamp	SKU	price	advertised	quantity
2012-01-01	JUICE1	3.5	0	100
2012-01-01	BREAD3	5.76	0	47
2012-01-02	JUICE1	3.5	0	97
2012-01-02	BREAD3	5.5	1	68
...
2013-12-31	JUICE1	3.75	0	347
2013-12-31	BREAD3	5.7	0	94

In this example, there's a SKU, a retail price, and a flag indicating whether an item was advertised in addition to the timestamp and target quantity. There are evidently two series in this dataset - one for the JUICE1 SKU and one for the BREAD3 SKU; the `SKU` column is a **time series ID column** since grouping by it gives two groups containing a single series each. Before sweeping over models, AutoML does basic validation of the input configuration and data and adds engineered features.

Data length requirements

To train a forecasting model, you must have a sufficient amount of historical data. This threshold quantity varies with the training configuration. If you've provided validation data, the minimum number of training observations required per time series is given by,

$$T_{\text{user validation}} = H + \max(l_{\text{max}}, s_{\text{window}}) + 1,$$

where H is the forecast horizon, l_{max} is the maximum lag order, and s_{window} is the window size for rolling aggregation features. If you're using cross-validation, the minimum number of observations is,

$$T_{\text{CV}} = 2H + (n_{\text{CV}} - 1)n_{\text{step}} + \max(l_{\text{max}}, s_{\text{window}}) + 1,$$

where n_{CV} is the number of cross-validation folds and n_{step} is the CV step size, or offset between CV folds. The basic logic behind these formulas is that you should always have at least a horizon of training observations for each time series, including some padding for lags and cross-validation splits. See [forecasting model selection](#) for more details on cross-validation for forecasting.

Missing data handling

AutoML's time series models require regularly spaced observations in time. Regularly spaced, here, includes cases like monthly or yearly observations where the number of days between observations may vary. Prior to modeling, AutoML must ensure there are no missing series values *and* that the observations are regular. Hence, there are two missing data cases:

- A value is missing for some cell in the tabular data
- A *row* is missing which corresponds with an expected observation given the time series frequency

In the first case, AutoML imputes missing values using common, configurable techniques.

An example of a missing, expected row is shown in the following table:

timestamp	quantity
2012-01-01	100
2012-01-03	106
2012-01-04	103
...	...
2013-12-31	347

This series ostensibly has a daily frequency, but there's no observation for Jan. 2, 2012. In this case, AutoML will attempt to fill in the data by adding a new row for Jan. 2, 2012. The new value for the `quantity` column, and any other columns in the data, will then be imputed like other missing values. Clearly, AutoML must know the series frequency in order to fill in observation gaps like this. AutoML automatically detects this frequency, or, optionally, the user can provide it in the configuration.

The imputation method for filling missing values can be [configured](#) in the input. The default methods are listed in the following table:

Column Type	Default Imputation Method
Target	Forward fill (last observation carried forward)
Numeric Feature	Median value

Missing values for categorical features are handled during numerical encoding by including an additional category corresponding to a missing value. Imputation is implicit in this case.

Automated feature engineering

AutoML generally adds new columns to user data to increase modeling accuracy. Engineered feature can include the following:

Feature Group	Default/Optional
Calendar features derived from the time index (for example, day of week)	Default
Categorical features derived from time series IDs	Default
Encoding categorical types to numeric type	Default
Indicator features for holidays associated with a given country or region	Optional
Lags of target quantity	Optional
Lags of feature columns	Optional
Rolling window aggregations (for example, rolling average) of target quantity	Optional
Seasonal decomposition (STL 	Optional

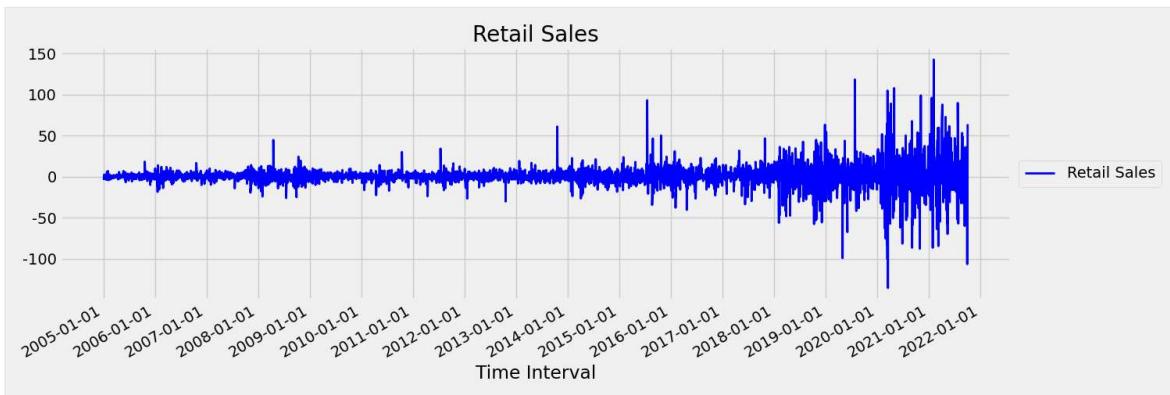
You can configure featurization from the AutoML SDK via the [ForecastingJob](#) class or from the [Azure Machine Learning Studio](#) web interface.

Non-stationary time series detection and handling

A time series where mean and variance change over time is called a **non-stationary**. For example, time series that exhibit stochastic trends are non-stationary by nature. To visualize this, the following image plots a series that is generally trending upward. Now, compute and compare the mean (average) values for the first and the second half of the series. Are they the same? Here, the mean of the series in the first half of the plot is significantly smaller than in the second half. The fact that the mean of the series depends on the time interval one is looking at, is an example of the time-varying moments. Here, the mean of a series is the first moment.



Next, let's examine the following image, which plots the original series in first differences, $\Delta y_t = y_t - y_{t-1}$. The mean of the series is roughly constant over the time range while the variance appears to vary. Thus, this is an example of a first order stationary times series.



AutoML regression models can't inherently deal with stochastic trends, or other well-known problems associated with non-stationary time series. As a result, out-of-sample forecast accuracy can be poor if such trends are present.

AutoML automatically analyzes time series dataset to determine stationarity. When non-stationary time series are detected, AutoML applies a differencing transform automatically to mitigate the impact of non-stationary behavior.

Model sweeping

After data has been prepared with missing data handling and feature engineering, AutoML sweeps over a set of models and hyper-parameters using a [model recommendation service](#). The models are ranked based on validation or cross-validation metrics and then, optionally, the top models may be used in an ensemble model. The best model, or any of the trained models, can be inspected, downloaded, or deployed to produce forecasts as needed. See the [model sweeping and selection](#) article for more details.

Model grouping

When a dataset contains more than one time series, as in the given data example, there are multiple ways to model that data. For instance, we may simply group by the **time series ID column(s)** and train independent models for each series. A more general approach is to partition the data into groups that may each contain multiple, likely related series and train a model per group. By default, AutoML forecasting uses a mixed approach to model grouping. Time series models, plus ARIMAX and Prophet, assign one series to one group and other regression models assign all series to a single group. The following table summarizes the model groupings in two categories, one-to-one and many-to-one:

Each Series in Own Group (1:1)	All Series in Single Group (N:1)
Naive, Seasonal Naive, Average, Seasonal Average, Exponential Smoothing, ARIMA, ARIMAX, Prophet	Linear SGD, LARS LASSO, Elastic Net, K Nearest Neighbors, Decision Tree, Random Forest, Extremely Randomized Trees, Gradient Boosted Trees, LightGBM, XGBoost, ForecastTCN

More general model groupings are possible via AutoML's Many-Models solution; see our [Many Models- Automated ML notebook ↗](#) and [Hierarchical time series- Automated ML notebook ↗](#).

Next steps

- Learn more about [model sweeping and selection](#) for forecasting in AutoML.
- Learn about how AutoML creates [features from the calendar](#).
- Learn about how AutoML creates [lag features](#).
- Read answers to [frequently asked questions](#) about forecasting in AutoML.

Additional resources

Documentation

[Model explainability in automated ML \(preview\) - Azure Machine Learning](#)

Learn how to get explanations for how your automated ML model determines feature importance and makes predictions when using the Azure Machine Learning SDK.

[Share Responsible AI insights and make data-driven decisions with Azure Machine Learning Responsible AI scorecard - Azure Machine Learning](#)

Learn about how to use the Responsible AI scorecard to share responsible AI insights from your machine learning models and make data-driven decisions with non-technical and technical

stakeholders.

[Set up AutoML with Python - Azure Machine Learning](#)

Learn how to set up an AutoML training run with the Azure Machine Learning Python SDK using Azure Machine Learning automated ML.

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Prepare data for computer vision tasks - Azure Machine Learning](#)

Image data preparation for Azure Machine Learning automated ML to train computer vision models on classification, object detection, and segmentation

[Algorithm & component reference \(v2\) - Azure Machine Learning](#)

Learn about the Azure Machine Learning designer components that you can use to create your own machine learning projects. (v2)

[Model sweeping and selection for forecasting in AutoML - Azure Machine Learning](#)

Learn how Azure Machine Learning's AutoML searches for and selects forecasting models

[Create and explore datasets with labels - Azure Machine Learning](#)

Learn how to export data labels from your Azure Machine Learning labeling projects and use them for machine learning tasks.

[Show 5 more](#)

Model sweeping and selection for forecasting in AutoML

Article • 02/03/2023 • 3 minutes to read

This article focuses on how AutoML searches for and selects forecasting models. Please see the [methods overview article](#) for more general information about forecasting methodology in AutoML. Instructions and examples for training forecasting models in AutoML can be found in our [set up AutoML for time series forecasting](#) article.

Model sweeping

The central task for AutoML is to train and evaluate several models and choose the best one with respect to the given primary metric. The word "model" here refers to both the model class - such as ARIMA or Random Forest - and the specific hyper-parameter settings which distinguish models within a class. For instance, ARIMA refers to a class of models that share a mathematical template and a set of statistical assumptions. Training, or fitting, an ARIMA model requires a list of positive integers that specify the precise mathematical form of the model; these are the hyper-parameters. ARIMA(1, 0, 1) and ARIMA(2, 1, 2) have the same class, but different hyper-parameters and, so, can be separately fit with the training data and evaluated against each other. AutoML searches, or *sweeps*, over different model classes and within classes by varying hyper-parameters.

The following table shows the different hyper-parameter sweeping methods that AutoML uses for different model classes:

Model class group	Model type	Hyper-parameter sweeping method
Naive, Seasonal Naive, Average, Seasonal Average	Time series	No sweeping within class due to model simplicity
Exponential Smoothing, ARIMA(X)	Time series	Grid search for within-class sweeping
Prophet	Regression	No sweeping within class
Linear SGD, LARS LASSO, Elastic Net, K Nearest Neighbors, Decision Tree, Random Forest, Extremely Randomized Trees, Gradient Boosted Trees, LightGBM, XGBoost	Regression	AutoML's model recommendation service ↗ dynamically explores hyper-parameter spaces

Model class group	Model type	Hyper-parameter sweeping method
ForecastTCN	Regression	Static list of models followed by random search over network size, dropout ratio, and learning rate.

For a description of the different model types, see the [forecasting models](#) section of the methods overview article.

The amount of sweeping that AutoML does depends on the forecasting job configuration. You can specify the stopping criteria as a time limit or a limit on the number of trials, or equivalently the number of models. Early termination logic can be used in both cases to stop sweeping if the primary metric is not improving.

Model selection

AutoML forecasting model search and selection proceeds in the following three phases:

1. Sweep over time series models and select the best model from *each class* using [penalized likelihood methods](#).
2. Sweep over regression models and rank them, along with the best time series models from phase 1, according to their primary metric values from validation sets.
3. Build an ensemble model from the top ranked models, calculate its validation metric, and rank it with the other models.

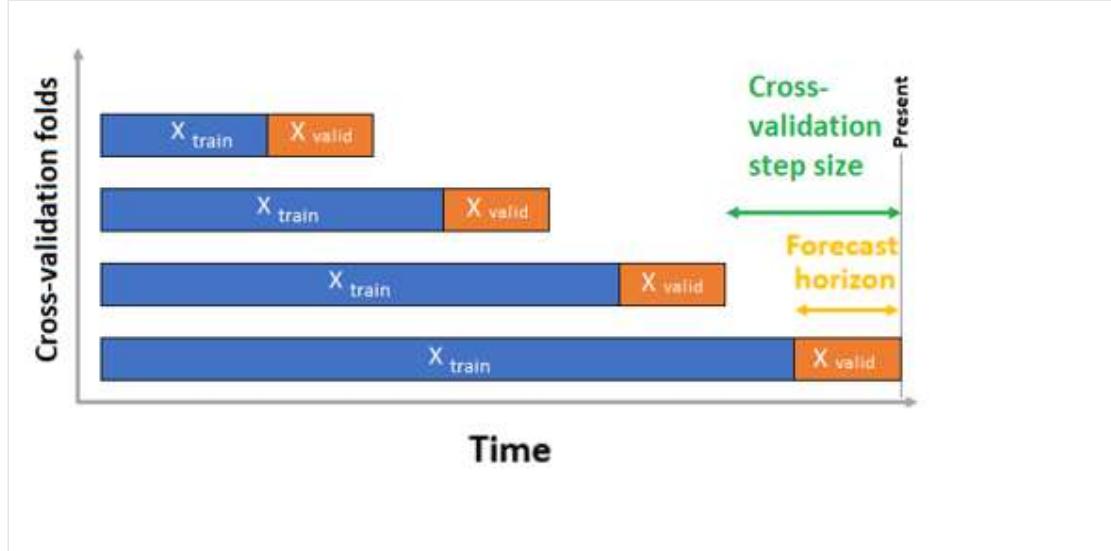
The model with the top ranked metric value at the end of phase 3 is designated the best model.

ⓘ Important

AutoML's final phase of model selection always calculates metrics on **out-of-sample** data. That is, data that was not used to fit the models. This helps to protect against over-fitting.

AutoML has two validation configurations - cross-validation and explicit validation data. In the cross-validation case, AutoML uses the input configuration to create data splits into training and validation folds. Time order must be preserved in these splits, so AutoML uses so-called **Rolling Origin Cross Validation** which divides the series into training and validation data using an origin time point. Sliding the origin in time

generates the cross-validation folds. Each validation fold contains the next horizon of observations immediately following the position of the origin for the given fold. This strategy preserves the time series data integrity and mitigates the risk of information leakage.



AutoML follows the usual cross-validation procedure, training a separate model on each fold and averaging validation metrics from all folds.

Cross-validation for forecasting jobs is configured by setting the number of cross-validation folds and, optionally, the number of time periods between two consecutive cross-validation folds. See the [custom cross-validation settings](#) guide for more information and an example of configuring cross-validation for forecasting.

You can also bring your own validation data. Learn more in the [configure data splits and cross-validation in AutoML](#) article.

Next steps

- Learn more about [how to set up AutoML to train a time-series forecasting model](#).
- Browse [AutoML Forecasting Frequently Asked Questions](#).
- Learn about [calendar features for time series forecasting in AutoML](#).
- Learn about [how AutoML uses machine learning to build forecasting models](#).

Additional resources

[Documentation](#)

[Build & train models \(v1\) - Azure Machine Learning](#)

Learn how to train models with Azure Machine Learning (v1). Explore the different training methods and choose the right one for your project.

[azure.ai.ml.operations.DatastoreOperations class](#)

Represents a client for performing operations on Datastores. You should not instantiate this class directly. Instead, you should create MLClient and use this client via the property MLClient.datastores

[Migrate from Estimators to ScriptRunConfig - Azure Machine Learning](#)

Migration guide for migrating from Estimators to ScriptRunConfig for configuring training jobs.

[azure.ai.ml.automl package](#)

Contains automated machine learning classes for Azure Machine Learning SDKv2. Main areas include managing AutoML tasks.

[azure.ai.ml.entities.Datastore class](#)

Datastore of an Azure ML workspace, abstract class.

[Upgrade steps for Azure Container Instances web services to managed online endpoints - Azure Machine Learning](#)

Upgrade steps for Azure Container Instances web services to managed online endpoints in Azure Machine Learning

[azure.ai.ml.entities.AzureBlobDatastore class](#)

Azure blob storage that is linked to an Azure ML workspace.

[azureml.widgets.RunDetails class - Azure Machine Learning Python](#)

Represents a Jupyter notebook widget used to view the progress of model training. A widget is asynchronous and provides updates until training finishes.

[Show 5 more](#)

Calendar features for time series forecasting in AutoML

Article • 01/18/2023 • 5 minutes to read

This article focuses on the calendar-based features that AutoML creates to increase the accuracy of forecasting regression models. Since holidays can have a strong influence on how the modeled system behaves, the time before, during, and after a holiday can bias the series' patterns. Each holiday generates a window over your existing dataset that the learner can assign an effect to. This can be especially useful in scenarios such as holidays that generate high demands for specific products. See the [methods overview article](#) for more general information about forecasting methodology in AutoML. Instructions and examples for training forecasting models in AutoML can be found in our [set up AutoML for time series forecasting](#) article.

As a part of feature engineering, AutoML transforms datetime type columns provided in the training data into new columns of calendar-based features. These features can help regression models learn seasonal patterns at several cadences. AutoML can always create calendar features from the time index of the time series since this is a required column in the training data. Calendar features are also made from other columns with datetime type, if any are present. See the [how AutoML uses your data](#) guide for more information on data requirements.

AutoML considers two categories of calendar features: standard features that are based entirely on date and time values and holiday features which are specific to a country or region of the world. We'll go over these features in the remainder of the article.

Standard calendar features

The following table shows the full set of AutoML's standard calendar features along with an example output. The example uses the standard `YY-mm-dd %H-%m-%d` format for datetime representation.

Feature	Description	Example output for 2011-01-01 00:25:30
<code>name</code>		
<code>year</code>	Numeric feature representing the calendar year	2011

Feature name	Description	Example output for 2011-01-01 00:25:30
<code>year_iso</code>	Represents ISO year as defined in ISO 8601. ISO years start on the first week of year that has a Thursday. For example, if January 1 is a Friday, the ISO year begins on January 4. ISO years may differ from calendar years.	2010
<code>half</code>	Feature indicating whether the date is in the first or second half of the year. It is 1 if the date is prior to July 1 and 2 otherwise.	
<code>quarter</code>	Numeric feature representing the quarter of the given date. It takes values 1, 2, 3, or 4 representing first, second, third, fourth quarter of calendar year.	1
<code>month</code>	Numeric feature representing the calendar month. It takes values 1 through 12.	1
<code>month_lbl</code>	String feature representing the name of month.	'January'
<code>day</code>	Numeric feature representing the day of the month. It takes values from 1 through 31.	1
<code>hour</code>	Numeric feature representing the hour of the day. It takes values 0 through 23.	0
<code>minute</code>	Numeric feature representing the minute within the hour. It takes values 0 through 59.	25
<code>second</code>	Numeric feature representing the second of the given datetime. In the case where only date format is provided, then it is assumed as 0. It takes values 0 through 59.	30
<code>am_pm</code>	Numeric feature indicating whether the time is in the morning or evening. It is 0 for times before 12PM and 1 for times after 12PM.	0
<code>am_pm_lbl</code>	String feature indicating whether the time is in the morning or evening.	'am'
<code>hour12</code>	Numeric feature representing the hour of the day on a 12 hour clock. It takes values 0 through 12 for first half of the day and 1 through 11 for second half.	0
<code>wday</code>	Numeric feature representing the day of the week. It takes values 0 through 6, where 0 corresponds to Monday.	5
<code>wday_lbl</code>	String feature representing name of the day of the week.	

Feature name	Description	Example output for 2011-01-01 00:25:30
qday	Numeric feature representing the day within the quarter. It takes values 1 through 92.	1
yday	Numeric feature representing the day of the year. It takes values 1 through 365, or 1 through 366 in the case of leap year.	1
week	Numeric feature representing ISO week as defined in ISO 8601. ISO weeks always start on Monday and end on Sunday. It takes values 1 through 52, or 53 for years having 1st January falling on Thursday or for leap years having 1st January falling on Wednesday.	52

The full set of standard calendar features may not be created in all cases. The generated set depends on the frequency of the time series and whether the training data contains datetime features in addition to the time index. The following table shows the features created for different column types:

Column purpose	Calendar features
Time index	The full set minus calendar features that have high correlation with other features. For example, if the time series frequency is daily, then any features with a more granular frequency than daily will be removed since they don't provide useful information.
Other datetime column	A reduced set consisting of Year, Month, Day, DayOfWeek, DayOfYear, QuarterOfYear, WeekOfMonth, Hour, Minute, and Second. If the column is a date with no time, Hour, Minute, and Second will be 0.

Holiday features

AutoML can optionally create features representing holidays from a specific country or region. These features are configured in AutoML using the `country_or_region_for_holidays` parameter which accepts an [ISO country code ↗](#).

ⓘ Note

Holiday features can only be made for time series with daily frequency.

The following table summarizes the holiday features:

Feature name	Description
Holiday	String feature that specifies whether a date is a regional or national holiday. Days within some range of a holiday are also marked.
isPaidTimeOff	Binary feature that takes value 1 if the day is a "paid time-off holiday" in the given country or region.

AutoML uses Azure Open Datasets as a source for holiday information. For more information, see the [PublicHolidays](#) documentation.

To better understand the holiday feature generation, consider the following example data:

timeStamp	Substation	demand	precip	temp
22/12/2013	NW	5150.570833	0.000995833	61.02708333
23/12/2013	NW	5721.995833	0.017008333	54.62708333
24/12/2013	NW	5578.833333	0	36.29958333
25/12/2013	NW	5325.7	0	24.91208333
26/12/2013	NW	5859.683333	0	32.88416667
27/12/2013	NW	5785.25	0	35.85666667
28/12/2013	NW	5276.3125	0	43.41375
29/12/2013	NW	5206.291667	0.032954167	43.35083333
30/12/2013	NW	5813.270833	0	36.80083333
31/12/2013	NW	5943.595833	0	26.91166667
01/01/2014	NW	5486.45	0	29.33625
02/01/2014	NW	6267.7375	0.005866667	27.0425
03/01/2014	NW	6345.95	0.006570833	14.15583333
04/01/2014	NW	5993.175	0	18.35041667

To make American holiday features for this data, we set the `country_or_region_for_holiday` to 'US' in the [forecast settings](#) as shown in the following code sample:

Python

```
from azure.ai.ml import automl

# create a forecasting job
forecasting_job = automl.forecasting(
    compute='test_cluster',    # Name of single or multinode AML compute
    infrastructure created by user
    experiment_name=exp_name, # name of experiment
    training_data=sample_data,
    target_column_name='demand',
    primary_metric='NormalizedRootMeanSquaredError',
    n_cross_validations=3,
    enable_model_explainability=True
)

# set custom forecast settings
forecasting_job.set_forecast_settings(
    time_column_name='timeStamp',
    country_or_region_for_holidays='US'
)
```

The generated holiday features look like the following:

timeStamp	Substation	precip	temp	_automl_Holiday	_automl_IsPaidTimeOff
22/12/2013	NW	0.000995833	61.02708333	3 days before Christmas Day	0
23/12/2013	NW	0.017008333	54.62708333	2 days before Christmas Day	0
24/12/2013	NW	0	36.29958333	1 day before Christmas Day	0
25/12/2013	NW	0	24.91208333	Christmas Day	1
26/12/2013	NW	0	32.88416667	1 day after Christmas Day	0
27/12/2013	NW	0	35.85666667	2 days after Christmas Day	0
28/12/2013	NW	0	43.41375	3 days after Christmas Day	0
29/12/2013	NW	0.032954167	43.35083333	4 days after Christmas Day	0
30/12/2013	NW	0	36.80083333	5 days after Christmas Day	0
31/12/2013	NW	0	26.91166667	6 days after Christmas Day	0
01/01/2014	NW	0	29.33625	New Year's Day	1
02/01/2014	NW	0.005866667	27.0425	1 day after New Year's Day	0
03/01/2014	NW	0.006570833	14.15583333	2 days after New Year's Day	0
04/01/2014	NW	0	18.35041667	3 days after New Year's Day	0

Note that generated features have the prefix `_automl_` prepended to their column names. AutoML generally uses this prefix to distinguish input features from engineered features.

Next steps

- Learn more about [how to set up AutoML to train a time-series forecasting model](#).
- Browse [AutoML Forecasting Frequently Asked Questions](#).
- Learn about [AutoML Forecasting Lagged Features](#).
- Learn about [how AutoML uses machine learning to build forecasting models](#).

Additional resources

Lagged features for time series forecasting in AutoML

Article • 01/18/2023 • 3 minutes to read

This article focuses on AutoML's methods for creating lag and rolling window aggregation features for forecasting regression models. Features like these that use past information can significantly increase accuracy by helping the model to learn correlational patterns in time. See the [methods overview article](#) for general information about forecasting methodology in AutoML. Instructions and examples for training forecasting models in AutoML can be found in our [set up AutoML for time series forecasting](#) article.

Lag feature example

AutoML generates lags with respect to the forecast horizon. The example in this section illustrates this concept. Here, we use a forecast horizon of three and target lag order of one. Consider the following monthly time series:

Table 1: Original time series

Date	y_t
1/1/2001	0
2/1/2001	10
3/1/2001	20
4/1/2001	30
5/1/2001	40
6/1/2001	50

First, we generate the lag feature for the horizon $h = 1$ only. As you continue reading, it will become clear why we use individual horizons in each table.

Table 2: Lag featurization for $h = 1$

Date	y_t	Origin	y_{t-1}	h
1/1/2001	0	12/1/2000	-	1

Date	y_t	Origin	y_{t-1}	h
2/1/2001	10	1/1/2001	0	1
3/1/2001	20	2/1/2001	10	1
4/1/2001	30	3/1/2001	20	1
5/1/2001	40	4/1/2001	30	1
6/1/2001	50	4/1/2001	40	1

Table 2 is generated from Table 1 by shifting the y_t column down by a single observation. We've added a column named `Origin` that has the dates that the lag features originate from. Next, we generate the lagging feature for the forecast horizon $h = 2$ only.

Table 3: Lag featurization for $h = 2$

Date	y_t	Origin	y_{t-2}	h
1/1/2001	0	11/1/2000	-	2
2/1/2001	10	12/1/2000	-	2
3/1/2001	20	1/1/2001	0	2
4/1/2001	30	2/1/2001	10	2
5/1/2001	40	3/1/2001	20	2
6/1/2001	50	4/1/2001	30	2

Table 3 is generated from Table 1 by shifting the y_t column down by two observations. Finally, we will generate the lagging feature for the forecast horizon $h = 3$ only.

Table 4: Lag featurization for $h = 3$

Date	y_t	Origin	y_{t-3}	h
1/1/2001	0	10/1/2000	-	3
2/1/2001	10	11/1/2000	-	3
3/1/2001	20	12/1/2000	-	3
4/1/2001	30	1/1/2001	0	3
5/1/2001	40	2/1/2001	10	3

Date	y_t	Origin	y_{t-3}	h
6/1/2001	50	3/1/2001	20	3

Next, we concatenate Tables 1, 2, and 3 and rearrange the rows. The result is in the following table:

Table 5: Lag featurization complete

Date	y_t	Origin	$y_{t-1}^{(h)}$	h
1/1/2001	0	12/1/2000	-	1
1/1/2001	0	11/1/2000	-	2
1/1/2001	0	10/1/2000	-	3
2/1/2001	10	1/1/2001	0	1
2/1/2001	10	12/1/2000	-	2
2/1/2001	10	11/1/2000	-	3
3/1/2001	20	2/1/2001	10	1
3/1/2001	20	1/1/2001	0	2
3/1/2001	20	12/1/2000	-	3
4/1/2001	30	3/1/2001	20	1
4/1/2001	30	2/1/2001	10	2
4/1/2001	30	1/1/2001	0	3
5/1/2001	40	4/1/2001	30	1
5/1/2001	40	3/1/2001	20	2
5/1/2001	40	2/1/2001	10	3
6/1/2001	50	4/1/2001	40	1
6/1/2001	50	4/1/2001	30	2
6/1/2001	50	3/1/2001	20	3

In the final table, we've changed the name of the lag column to $y_{t-1}^{(h)}$ to reflect that the lag is generated with respect to a specific horizon. The table shows that the lags we

generated with respect to the horizon can be mapped to the conventional ways of generating lags in the previous tables.

Table 5 is an example of the data augmentation that AutoML applies to training data to enable direct forecasting from regression models. When the configuration includes lag features, AutoML creates horizon dependent lags along with an integer-valued horizon feature. This enables AutoML's forecasting regression models to make a prediction at horizon h without regard to the prediction at $h - 1$, in contrast to recursively defined models like ARIMA.

ⓘ Note

Generation of horizon dependent lag features adds new *rows* to the dataset. The number of new rows is proportional to forecast horizon. This dataset size growth can lead to out-of-memory errors on smaller compute nodes or when dataset size is already large. See the [frequently asked questions](#) article for solutions to this problem.

Another consequence of this lagging strategy is that lag order and forecast horizon are decoupled. If, for example, your forecast horizon is seven, and you want AutoML to use lag features, you do not have to set the lag order to seven to ensure prediction over a full forecast horizon. Since AutoML generates lags with respect to horizon, you can set the lag order to one and AutoML will augment the data so that lags of any order are valid up to forecast horizon.

Next steps

- Learn more about [how to set up AutoML to train a time-series forecasting model](#).
- Browse [AutoML Forecasting Frequently Asked Questions](#).
- Learn about [calendar features for time series forecasting in AutoML](#).
- Learn about [how AutoML uses machine learning to build forecasting models](#).

Additional resources

Prevent overfitting and imbalanced data with automated machine learning

Article • 10/20/2022 • 7 minutes to read

Overfitting and imbalanced data are common pitfalls when you build machine learning models. By default, Azure Machine Learning's automated machine learning provides charts and metrics to help you identify these risks, and implements best practices to help mitigate them.

Identify overfitting

Overfitting in machine learning occurs when a model fits the training data too well, and as a result can't accurately predict on unseen test data. In other words, the model has simply memorized specific patterns and noise in the training data, but is not flexible enough to make predictions on real data.

Consider the following trained models and their corresponding train and test accuracies.

Model	Train accuracy	Test accuracy
A	99.9%	95%
B	87%	87%
C	99.9%	45%

Considering model **A**, there is a common misconception that if test accuracy on unseen data is lower than training accuracy, the model is overfitted. However, test accuracy should always be less than training accuracy, and the distinction for overfit vs. appropriately fit comes down to *how much* less accurate.

When comparing models **A** and **B**, model **A** is a better model because it has higher test accuracy, and although the test accuracy is slightly lower at 95%, it is not a significant difference that suggests overfitting is present. You wouldn't choose model **B** simply because the train and test accuracies are closer together.

Model **C** represents a clear case of overfitting; the training accuracy is very high but the test accuracy isn't anywhere near as high. This distinction is subjective, but comes from knowledge of your problem and data, and what magnitudes of error are acceptable.

Prevent overfitting

In the most egregious cases, an overfitted model assumes that the feature value combinations seen during training will always result in the exact same output for the target.

The best way to prevent overfitting is to follow ML best-practices including:

- Using more training data, and eliminating statistical bias
- Preventing target leakage
- Using fewer features
- **Regularization and hyperparameter optimization**
- **Model complexity limitations**
- **Cross-validation**

In the context of automated ML, the first three items above are **best-practices you implement**. The last three bolded items are **best-practices automated ML implements** by default to protect against overfitting. In settings other than automated ML, all six best-practices are worth following to avoid overfitting models.

Best practices you implement

Use more data

Using **more data** is the simplest and best possible way to prevent overfitting, and as an added bonus typically increases accuracy. When you use more data, it becomes harder for the model to memorize exact patterns, and it is forced to reach solutions that are more flexible to accommodate more conditions. It's also important to recognize **statistical bias**, to ensure your training data doesn't include isolated patterns that won't exist in live-prediction data. This scenario can be difficult to solve, because there may not be overfitting between your train and test sets, but there may be overfitting present when compared to live test data.

Prevent target leakage

Target leakage is a similar issue, where you may not see overfitting between train/test sets, but rather it appears at prediction-time. Target leakage occurs when your model "cheats" during training by having access to data that it shouldn't normally have at prediction-time. For example, if your problem is to predict on Monday what a commodity price will be on Friday, but one of your features accidentally included data from Thursdays, that would be data the model won't have at prediction-time since it cannot see into the future. Target leakage is an easy mistake to miss, but is often characterized by abnormally high accuracy for your problem. If you are attempting to

predict stock price and trained a model at 95% accuracy, there is likely target leakage somewhere in your features.

Use fewer features

Removing features can also help with overfitting by preventing the model from having too many fields to use to memorize specific patterns, thus causing it to be more flexible. It can be difficult to measure quantitatively, but if you can remove features and retain the same accuracy, you have likely made the model more flexible and have reduced the risk of overfitting.

Best practices automated ML implements

Regularization and hyperparameter tuning

Regularization is the process of minimizing a cost function to penalize complex and overfitted models. There are different types of regularization functions, but in general they all penalize model coefficient size, variance, and complexity. Automated ML uses L1 (Lasso), L2 (Ridge), and ElasticNet (L1 and L2 simultaneously) in different combinations with different model hyperparameter settings that control overfitting. In simple terms, automated ML will vary how much a model is regulated and choose the best result.

Model complexity limitations

Automated ML also implements explicit **model complexity limitations** to prevent overfitting. In most cases this implementation is specifically for decision tree or forest algorithms, where individual tree max-depth is limited, and the total number of trees used in forest or ensemble techniques are limited.

Cross-validation

Cross-validation (CV) is the process of taking many subsets of your full training data and training a model on each subset. The idea is that a model could get "lucky" and have great accuracy with one subset, but by using many subsets the model won't achieve this high accuracy every time. When doing CV, you provide a validation holdout dataset, specify your CV folds (number of subsets) and automated ML will train your model and tune hyperparameters to minimize error on your validation set. One CV fold could be overfitted, but by using many of them it reduces the probability that your final model is overfitted. The tradeoff is that CV does result in longer training times and thus

greater cost, because instead of training a model once, you train it once for each n CV subsets.

ⓘ Note

Cross-validation is not enabled by default; it must be configured in automated ML settings. However, after cross-validation is configured and a validation data set has been provided, the process is automated for you. Learn more about [cross-validation configuration in Auto ML](#)

Identify models with imbalanced data

Imbalanced data is commonly found in data for machine learning classification scenarios, and refers to data that contains a disproportionate ratio of observations in each class. This imbalance can lead to a falsely perceived positive effect of a model's accuracy, because the input data has bias towards one class, which results in the trained model to mimic that bias.

In addition, automated ML jobs generate the following charts automatically, which can help you understand the correctness of the classifications of your model, and identify models potentially impacted by imbalanced data.

Chart	Description
Confusion Matrix	Evaluates the correctly classified labels against the actual labels of the data.
Precision-recall	Evaluates the ratio of correct labels against the ratio of found label instances of the data
ROC Curves	Evaluates the ratio of correct labels against the ratio of false-positive labels.

Handle imbalanced data

As part of its goal of simplifying the machine learning workflow, **automated ML has built in capabilities** to help deal with imbalanced data such as,

- A **weight column**: automated ML supports a column of weights as input, causing rows in the data to be weighted up or down, which can be used to make a class more or less "important".

- The algorithms used by automated ML detect imbalance when the number of samples in the minority class is equal to or fewer than 20% of the number of samples in the majority class, where minority class refers to the one with fewest samples and majority class refers to the one with most samples. Subsequently, AutoML will run an experiment with sub-sampled data to check if using class weights would remedy this problem and improve performance. If it ascertains a better performance through this experiment, then this remedy is applied.
- Use a performance metric that deals better with imbalanced data. For example, the AUC_weighted is a primary metric that calculates the contribution of every class based on the relative number of samples representing that class, hence is more robust against imbalance.

The following techniques are additional options to handle imbalanced data **outside of automated ML**.

- Resampling to even the class imbalance, either by up-sampling the smaller classes or down-sampling the larger classes. These methods require expertise to process and analyze.
- Review performance metrics for imbalanced data. For example, the F1 score is the harmonic mean of precision and recall. Precision measures a classifier's exactness, where higher precision indicates fewer false positives, while recall measures a classifier's completeness, where higher recall indicates fewer false negatives.

Next steps

See examples and learn how to build models using automated machine learning:

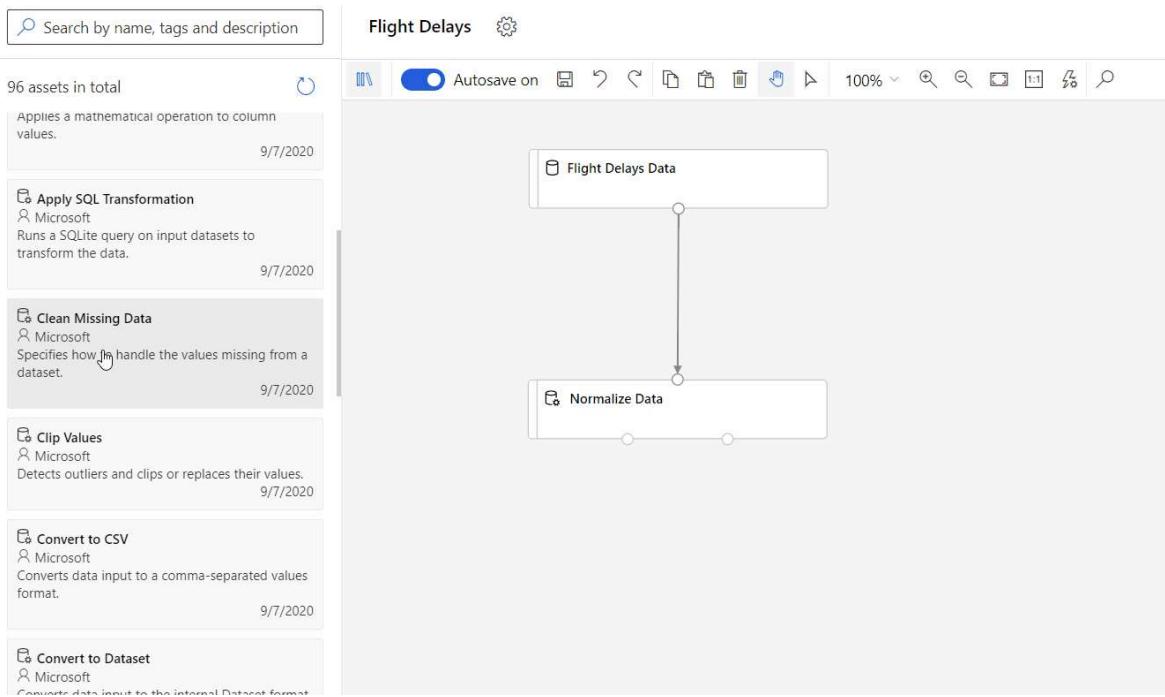
- Follow the [Tutorial: Train an object detection model with AutoML and Python](#).
- Configure the settings for automatic training experiment:
 - In Azure Machine Learning studio, [use these steps](#).
 - With the Python SDK, [use these steps](#).

What is Azure Machine Learning designer?

Article • 01/20/2023 • 4 minutes to read

Azure Machine Learning designer is a drag-and-drop interface used to train and deploy models in Azure Machine Learning. This article describes the tasks you can do in the designer.

- To get started with the designer, see [Tutorial: Train a no-code regression model](#).
- To learn about the components available in the designer, see the [Algorithm and component reference](#).



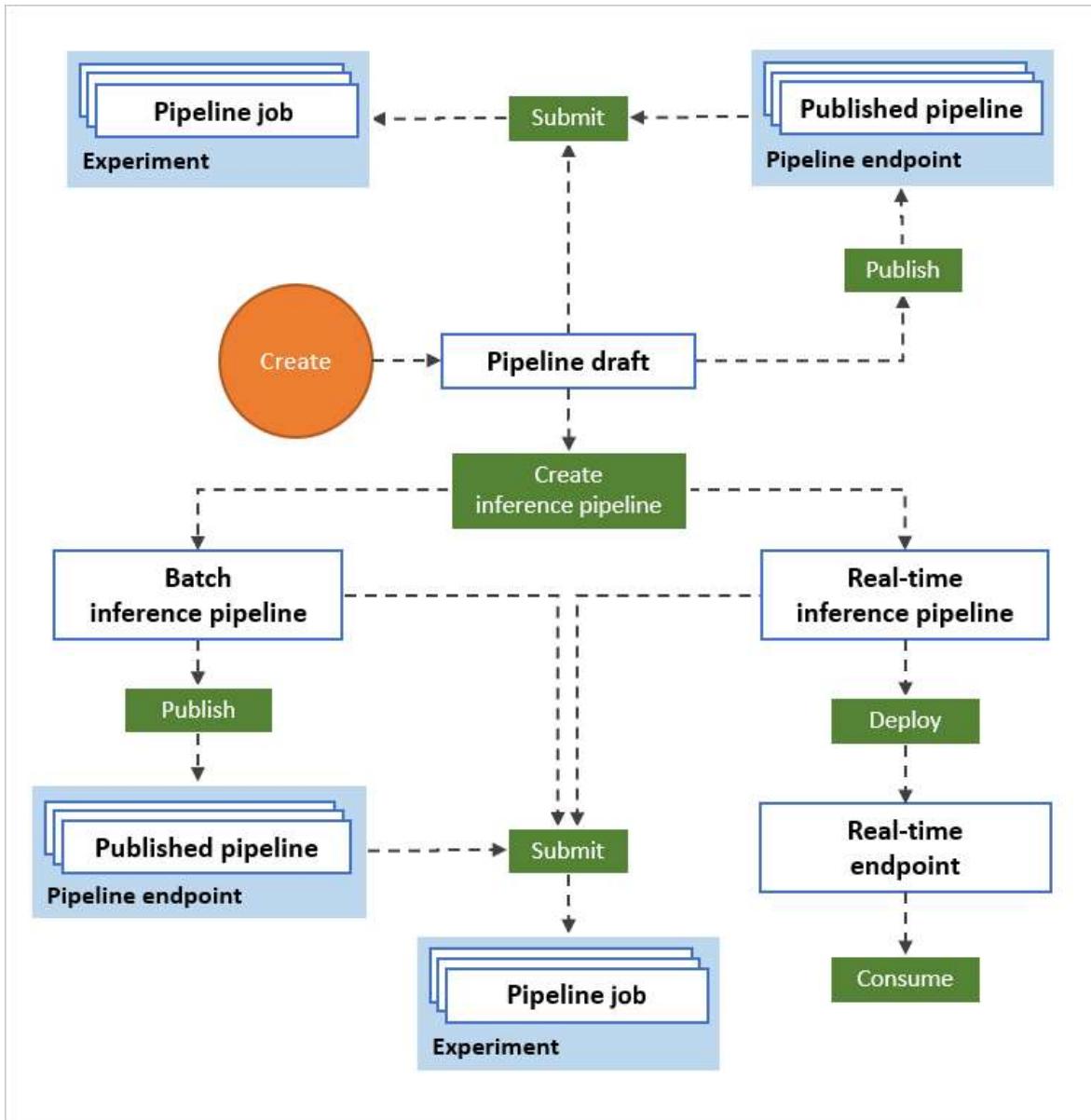
The designer uses your Azure Machine Learning [workspace](#) to organize shared resources such as:

- [Pipelines](#)
- [Data](#)
- [Compute resources](#)
- [Registered models](#)
- [Published pipelines](#)
- [Real-time endpoints](#)

Model training and deployment

Use a visual canvas to build an end-to-end machine learning workflow. Train, test, and deploy models all in the designer:

- Drag-and-drop **data assets** and **components** onto the canvas.
- Connect the components to create a **pipeline draft**.
- Submit a **pipeline run** using the compute resources in your Azure Machine Learning workspace.
- Convert your **training pipelines** to **inference pipelines**.
- **Publish** your pipelines to a REST **pipeline endpoint** to submit a new pipeline that runs with different parameters and data assets.
 - Publish a **training pipeline** to reuse a single pipeline to train multiple models while changing parameters and data assets.
 - Publish a **batch inference pipeline** to make predictions on new data by using a previously trained model.
- **Deploy** a **real-time inference pipeline** to an online endpoint to make predictions on new data in real time.



Pipeline

A [pipeline](#) consists of data assets and analytical components, which you connect. Pipelines have many uses: you can make a pipeline that trains a single model, or one that trains multiple models. You can create a pipeline that makes predictions in real time or in batch, or make a pipeline that only cleans data. Pipelines let you reuse your work and organize your projects.

Pipeline draft

As you edit a pipeline in the designer, your progress is saved as a [pipeline draft](#). You can edit a pipeline draft at any point by adding or removing components, configuring compute targets, creating parameters, and so on.

A valid pipeline has these characteristics:

- Data assets can only connect to components.
- Components can only connect to either data assets or other components.
- All input ports for components must have some connection to the data flow.
- All required parameters for each component must be set.

When you're ready to run your pipeline draft, you submit a pipeline job.

Pipeline job

Each time you run a pipeline, the configuration of the pipeline and its results are stored in your workspace as a **pipeline job**. You can go back to any pipeline job to inspect it for troubleshooting or auditing. **Clone** a pipeline job to create a new pipeline draft for you to edit.

Pipeline jobs are grouped into [experiments](#) to organize job history. You can set the experiment for every pipeline job.

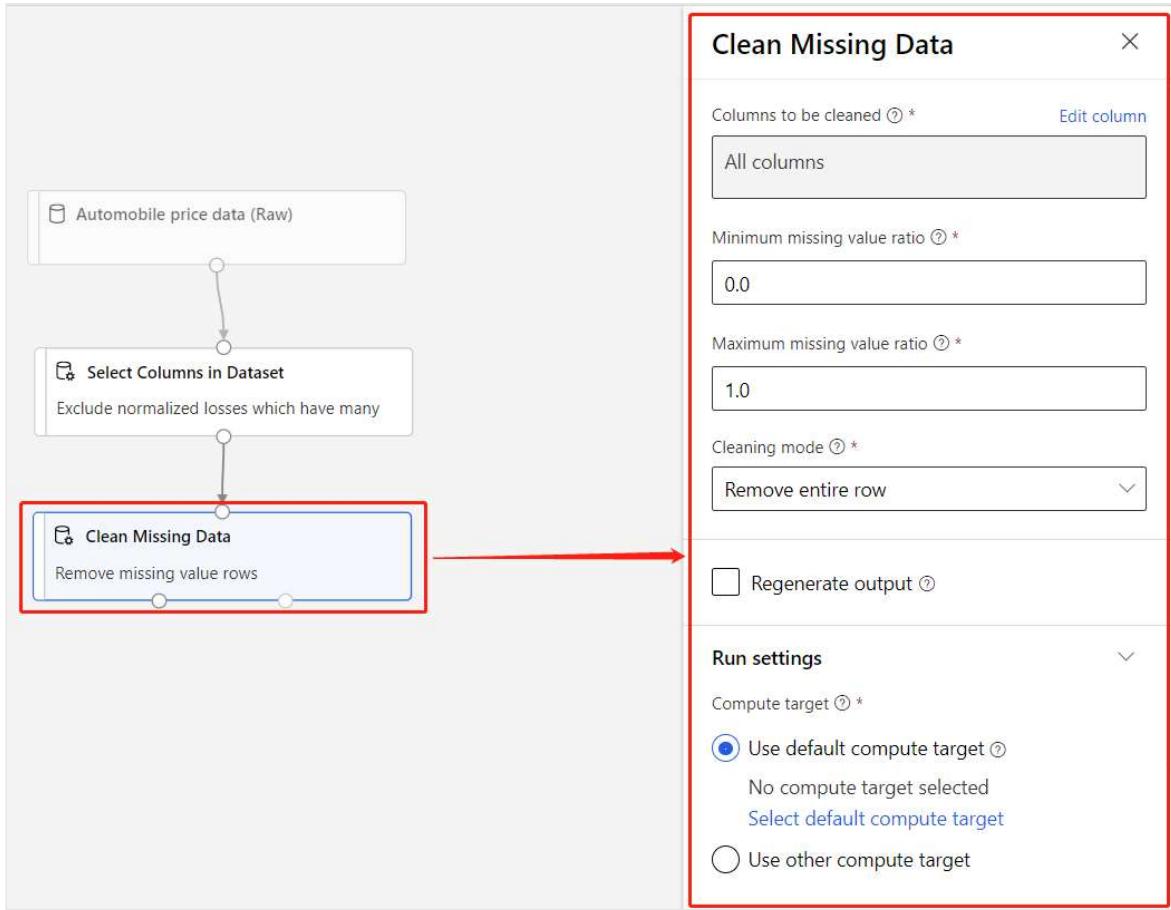
Data

A machine learning data asset makes it easy to access and work with your data. Several [sample data assets](#) are included in the designer for you to experiment with. You can [register](#) more data assets as you need them.

Component

A component is an algorithm that you can perform on your data. The designer has several components ranging from data ingress functions to training, scoring, and validation processes.

A component may have a set of parameters that you can use to configure the component's internal algorithms. When you select a component on the canvas, the component's parameters are displayed in the Properties pane to the right of the canvas. You can modify the parameters in that pane to tune your model. You can set the compute resources for individual components in the designer.



For some help navigating through the library of machine learning algorithms available, see [Algorithm & component reference overview](#). For help with choosing an algorithm, see the [Azure Machine Learning Algorithm Cheat Sheet](#).

Compute resources

Use compute resources from your workspace to run your pipeline and host your deployed models as online endpoints or pipeline endpoints (for batch inference). The supported compute targets are:

Compute target	Training	Deployment
Azure Machine Learning compute	✓	
Azure Kubernetes Service		✓

Compute targets are attached to your [Azure Machine Learning workspace](#). You manage your compute targets in your workspace in the [Azure Machine Learning studio](#).

Deploy

To perform real-time inferencing, you must deploy a pipeline as an [online endpoint](#). The online endpoint creates an interface between an external application and your scoring model. A call to an online endpoint returns prediction results to the application in real time. To make a call to an online endpoint, you pass the API key that was created when you deployed the endpoint. The endpoint is based on REST, a popular architecture choice for web programming projects.

Online endpoints must be deployed to an Azure Kubernetes Service cluster.

To learn how to deploy your model, see [Tutorial: Deploy a machine learning model with the designer](#).

ⓘ Note

Azure Machine Learning Endpoints (preview) provide an improved, simpler deployment experience. Endpoints support both real-time and batch inference scenarios. Endpoints provide a unified interface to invoke and manage model deployments across compute types. See [What are Azure Machine Learning endpoints \(preview\)?](#).

Publish

You can also publish a pipeline to a [pipeline endpoint](#). Similar to an online endpoint, a pipeline endpoint lets you submit new pipeline jobs from external applications using REST calls. However, you cannot send or receive data in real time using a pipeline endpoint.

Published pipelines are flexible, they can be used to train or retrain models, [perform batch inferencing](#), process new data, and much more. You can publish multiple pipelines to a single pipeline endpoint and specify which pipeline version to run.

A published pipeline runs on the compute resources you define in the pipeline draft for each component.

The designer creates the same [PublishedPipeline](#) object as the SDK.

Next steps

- Learn the fundamentals of predictive analytics and machine learning with [Tutorial: Predict automobile price with the designer](#)
- Learn how to modify existing [designer samples](#) to adapt them to your needs.

Additional resources

Documentation

[Train Model: Component Reference - Azure Machine Learning](#)

Learn how to use the **Train Model** component in Azure Machine Learning to train a classification or regression model.

[Linear Regression: Component Reference - Azure Machine Learning](#)

Learn how to use the Linear Regression component in Azure Machine Learning to create a linear regression model for use in a pipeline.

[Two-Class Decision Forest: Component Reference - Azure Machine Learning](#)

Learn how to use the Two-Class Decision Forest component in Azure Machine Learning to create a machine learning model based on the decision forests algorithm.

[Tutorial: Designer - train a no-code regression model - Azure Machine Learning](#)

Train a regression model that predicts car prices using the Azure Machine Learning designer.

[Algorithm & component reference - Azure Machine Learning](#)

Learn about the Azure Machine Learning designer components that you can use to create your own machine learning projects.

[Evaluate Model: Component Reference - Azure Machine Learning](#)

Learn how to use the Evaluate Model component in Azure Machine Learning to measure the accuracy of a trained model.

[Select Columns in Dataset: Component Reference - Azure Machine Learning](#)

Learn how to use the Select Columns in Dataset component in Azure Machine Learning to choose a subset of columns to use in downstream operations.

[Machine Learning Algorithm Cheat Sheet - designer - Azure Machine Learning](#)

A printable Machine Learning Algorithm Cheat Sheet helps you choose the right algorithm for your predictive model in Azure Machine Learning designer.

[Show 5 more](#)

Training

Learning paths and modules

[Orchestrate machine learning with pipelines - Training](#)

Orchestrate machine learning with pipelines

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer

Article • 01/20/2023 • 3 minutes to read

The **Azure Machine Learning Algorithm Cheat Sheet** helps you choose the right algorithm from the designer for a predictive analytics model.

ⓘ Note

Designer supports two type of components, classic prebuilt components and custom components. These two types of components are not compatible.

Classic prebuilt components provides prebuilt components majorly for data processing and traditional machine learning tasks like regression and classification. This type of component continues to be supported but will not have any new components added.

Custom components allow you to provide your own code as a component. It supports sharing across workspaces and seamless authoring across Studio, CLI, and SDK interfaces.

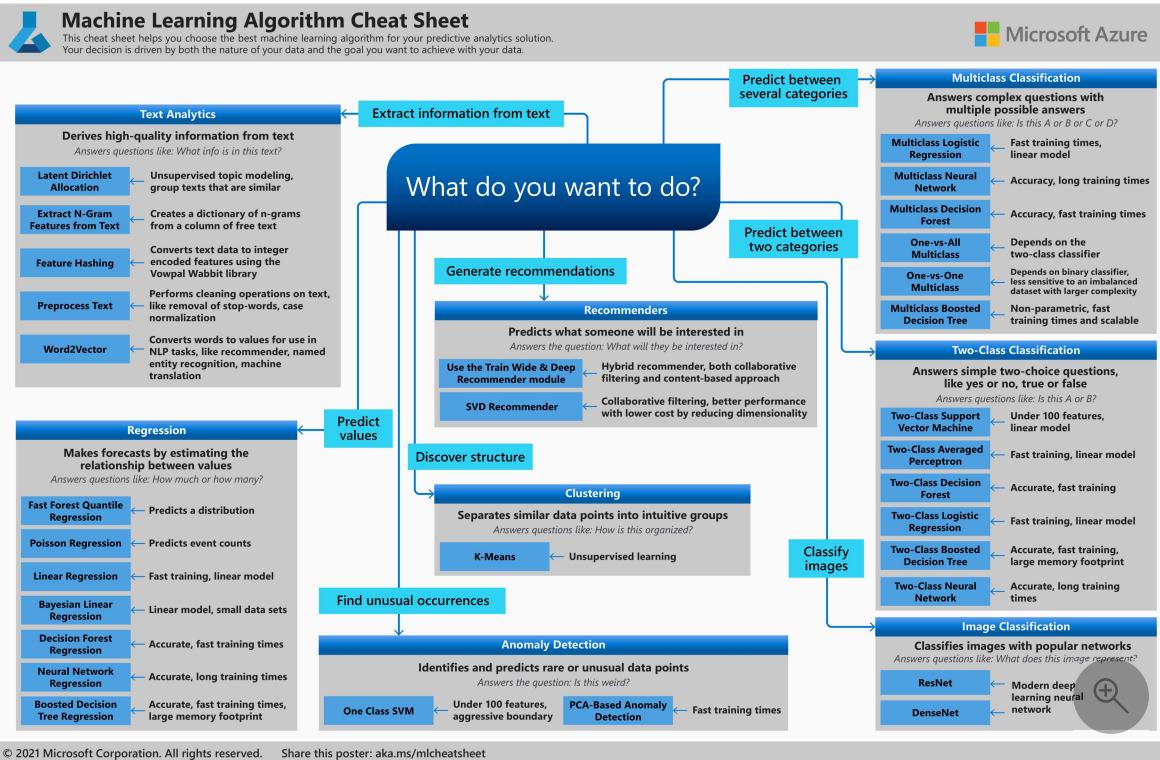
This article applies to classic prebuilt components.

Azure Machine Learning has a large library of algorithms from the *classification*, *recommender systems*, *clustering*, *anomaly detection*, *regression*, and *text analytics* families. Each is designed to address a different type of machine learning problem.

For more information, see [How to select algorithms](#).

Download: Machine Learning Algorithm Cheat Sheet

Download the cheat sheet here: [Machine Learning Algorithm Cheat Sheet \(11x17 in.\)](#)



© 2021 Microsoft Corporation. All rights reserved. Share this poster: aka.ms/mlcheatsheet

Download and print the Machine Learning Algorithm Cheat Sheet in tabloid size to keep it handy and get help choosing an algorithm.

How to use the Machine Learning Algorithm Cheat Sheet

The suggestions offered in this algorithm cheat sheet are approximate rules-of-thumb. Some can be bent, and some can be flagrantly violated. This cheat sheet is intended to suggest a starting point. Don't be afraid to run a head-to-head competition between several algorithms on your data. There is simply no substitute for understanding the principles of each algorithm and the system that generated your data.

Every machine learning algorithm has its own style or inductive bias. For a specific problem, several algorithms may be appropriate, and one algorithm may be a better fit than others. But it's not always possible to know beforehand, which is the best fit. In cases like these, several algorithms are listed together in the cheat sheet. An appropriate strategy would be to try one algorithm, and if the results are not yet satisfactory, try the others.

To learn more about the algorithms in Azure Machine Learning designer, go to the [Algorithm and component reference](#).

Kinds of machine learning

There are three main categories of machine learning: *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised learning

In supervised learning, each data point is labeled or associated with a category or value of interest. An example of a categorical label is assigning an image as either a 'cat' or a 'dog'. An example of a value label is the sale price associated with a used car. The goal of supervised learning is to study many labeled examples like these, and then to be able to make predictions about future data points. For example, identifying new photos with the correct animal or assigning accurate sale prices to other used cars. This is a popular and useful type of machine learning.

Unsupervised learning

In unsupervised learning, data points have no labels associated with them. Instead, the goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure. Unsupervised learning groups data into clusters, as K-means does, or finds different ways of looking at complex data so that it appears simpler.

Reinforcement learning

In reinforcement learning, the algorithm gets to choose an action in response to each data point. It is a common approach in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It's also a natural fit for Internet of Things applications. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was. Based on this signal, the algorithm modifies its strategy in order to achieve the highest reward.

Next steps

- See more information on [How to select algorithms](#)
- [Learn about studio in Azure Machine Learning and the Azure portal.](#)
- [Tutorial: Build a prediction model in Azure Machine Learning designer.](#)
- [Learn about deep learning vs. machine learning.](#)

Additional resources

Documentation

[How to select a machine learning algorithm - Azure Machine Learning](#)

How to select Azure Machine Learning algorithms for supervised and unsupervised learning in clustering, classification, or regression experiments.

[Linear Regression: Component Reference - Azure Machine Learning](#)

Learn how to use the Linear Regression component in Azure Machine Learning to create a linear regression model for use in a pipeline.

[K-Means Clustering: Component Reference - Azure Machine Learning](#)

Learn how to use the K-Means Clustering component in the Azure Machine Learning to train clustering models.

[Two-Class Decision Forest: Component Reference - Azure Machine Learning](#)

Learn how to use the Two-Class Decision Forest component in Azure Machine Learning to create a machine learning model based on the decision forests algorithm.

[What is the Azure Machine Learning designer? - Azure Machine Learning](#)

Learn what the Azure Machine Learning designer is and what tasks you can use it for. The drag-and-drop UI enables model training and deployment.

[Train Model: Component Reference - Azure Machine Learning](#)

Learn how to use the **Train Model** component in Azure Machine Learning to train a classification or regression model.

[Evaluate Model: Component Reference - Azure Machine Learning](#)

Learn how to use the Evaluate Model component in Azure Machine Learning to measure the accuracy of a trained model.

[Decision Forest Regression: Component Reference - Azure Machine Learning](#)

Learn how to use the Decision Forest Regression component in Azure Machine Learning to create a regression model based on an ensemble of decision trees.

[Show 5 more](#)

Training

Learning paths and modules

[Microsoft Azure AI Fundamentals: Explore visual tools for machine learning - Training](#)

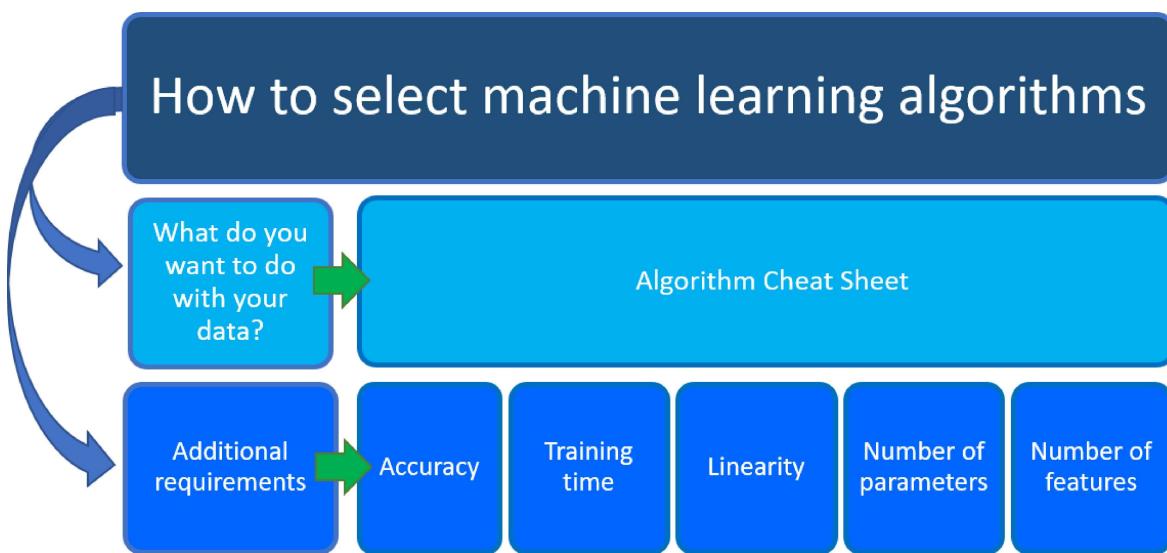
Microsoft Azure AI Fundamentals: Explore visual tools for machine learning

How to select algorithms for Azure Machine Learning

Article • 01/04/2023 • 7 minutes to read

A common question is “Which machine learning algorithm should I use?” The algorithm you select depends primarily on two different aspects of your data science scenario:

- **What you want to do with your data?** Specifically, what is the business question you want to answer by learning from your past data?
- **What are the requirements of your data science scenario?** Specifically, what is the accuracy, training time, linearity, number of parameters, and number of features your solution supports?



Business scenarios and the Machine Learning Algorithm Cheat Sheet

The [Azure Machine Learning Algorithm Cheat Sheet](#) helps you with the first consideration: **What you want to do with your data?** On the Machine Learning Algorithm Cheat Sheet, look for task you want to do, and then find a [Azure Machine Learning designer](#) algorithm for the predictive analytics solution.

Machine Learning designer provides a comprehensive portfolio of algorithms, such as [Multiclass Decision Forest](#), [Recommendation systems](#), [Neural Network Regression](#), [Multiclass Neural Network](#), and [K-Means Clustering](#). Each algorithm is designed to address a different type of machine learning problem. See the [Machine Learning designer algorithm and component reference](#) for a complete list along with

documentation about how each algorithm works and how to tune parameters to optimize the algorithm.

ⓘ Note

Download the cheat sheet here: [Machine Learning Algorithm Cheat Sheet \(11x17 in.\)](#)

Along with guidance in the Azure Machine Learning Algorithm Cheat Sheet, keep in mind other requirements when choosing a machine learning algorithm for your solution. Following are additional factors to consider, such as the accuracy, training time, linearity, number of parameters and number of features.

Comparison of machine learning algorithms

ⓘ Note

Designer supports two type of components, classic prebuilt components and custom components. These two types of components are not compatible.

Classic prebuilt components provides prebuilt components majorly for data processing and traditional machine learning tasks like regression and classification. This type of component continues to be supported but will not have any new components added.

Custom components allow you to provide your own code as a component. It supports sharing across workspaces and seamless authoring across Studio, CLI, and SDK interfaces.

This article applies to classic prebuilt components.

Some learning algorithms make particular assumptions about the structure of the data or the desired results. If you can find one that fits your needs, it can give you more useful results, more accurate predictions, or faster training times.

The following table summarizes some of the most important characteristics of algorithms from the classification, regression, and clustering families:

Algorithm	Accuracy	Training time	Linearity	Parameters	Notes
-----------	----------	---------------	-----------	------------	-------

Algorithm	Accuracy	Training time	Linearity	Parameters	Notes
Classification family					
Two-Class logistic regression	Good	Fast	Yes	4	
Two-class decision forest	Excellent	Moderate	No	5	Shows slower scoring times. Suggest not working with One-vs-All Multiclass, because of slower scoring times caused by thread locking in accumulating tree predictions
Two-class boosted decision tree	Excellent	Moderate	No	6	Large memory footprint
Two-class neural network	Good	Moderate	No	8	
Two-class averaged perceptron	Good	Moderate	Yes	4	
Two-class support vector machine	Good	Fast	Yes	5	Good for large feature sets
Multiclass logistic regression	Good	Fast	Yes	4	
Multiclass decision forest	Excellent	Moderate	No	5	Shows slower scoring times
Multiclass boosted decision tree	Excellent	Moderate	No	6	Tends to improve accuracy with some small risk of less coverage
Multiclass neural network	Good	Moderate	No	8	

Algorithm	Accuracy	Training time	Linearity	Parameters	Notes
One-vs-all multiclass	-	-	-	-	See properties of the two-class method selected
Regression family					
Linear regression	Good	Fast	Yes	4	
Decision forest regression	Excellent	Moderate	No	5	
Boosted decision tree regression	Excellent	Moderate	No	6	Large memory footprint
Neural network regression	Good	Moderate	No	8	
Clustering family					
K-means clustering	Excellent	Moderate	Yes	8	A clustering algorithm

Requirements for a data science scenario

Once you know what you want to do with your data, you need to determine additional requirements for your solution.

Make choices and possibly trade-offs for the following requirements:

- Accuracy
- Training time
- Linearity
- Number of parameters
- Number of features

Accuracy

Accuracy in machine learning measures the effectiveness of a model as the proportion of true results to total cases. In Machine Learning designer, the [Evaluate Model component](#) computes a set of industry-standard evaluation metrics. You can use this component to measure the accuracy of a trained model.

Getting the most accurate answer possible isn't always necessary. Sometimes an approximation is adequate, depending on what you want to use it for. If that is the case, you may be able to cut your processing time dramatically by sticking with more approximate methods. Approximate methods also naturally tend to avoid overfitting.

There are three ways to use the Evaluate Model component:

- Generate scores over your training data in order to evaluate the model
- Generate scores on the model, but compare those scores to scores on a reserved testing set
- Compare scores for two different but related models, using the same set of data

For a complete list of metrics and approaches you can use to evaluate the accuracy of machine learning models, see [Evaluate Model component](#).

Training time

In supervised learning, training means using historical data to build a machine learning model that minimizes errors. The number of minutes or hours necessary to train a model varies a great deal between algorithms. Training time is often closely tied to accuracy; one typically accompanies the other.

In addition, some algorithms are more sensitive to the number of data points than others. You might choose a specific algorithm because you have a time limitation, especially when the data set is large.

In Machine Learning designer, creating and using a machine learning model is typically a three-step process:

1. Configure a model, by choosing a particular type of algorithm, and then defining its parameters or hyperparameters.
2. Provide a dataset that is labeled and has data compatible with the algorithm.
Connect both the data and the model to [Train Model component](#).
3. After training is completed, use the trained model with one of the [scoring components](#) to make predictions on new data.

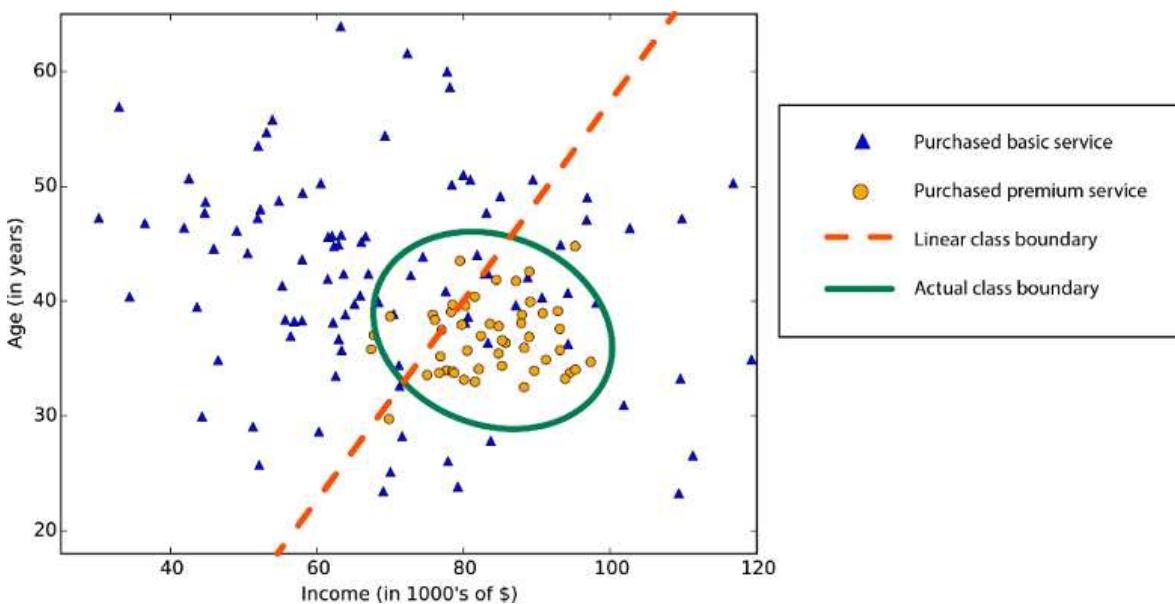
Linearity

Linearity in statistics and machine learning means that there is a linear relationship between a variable and a constant in your dataset. For example, linear classification algorithms assume that classes can be separated by a straight line (or its higher-dimensional analog).

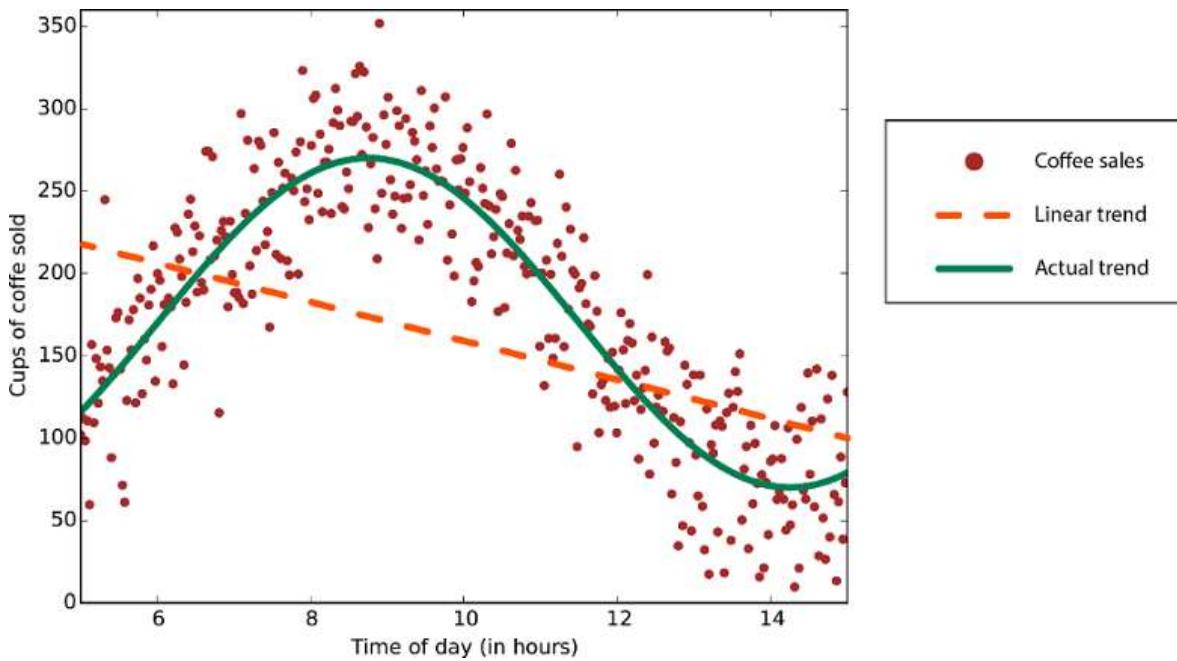
Lots of machine learning algorithms make use of linearity. In Azure Machine Learning designer, they include:

- [Multiclass logistic regression](#)
- [Two-class logistic regression](#)
- [Support vector machines](#)

Linear regression algorithms assume that data trends follow a straight line. This assumption isn't bad for some problems, but for others it reduces accuracy. Despite their drawbacks, linear algorithms are popular as a first strategy. They tend to be algorithmically simple and fast to train.



Nonlinear class boundary: Relying on a linear classification algorithm would result in low accuracy.



Data with a nonlinear trend: Using a linear regression method would generate much larger errors than necessary.

Number of parameters

Parameters are the knobs a data scientist gets to turn when setting up an algorithm. They are numbers that affect the algorithm's behavior, such as error tolerance or number of iterations, or options between variants of how the algorithm behaves. The training time and accuracy of the algorithm can sometimes be sensitive to getting just the right settings. Typically, algorithms with large numbers of parameters require the most trial and error to find a good combination.

Alternatively, there is the [Tune Model Hyperparameters component](#) in Machine Learning designer: The goal of this component is to determine the optimum hyperparameters for a machine learning model. The component builds and tests multiple models by using different combinations of settings. It compares metrics over all models to get the combinations of settings.

While this is a great way to make sure you've spanned the parameter space, the time required to train a model increases exponentially with the number of parameters. The upside is that having many parameters typically indicates that an algorithm has greater flexibility. It can often achieve very good accuracy, provided you can find the right combination of parameter settings.

Number of features

In machine learning, a feature is a quantifiable variable of the phenomenon you are trying to analyze. For certain types of data, the number of features can be very large compared to the number of data points. This is often the case with genetics or textual data.

A large number of features can bog down some learning algorithms, making training time unfeasibly long. [Support vector machines](#) are particularly well suited to scenarios with a high number of features. For this reason, they have been used in many applications from information retrieval to text and image classification. Support vector machines can be used for both classification and regression tasks.

Feature selection refers to the process of applying statistical tests to inputs, given a specified output. The goal is to determine which columns are more predictive of the output. The [Filter Based Feature Selection component](#) in Machine Learning designer provides multiple feature selection algorithms to choose from. The component includes correlation methods such as Pearson correlation and chi-squared values.

You can also use the [Permutation Feature Importance component](#) to compute a set of feature importance scores for your dataset. You can then leverage these scores to help you determine the best features to use in a model.

Next steps

- [Learn more about Azure Machine Learning designer](#)
- For descriptions of all the machine learning algorithms available in Azure Machine Learning designer, see [Machine Learning designer algorithm and component reference](#)
- To explore the relationship between deep learning, machine learning, and AI, see [Deep Learning vs. Machine Learning](#)

What are Azure Machine Learning endpoints?

Article • 02/24/2023 • 9 minutes to read

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) ↗

Use Azure Machine Learning endpoints to streamline model deployments for both real-time and batch inference deployments. Endpoints provide a unified interface to invoke and manage model deployments across compute types.

In this article, you learn about:

-  Endpoints
-  Deployments
-  Managed online endpoints
-  Kubernetes online endpoints
-  Batch inference endpoints

What are endpoints and deployments?

After you train a machine learning model, you need to deploy the model so that others can use it to do inferencing. In Azure Machine Learning, you can use **endpoints** and **deployments** to do so.

An **endpoint**, in this context, is an HTTPS path that provides an interface for clients to send requests (input data) and receive the inferencing (scoring) output of a trained model. An endpoint provides:

- Authentication using "key & token" based auth
- SSL termination
- A stable scoring URI (endpoint-name.region.inference.ml.azure.com)

A **deployment** is a set of resources required for hosting the model that does the actual inferencing.

A single endpoint can contain multiple deployments. Endpoints and deployments are independent Azure Resource Manager resources that appear in the Azure portal.

Azure Machine Learning allows you to implement both **online endpoints** and **batch endpoints**.

Multiple developer interfaces

Create and manage batch and online endpoints with multiple developer tools:

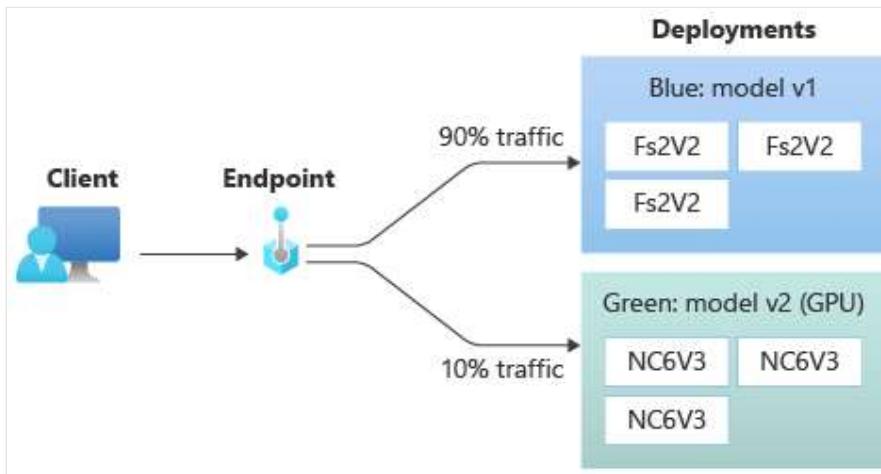
- The Azure CLI and the Python SDK
- Azure Resource Manager/REST API
- Azure Machine Learning studio web portal
- Azure portal (IT/Admin)
- Support for CI/CD MLOps pipelines using the Azure CLI interface & REST/ARM interfaces

What are online endpoints?

Online endpoints are endpoints that are used for online (real-time) inferencing.

Compared to **batch endpoints**, **online endpoints** contain **deployments** that are ready to receive data from clients and can send responses back in real time.

The following diagram shows an online endpoint that has two deployments, 'blue' and 'green'. The blue deployment uses VMs with a CPU SKU, and runs version 1 of a model. The green deployment uses VMs with a GPU SKU, and uses version 2 of the model. The endpoint is configured to route 90% of incoming traffic to the blue deployment, while green receives the remaining 10%.



Online deployments requirements

To create an online endpoint, you need to specify the following elements:

- Model files (or specify a registered model in your workspace)
- Scoring script - code needed to do scoring/inferencing
- Environment - a Docker image with Conda dependencies, or a dockerfile
- Compute instance & scale settings

Learn how to deploy online endpoints from the [CLI/SDK](#) and the [studio web portal](#).

Test and deploy locally for faster debugging

Deploy locally to test your endpoints without deploying to the cloud. Azure Machine Learning creates a local Docker image that mimics the Azure Machine Learning image. Azure Machine Learning will build and run deployments for you locally, and cache the image for rapid iterations.

Native blue/green deployment

Recall, that a single endpoint can have multiple deployments. The online endpoint can do load balancing to give any percentage of traffic to each deployment.

Traffic allocation can be used to do safe rollout blue/green deployments by balancing requests between different instances.

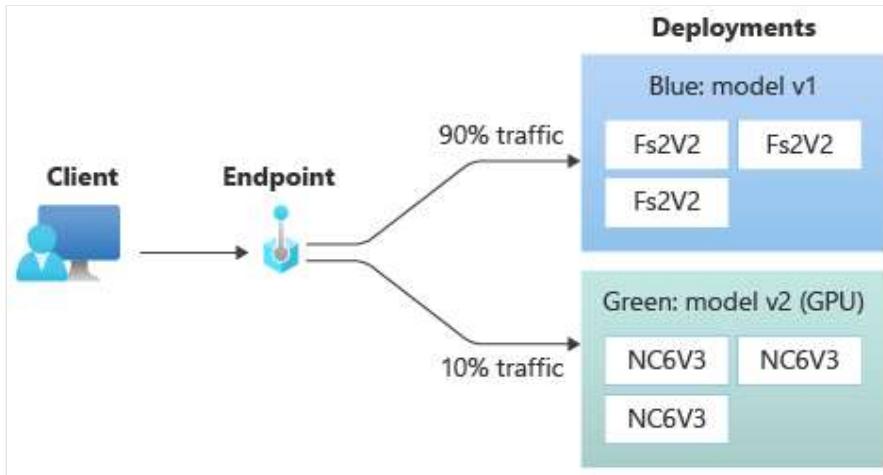
💡 Tip

A request can bypass the configured traffic load balancing by including an HTTP header of `azureml-model-deployment`. Set the header value to the name of the deployment you want the request to route to.

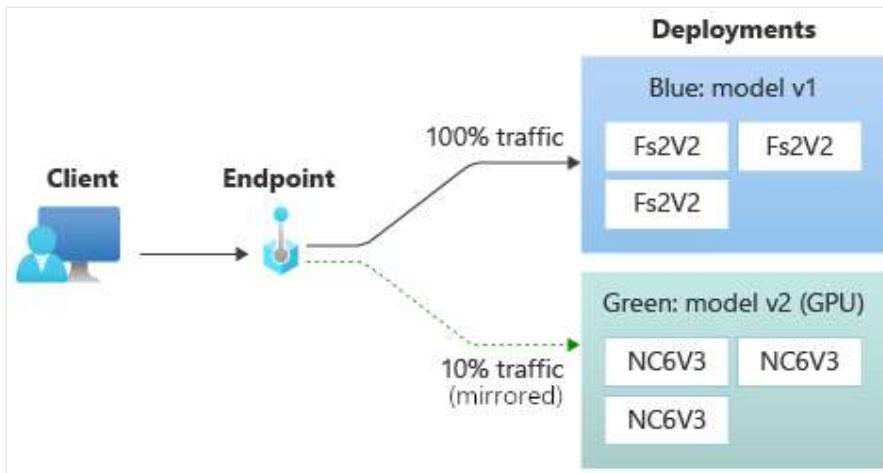
Update traffic allocation X

Ensure that allocated traffic between all deployments adds up to 0% or 100%

Deployment name	Traffic allocation %
green	<input type="text" value="10"/> <input type="range" value="10"/> 10%
blue	<input type="text" value="90"/> <input type="range" value="90"/> 90%
Total traffic percentage	100% <input type="button" value="Edit"/>



Traffic to one deployment can also be mirrored (or copied) to another deployment. Mirroring traffic (also called shadowing) is useful when you want to test for things like response latency or error conditions without impacting live clients; for example, when implementing a blue/green deployment where 100% of the traffic is routed to blue and 10% is mirrored to the green deployment. With mirroring, the results of the traffic to the green deployment aren't returned to the clients but metrics and logs are collected. Testing the new deployment with traffic mirroring/shadowing is also known as [shadow testing](#), and the functionality is currently a preview feature.



Learn how to [safely rollout to online endpoints](#).

Application Insights integration

All online endpoints integrate with Application Insights to monitor SLAs and diagnose issues.

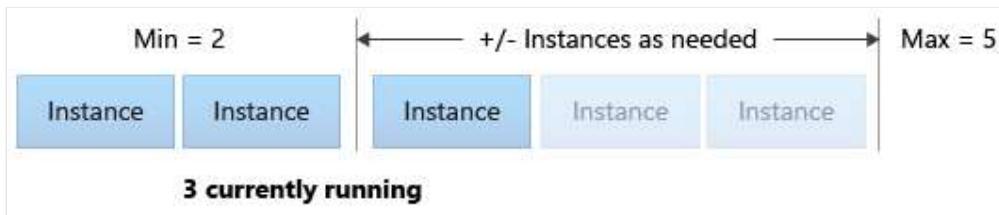
However [managed online endpoints](#) also include out-of-box integration with Azure Logs and Azure Metrics.

Security

- Authentication: Key and Azure Machine Learning Tokens
- Managed identity: User assigned and system assigned
- SSL by default for endpoint invocation

Autoscaling

Autoscale automatically runs the right amount of resources to handle the load on your application. Managed endpoints support autoscaling through integration with the [Azure monitor autoscale](#) feature. You can configure metrics-based scaling (for instance, CPU utilization >70%), schedule-based scaling (for example, scaling rules for peak business hours), or a combination.



Visual Studio Code debugging

Visual Studio Code enables you to interactively debug endpoints.

The screenshot shows a Visual Studio Code interface for debugging a Python file named `score.py`. The code implements a scoring endpoint using a pre-trained machine learning model. A breakpoint is set at line 15, where the model is loaded from a file. The interface includes a sidebar with `VARIABLES`, `WATCH`, `CALL STACK`, and `BREAKPOINTS` sections. The main editor shows the Python code with syntax highlighting and line numbers. The terminal at the bottom displays logs from an Azure ML Dev Container, indicating the start of an app insights client and request ID generator.

```
# AZUREML_MODEL_DIR is an environment variable created during deployment.
# It is the path to the model folder (. /azureml-models/$MODEL_NAME/$VERSION)
# For multiple models, it points to the folder containing all deployed models (. /azureml-models)
model_path = os.path.join(os.getenv("AZUREML_MODEL_DIR"), "sklearn_regression_model.pkl")
# deserialize the model file back into a sklearn model
model = joblib.load(model_path)
logging.info("Init complete")
```

Private endpoint support

Optionally, you can secure communication with a managed online endpoint by using private endpoints.

You can configure security for inbound scoring requests and outbound communications with the workspace and other services separately. Inbound communications use the private endpoint of the Azure Machine Learning workspace. Outbound communications use private endpoints created per deployment.

For more information, see [Secure online endpoints](#).

Managed online endpoints vs Kubernetes online endpoints

There are two types of online endpoints: **managed online endpoints** and **Kubernetes online endpoints**.

Managed online endpoints help to deploy your ML models in a turnkey manner.

Managed online endpoints work with powerful CPU and GPU machines in Azure in a scalable, fully managed way. Managed online endpoints take care of serving, scaling, securing, and monitoring your models, freeing you from the overhead of setting up and managing the underlying infrastructure. The main example in this doc uses managed online endpoints for deployment.

Kubernetes online endpoint allows you to deploy models and serve online endpoints at your fully configured and managed [Kubernetes cluster anywhere](#), with CPUs or GPUs.

The following table highlights the key differences between managed online endpoints and Kubernetes online endpoints.

	Managed online endpoints	Kubernetes online endpoints
Recommended users	Users who want a managed model deployment and enhanced MLOps experience	Users who prefer Kubernetes and can self-manage infrastructure requirements
Node provisioning	Managed compute provisioning, update, removal	User responsibility
Node maintenance	Managed host OS image updates, and security hardening	User responsibility

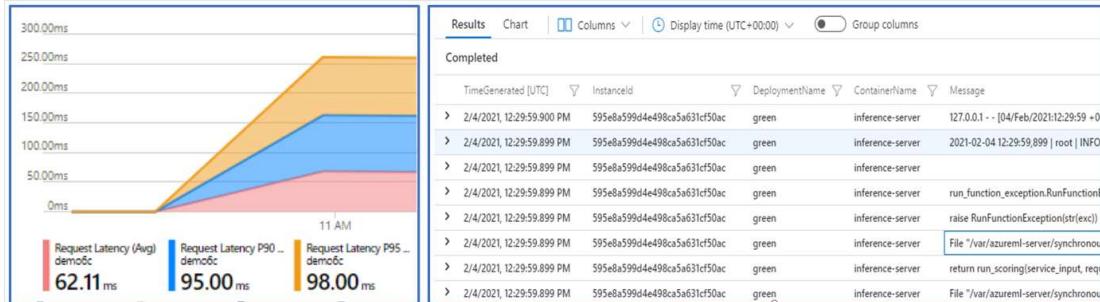
	Managed online endpoints	Kubernetes online endpoints
Cluster sizing (scaling)	Managed manual and autoscale, supporting additional nodes provisioning	Manual and autoscale, supporting scaling the number of replicas within fixed cluster boundaries
Compute type	Managed by the service	Customer-managed Kubernetes cluster (Kubernetes)
Managed identity	Supported	Supported
Virtual Network (VNET)	Supported via managed network isolation	User responsibility
Out-of-box monitoring & logging	Azure Monitor and Log Analytics powered (includes key metrics and log tables for endpoints and deployments)	User responsibility
Logging with Application Insights (legacy)	Supported	Supported
View costs	Detailed to endpoint / deployment level	Cluster level
Cost applied to	VMs assigned to the deployments	VMs assigned to the cluster
Mirrored traffic	Supported (preview)	Unsupported
No-code deployment	Supported (MLflow and Triton models)	Supported (MLflow and Triton models)

Managed online endpoints

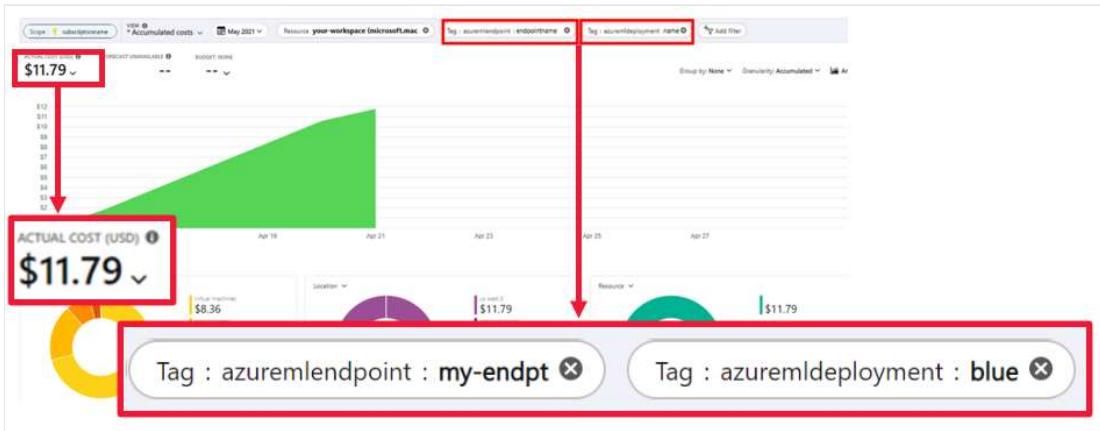
Managed online endpoints can help streamline your deployment process. Managed online endpoints provide the following benefits over Kubernetes online endpoints:

- Managed infrastructure
 - Automatically provisions the compute and hosts the model (you just need to specify the VM type and scale settings)
 - Automatically updates and patches the underlying host OS image
 - Automatic node recovery if there's a system failure
- Monitoring and logs

- Monitor model availability, performance, and SLA using [native integration with Azure Monitor](#).
- Debug deployments using the logs and native integration with Azure Log Analytics.



- View costs
 - Managed online endpoints let you [monitor cost at the endpoint and deployment level](#)



⚠ Note

Managed online endpoints are based on Azure Machine Learning compute.

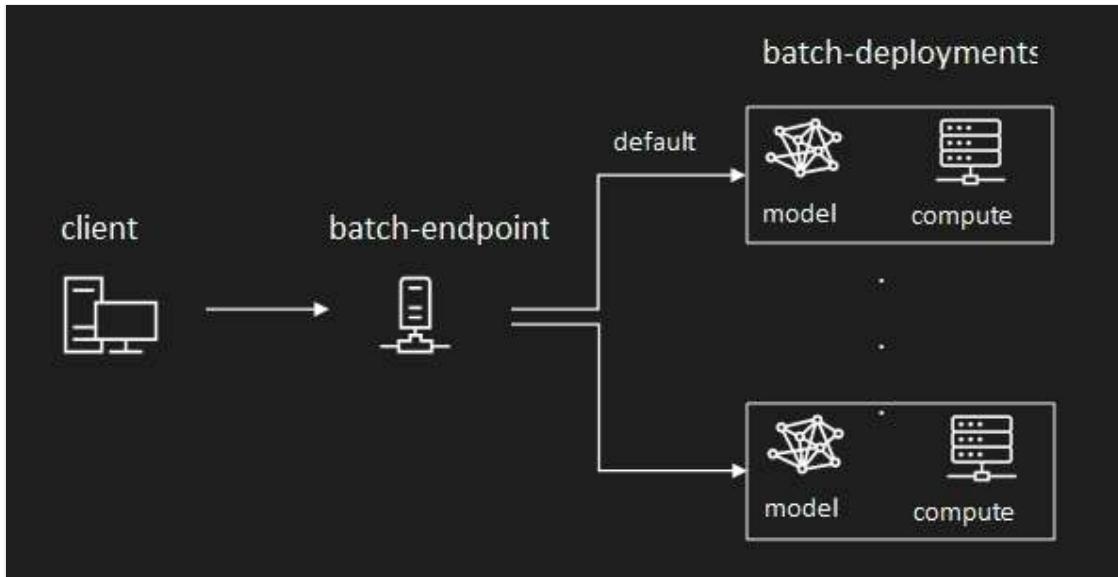
When using a managed online endpoint, you pay for the compute and networking charges. There is no additional surcharge.

If you use a virtual network and secure outbound (egress) traffic from the managed online endpoint, there is an additional cost. For egress, three private endpoints are created *per deployment* for the managed online endpoint. These are used to communicate with the default storage account, Azure Container Registry, and workspace. Additional networking charges may apply. For more information on pricing, see the [Azure pricing calculator](#).

For a step-by-step tutorial, see [How to deploy online endpoints](#).

What are batch endpoints?

Batch endpoints are endpoints that are used to do batch inferencing on large volumes of data over a period of time. Batch endpoints receive pointers to data and run jobs asynchronously to process the data in parallel on compute clusters. Batch endpoints store outputs to a data store for further analysis.



Batch deployment requirements

To create a batch deployment, you need to specify the following elements:

- Model files (or specify a model registered in your workspace)
- Compute
- Scoring script - code needed to do the scoring/inferencing
- Environment - a Docker image with Conda dependencies

If you're deploying [MLFlow models in batch deployments](#), there's no need to provide a scoring script and execution environment, as both are autogenerated.

Learn more about how to [deploy and use batch endpoints](#).

Managed cost with autoscaling compute

Invoking a batch endpoint triggers an asynchronous batch inference job. Compute resources are automatically provisioned when the job starts, and automatically deallocated as the job completes. So you only pay for compute when you use it.

You can [override compute resource settings](#) (like instance count) and advanced settings (like mini batch size, error threshold, and so on) for each individual batch inference job

to speed up execution and reduce cost.

Flexible data sources and storage

You can use the following options for input data when invoking a batch endpoint:

- Cloud data: Either a path on Azure Machine Learning registered datastore, a reference to Azure Machine Learning registered V2 data asset, or a public URI. For more information, see [Data in Azure Machine Learning](#).
- Data stored locally: The data will be automatically uploaded to the Azure Machine Learning registered datastore and passed to the batch endpoint.

Note

- If you're using existing V1 FileDatasets for batch endpoints, we recommend migrating them to V2 data assets. You can then refer to the V2 data assets directly when invoking batch endpoints. Currently, only data assets of type `uri_folder` or `uri_file` are supported. Batch endpoints created with GA CLIV2 (2.4.0 and newer) or GA REST API (2022-05-01 and newer) will not support V1 Datasets.
- You can also extract the datastores' URI or path from V1 FileDatasets. For this, you'll use the `az ml dataset show` command with the `--query` parameter and use that information for invoke.
- While batch endpoints created with earlier APIs will continue to support V1 FileDatasets, we'll be adding more support for V2 data assets in the latest API versions for better usability and flexibility. For more information on V2 data assets, see [Work with data using SDK v2](#). For more information on the new V2 experience, see [What is v2](#).

For more information on supported input options, see [Accessing data from batch endpoints jobs](#).

Specify the storage output location to any datastore and path. By default, batch endpoints store their output to the workspace's default blob store, organized by the Job Name (a system-generated GUID).

Security

- Authentication: Azure Active Directory Tokens

- SSL: enabled by default for endpoint invocation
- VNET support: Batch endpoints support ingress protection. A batch endpoint with ingress protection will accept scoring requests only from hosts inside a virtual network but not from the public internet. A batch endpoint that is created in a private-link enabled workspace will have ingress protection. To create a private-link enabled workspace, see [Create a secure workspace](#).

Note

Creating batch endpoints in a private-link enabled workspace is only supported in the following versions.

- CLI - version 2.15.1 or higher.
- REST API - version 2022-05-01 or higher.
- SDK V2 - version 0.1.0b3 or higher.

Next steps

- [How to deploy online endpoints with the Azure CLI and Python SDK](#)
- [How to deploy batch endpoints with the Azure CLI and Python SDK](#)
- [How to use online endpoints with the studio](#)
- [Deploy models with REST](#)
- [How to monitor managed online endpoints](#)
- [How to view managed online endpoint costs](#)
- [Manage and increase quotas for resources with Azure Machine Learning](#)

Additional resources

Documentation

[Profile model memory and CPU usage \(v1\) - Azure Machine Learning](#)

Use CLI (v1) or SDK (v1) to profile your model before deployment. Profiling determines the memory and CPU usage of your model.

[View costs for managed online endpoints - Azure Machine Learning](#)

Learn to how view costs for a managed online endpoint in Azure Machine Learning.

[JSONL format for computer vision tasks - Azure Machine Learning](#)

Learn how to format your JSONL files for data consumption in automated ML experiments for computer vision tasks with the CLI v2 and Python SDK v2.

[Deploy models to compute instances - Azure Machine Learning](#)

Learn how to deploy your Azure Machine Learning models as a web service using compute instances.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Autoscale online endpoints - Azure Machine Learning](#)

Learn to scale up online endpoints. Get more CPU, memory, disk space, and extra features.

[Search for assets - Azure Machine Learning](#)

Find your Azure Machine Learning assets with search

[Deploy a model for inference with GPU - Azure Machine Learning](#)

This article teaches you how to use Azure Machine Learning to deploy a GPU-enabled Tensorflow deep learning model as a web service.service and score inference requests.

[Show 5 more](#)

[Training](#)

Learning paths and modules

[Deploy an Azure Machine Learning model to a managed endpoint with CLI \(v2\) - Training](#)

Deploy an Azure Machine Learning model to a managed endpoint with CLI (v2)

ONNX and Azure Machine Learning: Create and accelerate ML models

Article • 11/04/2022 • 4 minutes to read

Learn how using the [Open Neural Network Exchange](#) (ONNX) can help optimize the inference of your machine learning model. Inference, or model scoring, is the phase where the deployed model is used for prediction, most commonly on production data.

Optimizing machine learning models for inference (or model scoring) is difficult since you need to tune the model and the inference library to make the most of the hardware capabilities. The problem becomes extremely hard if you want to get optimal performance on different kinds of platforms (cloud/edge, CPU/GPU, etc.), since each one has different capabilities and characteristics. The complexity increases if you have models from a variety of frameworks that need to run on a variety of platforms. It's very time consuming to optimize all the different combinations of frameworks and hardware. A solution to train once in your preferred framework and run anywhere on the cloud or edge is needed. This is where ONNX comes in.

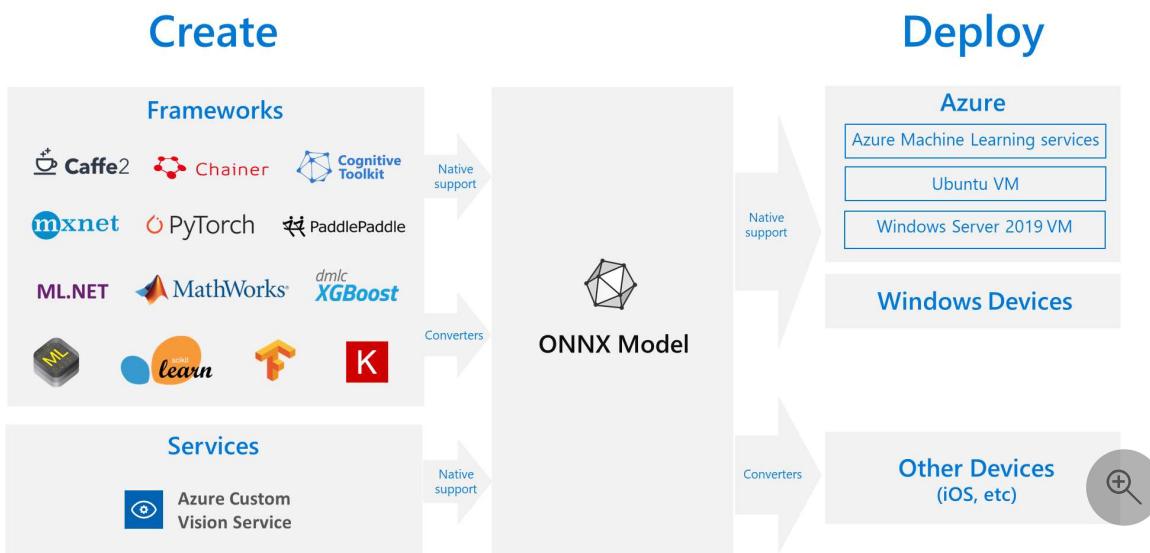
Microsoft and a community of partners created ONNX as an open standard for representing machine learning models. Models from [many frameworks](#) including TensorFlow, PyTorch, SciKit-Learn, Keras, Chainer, MXNet, MATLAB, and SparkML can be exported or converted to the standard ONNX format. Once the models are in the ONNX format, they can be run on a variety of platforms and devices.

[ONNX Runtime](#) is a high-performance inference engine for deploying ONNX models to production. It's optimized for both cloud and edge and works on Linux, Windows, and Mac. Written in C++, it also has C, Python, C#, Java, and JavaScript (Node.js) APIs for usage in a variety of environments. ONNX Runtime supports both DNN and traditional ML models and integrates with accelerators on different hardware such as TensorRT on NVidia GPUs, OpenVINO on Intel processors, DirectML on Windows, and more. By using ONNX Runtime, you can benefit from the extensive production-grade optimizations, testing, and ongoing improvements.

ONNX Runtime is used in high-scale Microsoft services such as Bing, Office, and Azure Cognitive Services. Performance gains are dependent on a number of factors, but these Microsoft services have seen an **average 2x performance gain on CPU**. In addition to Azure Machine Learning services, ONNX Runtime also runs in other products that support Machine Learning workloads, including:

- Windows: The runtime is built into Windows as part of [Windows Machine Learning](#) and runs on hundreds of millions of devices.

- Azure SQL product family: Run native scoring on data in [Azure SQL Edge](#) and [Azure SQL Managed Instance](#).
- ML.NET: Run ONNX models in [ML.NET](#).



Get ONNX models

You can obtain ONNX models in several ways:

- Train a new ONNX model in Azure Machine Learning (see examples at the bottom of this article) or by using [automated Machine Learning capabilities](#)
- Convert existing model from another format to ONNX (see the [tutorials](#))
- Get a pre-trained ONNX model from the [ONNX Model Zoo](#)
- Generate a customized ONNX model from [Azure Custom Vision service](#)

Many models including image classification, object detection, and text processing can be represented as ONNX models. If you run into an issue with a model that cannot be converted successfully, please file an issue in the GitHub of the respective converter that you used. You can continue using your existing format model until the issue is addressed.

Deploy ONNX models in Azure

With Azure Machine Learning, you can deploy, manage, and monitor your ONNX models. Using the standard [deployment workflow](#) and ONNX Runtime, you can create a REST endpoint hosted in the cloud. See example Jupyter notebooks at the end of this article to try it out for yourself.

Install and use ONNX Runtime with Python

Python packages for ONNX Runtime are available on [PyPi.org](#) (CPU, GPU). Please read [system requirements](#) before installation.

To install ONNX Runtime for Python, use one of the following commands:

```
Python

pip install onnxruntime      # CPU build
pip install onnxruntime-gpu  # GPU build
```

To call ONNX Runtime in your Python script, use:

```
Python

import onnxruntime
session = onnxruntime.InferenceSession("path to model")
```

The documentation accompanying the model usually tells you the inputs and outputs for using the model. You can also use a visualization tool such as [Netron](#) to view the model. ONNX Runtime also lets you query the model metadata, inputs, and outputs:

```
Python

session.get_modelmeta()
first_input_name = session.get_inputs()[0].name
first_output_name = session.get_outputs()[0].name
```

To inference your model, use `run` and pass in the list of outputs you want returned (leave empty if you want all of them) and a map of the input values. The result is a list of the outputs.

```
Python

results = session.run(["output1", "output2"], {
                      "input1": indata1, "input2": indata2})
results = session.run([], {"input1": indata1, "input2": indata2})
```

For the complete Python API reference, see the [ONNX Runtime reference docs](#).

Examples

See [how-to-use-azureml/deployment/onnx](#) for example Python notebooks that create and deploy ONNX models.

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Samples for usage in other languages can be found in the [ONNX Runtime GitHub](#).

More info

Learn more about **ONNX** or contribute to the project:

- [ONNX project website](#)
- [ONNX code on GitHub](#)

Learn more about **ONNX Runtime** or contribute to the project:

- [ONNX Runtime project website](#)
- [ONNX Runtime GitHub Repo](#)

Prebuilt Docker images for inference

Article • 12/02/2022 • 2 minutes to read

Prebuilt Docker container images for inference are used when deploying a model with Azure Machine Learning. The images are prebuilt with popular machine learning frameworks and Python packages. You can also extend the packages to add other packages by using one of the following methods:

Why should I use prebuilt images?

- Reduces model deployment latency.
- Improves model deployment success rate.
- Avoid unnecessary image build during model deployment.
- Only have required dependencies and access right in the image/container.

List of prebuilt Docker images for inference

ⓘ Important

The list provided below includes only **currently supported** inference docker images by Azure Machine Learning.

- All the docker images run as non-root user.
- We recommend using `latest` tag for docker images. Prebuilt docker images for inference are published to Microsoft container registry (MCR), to query list of tags available, follow [instructions on the GitHub repository](#).
- If you want to use a specific tag for any inference docker image, we support from `latest` to the tag that is *6 months* old from the `latest`.

Inference minimal base images

Framework version	CPU/GPU	Pre-installed packages	MCR Path
NA	CPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu18.04-py37-cpu-inference:latest</code>
NA	GPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu18.04-py37-cuda11.0.3-gpu-inference:latest</code>

Framework version	CPU/GPU	Pre-installed packages	MCR Path
NA	CPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu20.04-py38-cpu-inference:latest</code>
NA	GPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu20.04-py38-cuda11.6.2-gpu-inference:latest</code>

How to use inference prebuilt docker images?

Check examples in the Azure machine learning GitHub repository [↗](#)

Next steps

- [Deploy and score a machine learning model by using an online endpoint](#)
- [Learn more about custom containers](#)
- [azureml-examples GitHub repository ↗](#)

MLflow and Azure Machine Learning

Article • 02/24/2023 • 6 minutes to read

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) 

MLflow  is an open-source framework that's designed to manage the complete machine learning lifecycle. Its ability to train and serve models on different platforms allows you to use a consistent set of tools regardless of where your experiments are running: locally on your computer, on a remote compute target, on a virtual machine, or on an Azure Machine Learning compute instance.

Tip

Azure Machine Learning workspaces are MLflow-compatible, which means you can use Azure Machine Learning workspaces in the same way that you use an MLflow tracking server. Such compatibility has the following advantages:

- We don't host MLflow server instances under the hood. The workspace can talk the MLflow standard.
- You can use Azure Machine Learning workspaces as your tracking server for any MLflow code, whether it runs on Azure Machine Learning or not. You only need to configure MLflow to point to the workspace where the tracking should happen.
- You can run any training routine that uses MLflow in Azure Machine Learning without any change.

Note

Unlike the Azure Machine Learning SDK v1, there's no logging functionality in the SDK v2 and we recommend using MLflow for logging. Such strategy allows your training routines to become cloud-agnostic and portable, removing any dependency in your code with Azure Machine Learning.

Tracking with MLflow

Azure Machine Learning uses MLflow Tracking for metric logging and artifact storage for your experiments. When connected to Azure Machine Learning, all tracking performed

using MLflow is materialized in the workspace you are working on. To learn more about how to instrument your experiments for tracking experiments and training routines, see [Log metrics, parameters, and files with MLflow](#). You can also use MLflow to [Query & compare experiments and runs with MLflow](#).

Centralize tracking

You can connect MLflow to Azure Machine Learning workspaces even when you are running locally or in a different cloud. The workspace provides a centralized, secure, and scalable location to store training metrics and models.

Capabilities include:

- [Track machine learning experiments and models running locally or in the cloud](#) with MLflow in Azure Machine Learning.
- [Track Azure Databricks machine learning experiments](#) with MLflow in Azure Machine Learning.
- [Track Azure Synapse Analytics machine learning experiments](#) with MLflow in Azure Machine Learning.

Example notebooks

- [Training and tracking an XGBoost classifier with MLflow](#) : Demonstrates how to track experiments by using MLflow, log models, and combine multiple flavors into pipelines.
- [Training and tracking an XGBoost classifier with MLflow using service principal authentication](#) : Demonstrates how to track experiments by using MLflow from compute that's running outside Azure Machine Learning. It shows how to authenticate against Azure Machine Learning services by using a service principal.
- [Hyper-parameter optimization using Hyperopt and nested runs in MLflow](#) : Demonstrates how to use child runs in MLflow to do hyper-parameter optimization for models by using the popular library Hyperopt. It shows how to transfer metrics, parameters, and artifacts from child runs to parent runs.
- [Logging models with MLflow](#) : Demonstrates how to use the concept of models instead of artifacts with MLflow, including how to construct custom models.
- [Manage runs and experiments with MLflow](#) : Demonstrates how to query experiments, runs, metrics, parameters, and artifacts from Azure Machine Learning by using MLflow.

 **Important**

- MLflow in R support is limited to tracking experiment's metrics, parameters and models on Azure Machine Learning jobs. Interactive training on RStudio, Posit (formerly RStudio Workbench) or Jupyter Notebooks with R kernels is not supported. Model management and registration is not supported using the MLflow R SDK. As an alternative, use Azure Machine Learning CLI or [Azure Machine Learning studio](#) for model registration and management. View the following [R example about using the MLflow tracking client with Azure Machine Learning](#).
- MLflow in Java support is limited to tracking experiment's metrics and parameters on Azure Machine Learning jobs. Artifacts and models can't be tracked using the MLflow Java SDK. As an alternative, use the `outputs` folder in jobs along with the method `mlflow.save_model` to save models (or artifacts) you want to capture. View the following [Java example about using the MLflow tracking client with the Azure Machine Learning](#).

Model registries with MLflow

Azure Machine Learning supports MLflow for model management. This support represents a convenient way to support the entire model lifecycle for users who are familiar with the MLflow client.

To learn more about how to manage models by using the MLflow API in Azure Machine Learning, view [Manage model registries in Azure Machine Learning with MLflow](#).

Example notebooks

- [Manage model registries with MLflow](#): Demonstrates how to manage models in registries by using MLflow.

Model deployment with MLflow

You can [deploy MLflow models to Azure Machine Learning](#) and take advantage of the improved experience when you use this type of models. Azure Machine Learning supports deploying MLflow models to both real-time and batch endpoints without having to indicate an environment or a scoring script. Deployment is supported using either MLflow SDK, Azure Machine Learning CLI, Azure Machine Learning SDK for Python, or the [Azure Machine Learning studio](#) portal.

Learn more at [Guidelines for deploying MLflow models](#).

Example notebooks

- [Deploy MLflow to Online Endpoints](#) : Demonstrates how to deploy models in MLflow format to online endpoints using MLflow SDK.
- [Deploy MLflow to Online Endpoints with safe rollout](#) : Demonstrates how to deploy models in MLflow format to online endpoints using MLflow SDK with progressive rollout of models and the deployment of multiple model's versions in the same endpoint.
- [Deploy MLflow to web services \(V1\)](#) : Demonstrates how to deploy models in MLflow format to web services (ACI/AKS v1) using MLflow SDK.
- [Deploying models trained in Azure Databricks to Azure Machine Learning with MLflow](#) : Demonstrates how to train models in Azure Databricks and deploy them in Azure Machine Learning. It also includes how to handle cases where you also want to track the experiments with the MLflow instance in Azure Databricks.

Training MLflow projects (preview)

You can submit training jobs to Azure Machine Learning by using [MLflow projects](#) (preview). You can submit jobs locally with Azure Machine Learning tracking or migrate your jobs to the cloud via [Azure Machine Learning compute](#).

Learn more at [Train machine learning models with MLflow projects and Azure Machine Learning](#).

Example notebooks

- [Track an MLflow project in Azure Machine Learning workspaces](#)
- [Train and run an MLflow project on Azure Machine Learning jobs](#).

MLflow SDK, Azure Machine Learning v2, and Azure Machine Learning studio capabilities

The following table shows which operations are supported by each of the tools available in the machine learning lifecycle.

Feature	MLflow SDK	Azure Machine Learning CLI/SDK	Azure Machine Learning studio
---------	---------------	-----------------------------------	----------------------------------

Feature	MLflow SDK	Azure Machine Learning CLI/SDK	Azure Machine Learning studio
Track and log metrics, parameters, and models	✓		
Retrieve metrics, parameters, and models	✓	1	✓
Submit training jobs	✓ ²	✓	✓
Submit training jobs with Azure Machine learning data assets		✓	✓
Submit training jobs with machine learning pipelines		✓	✓
Manage experiments and runs	✓	✓	✓
Manage MLflow models	✓ ³	✓	✓
Manage non-MLflow models		✓	✓
Deploy MLflow models to Azure Machine Learning (Online & Batch)	✓ ⁴	✓	✓
Deploy non-MLflow models to Azure Machine Learning		✓	✓

➊ Note

- ¹ Only artifacts and models can be downloaded.
- ² Using MLflow projects (preview).
- ³ Some operations may not be supported. View [Manage model registries in Azure Machine Learning with MLflow](#) for details.
- ⁴ Deployment of MLflow models to batch inference by using the MLflow SDK is not possible at the moment. As an alternative, see [Deploy and run MLflow models in Spark jobs](#).

Next steps

- [Concept: From artifacts to models in MLflow](#).
- [How-to: Configure MLflow for Azure Machine Learning](#).
- [How-to: Migrate logging from SDK v1 to MLflow](#)
- [How-to: Track ML experiments and models with MLflow](#).

- [How-to: Log MLflow models.](#)
 - [Guidelines for deploying MLflow models.](#)
-

Additional resources

Documentation

[Experiment tracking and deploying models - Azure Data Science Virtual Machine](#)

Learn how to track and log experiments from the Data Science Virtual Machine with Azure Machine Learning and/or MLFlow.

[Detect data drift on datasets \(preview\) - Azure Machine Learning](#)

Learn how to set up data drift detection in Azure Learning. Create datasets monitors (preview), monitor for data drift, and set up alerts.

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Apache Spark in Azure Machine Learning \(preview\) - Azure Machine Learning](#)

This article explains the options for accessing Apache Spark in Azure Machine Learning.

[From artifacts to models in MLflow - Azure Machine Learning](#)

Learn about how MLflow uses the concept of models instead of artifacts to represent your trained models and enable a streamlined path to deployment.

[How to view AutoML model training code - Azure Machine Learning AutoML](#)

How to view model training code for an automated ML trained model and explanation of each stage.

[Troubleshoot automated ML experiments - Azure Machine Learning](#)

Learn how to troubleshoot and resolve issues in your automated machine learning experiments.

[Deploy MLflow models as web services - Azure Machine Learning](#)

Set up MLflow with Azure Machine Learning to deploy your ML models as an Azure web service.

[Show 5 more](#)

Training

Learning paths and modules

[Use MLflow with Azure Machine Learning jobs submitted with CLI \(v2\) - Training](#)

Use MLflow with Azure Machine Learning jobs submitted with CLI (v2)

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

From artifacts to models in MLflow

Article • 02/24/2023 • 6 minutes to read

The following article explains the differences between an artifact and a model in MLflow and how to transition from one to the other. It also explains how Azure Machine Learning uses the MLflow model's concept to enable streamlined deployment workflows.

What's the difference between an artifact and a model?

If you are not familiar with MLflow, you may not be aware of the difference between logging artifacts or files vs. logging MLflow models. There are some fundamental differences between the two:

Artifacts

Any file generated (and captured) from an experiment's run or job is an artifact. It may represent a model serialized as a Pickle file, the weights of a PyTorch or TensorFlow model, or even a text file containing the coefficients of a linear regression. Other artifacts can have nothing to do with the model itself, but they can contain configuration to run the model, pre-processing information, sample data, etc. As you can see, an artifact can come in any format.

You may have been logging artifacts already:

```
Python

filename = 'model.pkl'
with open(filename, 'wb') as f:
    pickle.dump(model, f)

mlflow.log_artifact(filename)
```

Models

A model in MLflow is also an artifact. However, we make stronger assumptions about this type of artifacts. Such assumptions provide a clear contract between the saved files and what they mean. When you log your models as artifacts (simple files), you need to know what the model builder meant for each of them in order to know how to load the

model for inference. On the contrary, MLflow models can be loaded using the contract specified in the [The MLModel format](#).

In Azure Machine Learning, logging models has the following advantages:

- ✓ You can deploy them on real-time or batch endpoints without providing an scoring script nor an environment.
- ✓ When deployed, Model's deployments have a Swagger generated automatically and the **Test** feature can be used in Azure Machine Learning studio.
- ✓ Models can be used as pipelines inputs directly.
- ✓ You can use the [Responsible AI dashboard \(preview\)](#).

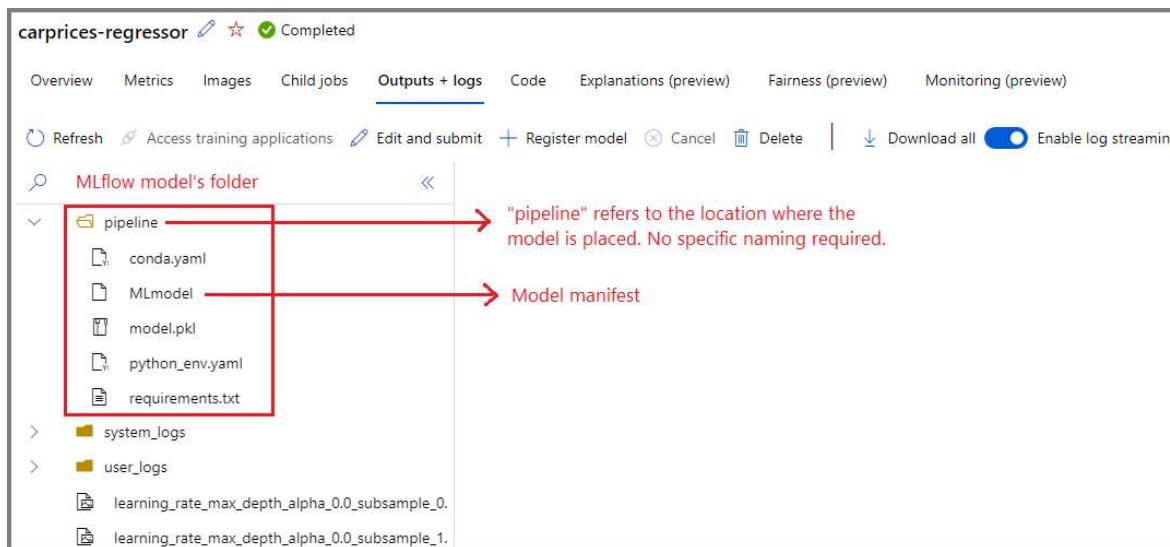
Models can get logged by using MLflow SDK:

Python

```
import mlflow
mlflow.sklearn.log_model(sklearn_estimator, "classifier")
```

The MLmodel format

MLflow adopts the MLmodel format as a way to create a contract between the artifacts and what they represent. The MLmodel format stores assets in a folder. Among them, there is a particular file named `MLmodel`. This file is the single source of truth about how a model can be loaded and used.



The following example shows how the `MLmodel` file for a computer vision model trained with `fastai` may look like:

`MLmodel`

YAML

```
artifact_path: classifier
flavors:
  fastai:
    data: model.fastai
    fastai_version: 2.4.1
  python_function:
    data: model.fastai
    env: conda.yaml
    loader_module: mlflow.fastai
    python_version: 3.8.12
model_uuid: e694c68eba484299976b06ab9058f636
run_id: e13da8ac-b1e6-45d4-a9b2-6a0a5cfac537
signature:
  inputs: '[{"type": "tensor",
            "tensor-spec":
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}
            }'
  outputs: '[{"type": "tensor",
             "tensor-spec":
               {"dtype": "float32", "shape": [-1, 2]}
             }]'
```

The model's flavors

Considering the variety of machine learning frameworks available to use, MLflow introduced the concept of flavor as a way to provide a unique contract to work across all of them. A flavor indicates what to expect for a given model created with a specific framework. For instance, TensorFlow has its own flavor, which specifies how a TensorFlow model should be persisted and loaded. Because each model flavor indicates how they want to persist and load models, the MLModel format doesn't enforce a single serialization mechanism that all the models need to support. Such decision allows each flavor to use the methods that provide the best performance or best support according to their best practices - without compromising compatibility with the MLModel standard.

The following is an example of the `flavors` section for an `fastai` model.

YAML

```
flavors:
  fastai:
    data: model.fastai
    fastai_version: 2.4.1
  python_function:
    data: model.fastai
    env: conda.yaml
```

```
loader_module: mlflow.fastai  
python_version: 3.8.12
```

Signatures

Model signatures in MLflow [↗](#) are an important part of the model specification, as they serve as a data contract between the model and the server running our models. They are also important for parsing and enforcing model's input's types at deployment time. MLflow enforces types when data is submitted to your model if a signature is available [↗](#).

Signatures are indicated when the model gets logged and persisted in the `MLmodel` file, in the `signature` section. Autolog's feature in MLflow automatically infers signatures in a best effort way. However, it may be required to [log the models manually if the signatures inferred are not the ones you need ↗](#).

There are two types of signatures:

- **Column-based signature** corresponding to signatures that operate to tabular data. For models with this signature, MLflow supplies `pandas.DataFrame` objects as inputs.
- **Tensor-based signature**: corresponding to signatures that operate with n-dimensional arrays or tensors. For models with this signature, MLflow supplies `numpy.ndarray` as inputs (or a dictionary of `numpy.ndarray` in the case of named-tensors).

The following example corresponds to a computer vision model trained with `fastai`.

This model receives a batch of images represented as tensors of shape `(300, 300, 3)` with the RGB representation of them (unsigned integers). It outputs batches of predictions (probabilities) for two classes.

MLmodel

YAML

```
signature:  
  inputs: '[{"type": "tensor",  
            "tensor-spec":  
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}  
          }]  
  outputs: '[{"type": "tensor",  
             "tensor-spec":  
               {"dtype": "float32", "shape": [-1, 2]}  
           }]
```

💡 Tip

Azure Machine Learning generates Swagger for model's deployment in MLflow format with a signature available. This makes easier to test deployed endpoints using the Azure Machine Learning studio.

Model's environment

Requirements for the model to run are specified in the `conda.yaml` file. Dependencies can be automatically detected by MLflow or they can be manually indicated when you call `mlflow.<flavor>.log_model()` method. The latter can be needed in cases that the libraries included in your environment are not the ones you intended to use.

The following is an example of an environment used for a model created with `fastai` framework:

conda.yaml

YAML

```
channels:
- conda-forge
dependencies:
- python=3.8.5
- pip
- pip:
  - mlflow
  - astunparse==1.6.3
  - cffi==1.15.0
  - configparser==3.7.4
  - defusedxml==0.7.1
  - fastai==2.4.1
  - google-api-core==2.7.1
  - ipython==8.2.0
  - psutil==5.9.0
name: mlflow-env
```

⚠ Note

MLflow environments and Azure Machine Learning environments are different concepts. While the former operates at the level of the model, the latter operates at the level of the workspace (for registered environments) or jobs/deployments (for anonymous environments). When you deploy MLflow models in Azure Machine Learning, the model's environment is built and used for deployment.

Alternatively, you can override this behaviour with the [Azure Machine Learning CLI v2](#) and deploy MLflow models using a specific Azure Machine Learning environments.

Model's predict function

All MLflow models contain a `predict` function. **This function is the one that is called when a model is deployed using a no-code-deployment experience.** What the `predict` function returns (classes, probabilities, a forecast, etc.) depend on the framework (i.e. flavor) used for training. Read the documentation of each flavor to know what they return.

In some cases, you may need to customize this function to change the way inference is executed. On those cases, you will need to [log models with a different behavior in the predict method](#) or [log a custom model's flavor](#).

Loading MLflow models back

Models created as MLflow models can be loaded back directly from the run where they were logged, from the file system where they are saved or from the model registry where they are registered. MLflow provides a consistent way to load those models regardless of the location.

There are two workflows available for loading models:

- **Loading back the same object and types that were logged:** You can load models using MLflow SDK and obtain an instance of the model with types belonging to the training library. For instance, an ONNX model will return a `ModelProto` while a decision tree trained with Scikit-Learn model will return a `DecisionTreeClassifier` object. Use `mlflow.<flavor>.load_model()` to do so.
- **Loading back a model for running inference:** You can load models using MLflow SDK and obtain a wrapper where MLflow guarantees there will be a `predict` function. It doesn't matter which flavor you are using, every MLflow model needs to implement this contract. Furthermore, MLflow guarantees that this function can be called using arguments of type `pandas.DataFrame`, `numpy.ndarray` or `dict[string, numpyndarray]` (depending on the signature of the model). MLflow handles the type conversion to the input type the model actually expects. Use `mlflow.pyfunc.load_model()` to do so.

Start logging models

We recommend starting taking advantage of MLflow models in Azure Machine Learning. There are different ways to start using the model's concept with MLflow. Read [How to log MLFlow models](#) to a comprehensive guide.

Additional resources

Documentation

[Using MLflow models in batch deployments - Azure Machine Learning](#)

Learn how to deploy MLflow models in batch deployments

[Train with MLflow Projects \(Preview\) - Azure Machine Learning](#)

Set up MLflow with Azure Machine Learning to log metrics and artifacts from ML models

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Configure MLflow for Azure Machine Learning - Azure Machine Learning](#)

Connect MLflow to Azure Machine Learning workspaces to log metrics, artifacts and deploy models.

[Manage models registries in Azure Machine Learning with MLflow - Azure Machine Learning](#)

Explains how to use MLflow for managing models in Azure Machine Learning

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Machine Learning registries \(preview\) - Azure Machine Learning](#)

Learn what are Azure Machine Learning registries and how to use to for MLOps

[Show 5 more](#)

Training

Learning paths and modules

[Use MLflow with Azure Machine Learning jobs submitted with CLI \(v2\) - Training](#)

Use MLflow with Azure Machine Learning jobs submitted with CLI (v2)

MLOps: Model management, deployment, and monitoring with Azure Machine Learning

Article • 02/10/2023 • 7 minutes to read

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)  [Python SDK azure-ai-ml v2 \(current\)](#) ↗

In this article, learn how to apply Machine Learning Operations (MLOps) practices in Azure Machine Learning for the purpose of managing the lifecycle of your models. Applying MLOps practices can improve the quality and consistency of your machine learning solutions.

What is MLOps?

MLOps is based on [DevOps](#) ↗ principles and practices that increase the efficiency of workflows. Examples include continuous integration, delivery, and deployment. MLOps applies these principles to the machine learning process, with the goal of:

- Faster experimentation and development of models.
- Faster deployment of models into production.
- Quality assurance and end-to-end lineage tracking.

MLOps in Machine Learning

Machine Learning provides the following MLOps capabilities:

- **Create reproducible machine learning pipelines.** Use machine learning pipelines to define repeatable and reusable steps for your data preparation, training, and scoring processes.
- **Create reusable software environments.** Use these environments for training and deploying models.
- **Register, package, and deploy models from anywhere.** You can also track associated metadata required to use the model.
- **Capture the governance data for the end-to-end machine learning lifecycle.** The logged lineage information can include who is publishing models and why changes were made. It can also include when models were deployed or used in production.

- **Notify and alert on events in the machine learning lifecycle.** Event examples include experiment completion, model registration, model deployment, and data drift detection.
- **Monitor machine learning applications for operational and machine learning-related issues.** Compare model inputs between training and inference. Explore model-specific metrics. Provide monitoring and alerts on your machine learning infrastructure.
- **Automate the end-to-end machine learning lifecycle with Machine Learning and Azure Pipelines.** By using pipelines, you can frequently update models. You can also test new models. You can continually roll out new machine learning models alongside your other applications and services.

For more information on MLOps, see [Machine learning DevOps](#).

Create reproducible machine learning pipelines

Use machine learning pipelines from Machine Learning to stitch together all the steps in your model training process.

A machine learning pipeline can contain steps from data preparation to feature extraction to hyperparameter tuning to model evaluation. For more information, see [Machine learning pipelines](#).

If you use the [designer](#) to create your machine learning pipelines, you can at any time select the ... icon in the upper-right corner of the designer page. Then select **Clone**. When you clone your pipeline, you iterate your pipeline design without losing your old versions.

Create reusable software environments

By using Machine Learning environments, you can track and reproduce your projects' software dependencies as they evolve. You can use environments to ensure that builds are reproducible without manual software configurations.

Environments describe the pip and conda dependencies for your projects. You can use them for training and deployment of models. For more information, see [What are Machine Learning environments?](#).

Register, package, and deploy models from anywhere

The following sections discuss how to register, package, and deploy models.

Register and track machine learning models

With model registration, you can store and version your models in the Azure cloud, in your workspace. The model registry makes it easy to organize and keep track of your trained models.

💡 Tip

A registered model is a logical container for one or more files that make up your model. For example, if you have a model that's stored in multiple files, you can register them as a single model in your Machine Learning workspace. After registration, you can then download or deploy the registered model and receive all the files that were registered.

Registered models are identified by name and version. Each time you register a model with the same name as an existing one, the registry increments the version. More metadata tags can be provided during registration. These tags are then used when you search for a model. Machine Learning supports any model that can be loaded by using Python 3.5.2 or higher.

💡 Tip

You can also register models trained outside Machine Learning.

ⓘ Important

- When you use the **Filter by Tags** option on the **Models** page of Azure Machine Learning Studio, instead of using `TagName : TagValue`, use `TagName=TagValue` without spaces.
- You can't delete a registered model that's being used in an active deployment.

For more information, [Work with models in Azure Machine Learning](#).

Package and debug models

Before you deploy a model into production, it's packaged into a Docker image. In most cases, image creation happens automatically in the background during deployment. You

can manually specify the image.

If you run into problems with the deployment, you can deploy on your local development environment for troubleshooting and debugging.

For more information, see [How to troubleshoot online endpoints](#).

Convert and optimize models

Converting your model to [Open Neural Network Exchange](#) (ONNX) might improve performance. On average, converting to ONNX can double performance.

For more information on ONNX with Machine Learning, see [Create and accelerate machine learning models](#).

Use models

Trained machine learning models are deployed as [endpoints](#) in the cloud or locally. Deployments use CPU, GPU for inferencing.

When deploying a model as an endpoint, you provide the following items:

- The models that are used to score data submitted to the service or device.
- An entry script. This script accepts requests, uses the models to score the data, and returns a response.
- A Machine Learning environment that describes the pip and conda dependencies required by the models and entry script.
- Any other assets such as text and data that are required by the models and entry script.

You also provide the configuration of the target deployment platform. For example, the VM family type, available memory, and number of cores. When the image is created, components required by Azure Machine Learning are also added. For example, assets needed to run the web service.

Batch scoring

Batch scoring is supported through batch endpoints. For more information, see [endpoints](#).

Online endpoints

You can use your models with an online endpoint. Online endpoints can use the following compute targets:

- Managed online endpoints
- Azure Kubernetes Service
- Local development environment

To deploy the model to an endpoint, you must provide the following items:

- The model or ensemble of models.
- Dependencies required to use the model. Examples are a script that accepts requests and invokes the model and conda dependencies.
- Deployment configuration that describes how and where to deploy the model.

For more information, see [Deploy online endpoints](#).

Controlled rollout

When deploying to an online endpoint, you can use controlled rollout to enable the following scenarios:

- Create multiple versions of an endpoint for a deployment
- Perform A/B testing by routing traffic to different deployments within the endpoint.
- Switch between endpoint deployments by updating the traffic percentage in endpoint configuration.

For more information, see [Controlled rollout of machine learning models](#).

Analytics

Microsoft Power BI supports using machine learning models for data analytics. For more information, see [Machine Learning integration in Power BI \(preview\)](#).

Capture the governance data required for MLOps

Machine Learning gives you the capability to track the end-to-end audit trail of all your machine learning assets by using metadata. For example:

- [Machine Learning datasets](#) help you track, profile, and version data.

- [Interpretability](#) allows you to explain your models, meet regulatory compliance, and understand how models arrive at a result for specific input.
- Machine Learning Job history stores a snapshot of the code, data, and computes used to train a model.
- The [Machine Learning Model Registry](#) captures all the metadata associated with your model. For example, metadata includes which experiment trained it, where it's being deployed, and if its deployments are healthy.
- [Integration with Azure](#) allows you to act on events in the machine learning lifecycle. Examples are model registration, deployment, data drift, and training (job) events.

 **Tip**

While some information on models and datasets is automatically captured, you can add more information by using *tags*. When you look for registered models and datasets in your workspace, you can use tags as a filter.

Notify, automate, and alert on events in the machine learning lifecycle

Machine Learning publishes key events to Azure Event Grid, which can be used to notify and automate on events in the machine learning lifecycle. For more information, see [Use Event Grid](#).

Automate the machine learning lifecycle

You can use GitHub and Azure Pipelines to create a continuous integration process that trains a model. In a typical scenario, when a data scientist checks a change into the Git repo for a project, Azure Pipelines starts a training job. The results of the job can then be inspected to see the performance characteristics of the trained model. You can also create a pipeline that deploys the model as a web service.

The [Machine Learning extension](#) makes it easier to work with Azure Pipelines. It provides the following enhancements to Azure Pipelines:

- Enables workspace selection when you define a service connection.
- Enables release pipelines to be triggered by trained models created in a training pipeline.

For more information on using Azure Pipelines with Machine Learning, see:

- Continuous integration and deployment of machine learning models with Azure Pipelines
- Machine Learning MLOps ↗ repository

Next steps

Learn more by reading and exploring the following resources:

- Set up MLOps with Azure DevOps
 - Learning path: End-to-end MLOps with Azure Machine Learning
 - How to deploy a model to an online endpoint with Machine Learning
 - Tutorial: Train and deploy a model
 - CI/CD of machine learning models with Azure Pipelines
 - Machine learning at scale
 - Azure AI reference architectures and best practices repo ↗
-

Additional resources

Documentation

[GitHub Actions for CI/CD - Azure Machine Learning](#)

Learn about how to create a GitHub Actions workflow to train a model on Azure Machine Learning

[Architecture & key concepts \(v1\) - Azure Machine Learning](#)

This article gives you a high-level understanding of the architecture, terms, and concepts that make up Azure Machine Learning.

[Detect data drift on datasets \(preview\) - Azure Machine Learning](#)

Learn how to set up data drift detection in Azure Learning. Create datasets monitors (preview), monitor for data drift, and set up alerts.

[MLflow and Azure Machine Learning - Azure Machine Learning](#)

Learn about how Azure Machine Learning uses MLflow to log metrics and artifacts from machine learning models, and to deploy your machine learning models to an endpoint.

[Git integration for Azure Machine Learning - Azure Machine Learning](#)

Learn how Azure Machine Learning integrates with a local Git repository to track repository, branch, and current commit information as part of a training job.

[What are machine learning pipelines? - Azure Machine Learning](#)

Learn how machine learning pipelines help you build, optimize, and manage machine learning workflows.

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Example Jupyter Notebooks \(v2\) - Azure Machine Learning](#)

Learn how to find and use the Jupyter Notebooks designed to help you explore the SDK (v2) and serve as models for your own machine learning projects.

[Show 5 more](#)

Training

Learning paths and modules

[Build and operate machine learning solutions with Azure Machine Learning - Training](#)

Build and operate machine learning solutions with Azure Machine Learning

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

Machine Learning registries (preview) for MLOps

Article • 02/24/2023 • 2 minutes to read

In this article, you'll learn how to scale MLOps across development, testing and production environments. Your environments can vary from few to many based on the complexity of your IT environment and is influenced by factors such as:

- Security and compliance policies - do production environments need to be isolated from development environments in terms of access controls, network architecture, data exposure, etc.?
- Subscriptions - Are your development environments in one subscription and production environments in a different subscription? Often separate subscriptions are used to account for billing, budgeting, and cost management purposes.
- Regions - Do you need to deploy to different Azure regions to support latency and redundancy requirements?

In such scenarios, you may be using different Azure Machine Learning workspaces for development, testing and production. This configuration presents the following challenges for model training and deployment:

- You need to train a model in a development workspace but deploy it an endpoint in a production workspace, possibly in a different Azure subscription or region. In this case, you must be able to trace back the training job. For example, to analyze the metrics, logs, code, environment, and data used to train the model if you encounter accuracy or performance issues with the production deployment.
- You need to develop a training pipeline with test data or anonymized data in the development workspace but retrain the model with production data in the production workspace. In this case, you may need to compare training metrics on sample vs production data to ensure the training optimizations are performing well with actual data.

Cross-workspace MLOps with registries

Registries, much like a Git repository, decouples ML assets from workspaces and hosts them in a central location, making them available to all workspaces in your organization.

If you want to promote models across environments (dev, test, prod), start by iteratively developing a model in dev. When you have a good candidate model, you can publish it

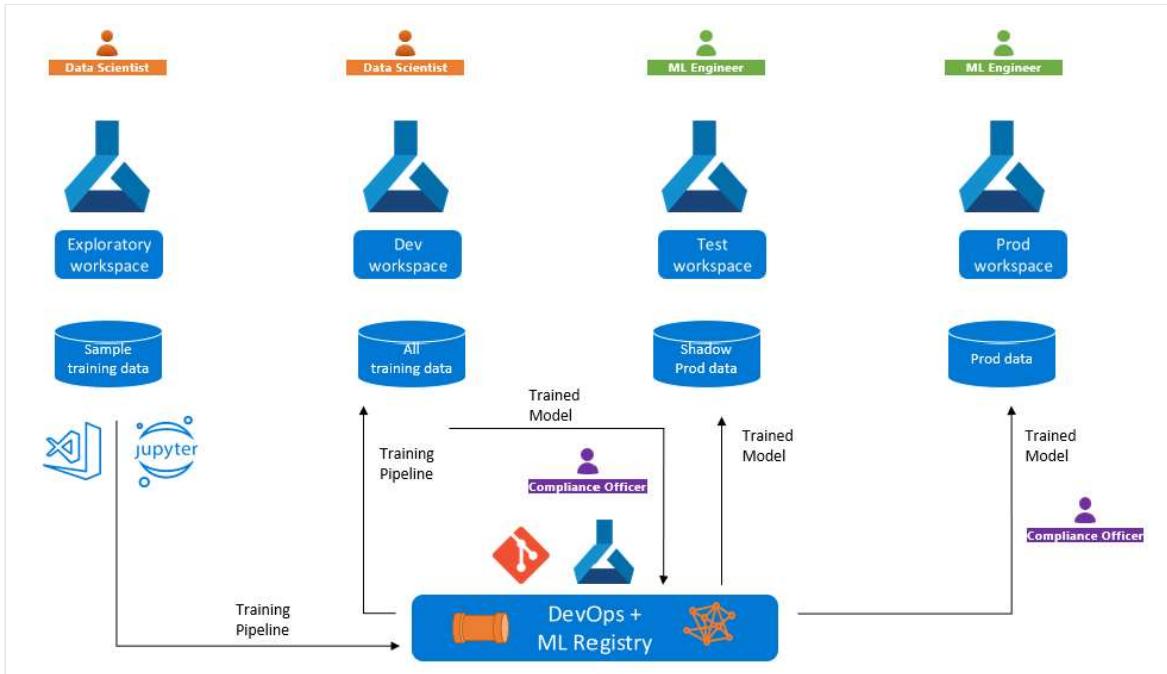
to a registry. You can then deploy the model from the registry to endpoints in different workspaces.

Tip

If you already have models registered in a workspace, you can promote them to a registry. You can also register a model directly in a registry from the output of a training job.

If you want to develop a pipeline in one workspace and then run it in others, start by registering the components and environments that form the building blocks of the pipeline. When you submit the pipeline job, the workspace it runs in is selected by the compute and training data, which are unique to each workspace.

The following diagram illustrates promotion of pipelines between exploratory and dev workspaces, then model promotion between dev, test, and production.



Next steps

- [Create a registry](#).
- [Share models, components, and environments using registries](#).

Additional resources

Documentation

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Train with MLflow Projects \(Preview\) - Azure Machine Learning](#)

Set up MLflow with Azure Machine Learning to log metrics and artifacts from ML models

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Customize outputs in batch deployments - Azure Machine Learning](#)

Learn how create deployments that generate custom outputs and files.

[Profile model memory and CPU usage \(v1\) - Azure Machine Learning](#)

Use CLI (v1) or SDK (v1) to profile your model before deployment. Profiling determines the memory and CPU usage of your model.

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Show 5 more](#)

What are Azure Machine Learning pipelines?

Article • 02/24/2023 • 5 minutes to read

APPLIES TO:  Azure CLI ml extension v1  Python SDK azureml v1

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) ↗

An Azure Machine Learning pipeline is an independently executable workflow of a complete machine learning task. An Azure Machine Learning pipeline helps to standardize the best practices of producing a machine learning model, enables the team to execute at scale, and improves the model building efficiency.

Why are Azure Machine Learning pipelines needed?

The core of a machine learning pipeline is to split a complete machine learning task into a multistep workflow. Each step is a manageable component that can be developed, optimized, configured, and automated individually. Steps are connected through well-defined interfaces. The Azure Machine Learning pipeline service automatically orchestrates all the dependencies between pipeline steps. This modular approach brings two key benefits:

- Standardize the Machine learning operation (MLOps) practice and support scalable team collaboration
- Training efficiency and cost reduction

Standardize the MLOps practice and support scalable team collaboration

Machine learning operation (MLOps) automates the process of building machine learning models and taking the model to production. This is a complex process. It usually requires collaboration from different teams with different skills. A well-defined machine learning pipeline can abstract this complex process into a multiple steps workflow, mapping each step to a specific task such that each team can work independently.

For example, a typical machine learning project includes the steps of data collection, data preparation, model training, model evaluation, and model deployment. Usually, the data engineers concentrate on data steps, data scientists spend most time on model training and evaluation, the machine learning engineers focus on model deployment and automation of the entire workflow. By leveraging machine learning pipeline, each team only needs to work on building their own steps. The best way of building steps is using [Azure Machine Learning component](#), a self-contained piece of code that does one step in a machine learning pipeline. All these steps built by different users are finally integrated into one workflow through the pipeline definition. The pipeline is a collaboration tool for everyone in the project. The process of defining a pipeline and all its steps can be standardized by each company's preferred DevOps practice. The pipeline can be further versioned and automated. If the ML projects are described as a pipeline, then the best MLOps practice is already applied.

Training efficiency and cost reduction

Besides being the tool to put MLOps into practice, the machine learning pipeline also improves large model training's efficiency and reduces cost. Taking modern natural language model training as an example. It requires pre-processing large amounts of data and GPU intensive transformer model training. It takes hours to days to train a model each time. When the model is being built, the data scientist wants to test different training code or hyperparameters and run the training many times to get the best model performance. For most of these trainings, there's usually small changes from one training to another one. It will be a significant waste if every time the full training from data processing to model training takes place. By using machine learning pipeline, it can automatically calculate which steps result is unchanged and reuse outputs from previous training. Additionally, the machine learning pipeline supports running each step on different computation resources. Such that, the memory heavy data processing work and run-on high memory CPU machines, and the computation intensive training can run on expensive GPU machines. By properly choosing which step to run on which type of machines, the training cost can be significantly reduced.

Getting started best practices

Depending on what a machine learning project already has, the starting point of building a machine learning pipeline may vary. There are a few typical approaches to building a pipeline.

The first approach usually applies to the team that hasn't used pipeline before and wants to take some advantage of pipeline like MLOps. In this situation, data scientists

typically have developed some machine learning models on their local environment using their favorite tools. Machine learning engineers need to take data scientists' output into production. The work involves cleaning up some unnecessary code from original notebook or Python code, changes the training input from local data to parameterized values, split the training code into multiple steps as needed, perform unit test of each step, and finally wraps all steps into a pipeline.

Once the teams get familiar with pipelines and want to do more machine learning projects using pipelines, they'll find the first approach is hard to scale. The second approach is set up a few pipeline templates, each try to solve one specific machine learning problem. The template predefines the pipeline structure including how many steps, each step's inputs and outputs, and their connectivity. To start a new machine learning project, the team first forks one template repo. The team leader then assigns members which step they need to work on. The data scientists and data engineers do their regular work. When they're happy with their result, they structure their code to fit in the pre-defined steps. Once the structured codes are checked-in, the pipeline can be executed or automated. If there's any change, each member only needs to work on their piece of code without touching the rest of the pipeline code.

Once a team has built a collection of machine learnings pipelines and reusable components, they could start to build the machine learning pipeline from cloning previous pipeline or tie existing reusable component together. At this stage, the team's overall productivity will be improved significantly.

Azure Machine Learning offers different methods to build a pipeline. For users who are familiar with DevOps practices, we recommend using [CLI](#). For data scientists who are familiar with python, we recommend writing pipeline using the [Azure Machine Learning SDK v1](#). For users who prefer to use UI, they could use the [designer to build pipeline by using registered components](#).

Which Azure pipeline technology should I use?

The Azure cloud provides several types of pipeline, each with a different purpose. The following table lists the different pipelines and what they're used for:

Scenario	Primary persona	Azure offering	OSS offering	Canonical pipe	Strengths
Model orchestration (Machine learning)	Data scientist	Azure Machine Learning Pipelines	Kubeflow Pipelines	Data -> Model	Distribution, caching, code-first, reuse

Scenario	Primary persona	Azure offering	OSS offering	Canonical pipe	Strengths
Data orchestration (Data prep)	Data engineer	Azure Data Factory pipelines	Apache Airflow	Data -> Data	Strongly typed movement, data-centric activities
Code & app orchestration (CI/CD)	App Developer / Ops	Azure Pipelines ↗	Jenkins	Code + Model -> App/Service	Most open and flexible activity support, approval queues, phases with gating

Next steps

Azure Machine Learning pipelines are a powerful facility that begins delivering value in the early development stages.

- Define pipelines with the Azure Machine Learning CLI v2
- Define pipelines with the Azure Machine Learning SDK v2
- Define pipelines with Designer
- Try out [CLI v2 pipeline example ↗](#)
- Try out [Python SDK v2 pipeline example ↗](#)

Additional resources

Documentation

[MLOps: ML model management v1 - Azure Machine Learning](#)

Learn about model management (MLOps) with Azure Machine Learning. Deploy, manage, track lineage and monitor your models to continuously improve them. (v1)

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Experiment tracking and deploying models - Azure Data Science Virtual Machine](#)

Learn how to track and log experiments from the Data Science Virtual Machine with Azure Machine Learning and/or MLFlow.

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Assess AI systems and make data-driven decisions with Azure Machine Learning Responsible AI dashboard - Azure Machine Learning](#)

Learn how to use the comprehensive UI and SDK/YAML components in the Responsible AI dashboard to debug your machine learning models and make data-driven decisions.

[Train deep learning Keras models \(SDK v2\) - Azure Machine Learning](#)

Learn how to train and register a Keras deep neural network classification model running on TensorFlow using Azure Machine Learning SDK (v2).

[Execute Python Script in the designer - Azure Machine Learning](#)

Learn how to use the Execute Python Script model in Azure Machine Learning designer to run custom operations written in Python.

[Azure Machine Learning glossary - Azure Machine Learning](#)

Glossary of terms for the Azure Machine Learning platform.

[Show 5 more](#)

Training

Learning paths and modules

[Orchestrate machine learning with pipelines - Training](#)

Orchestrate machine learning with pipelines

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

What is an Azure Machine Learning component?

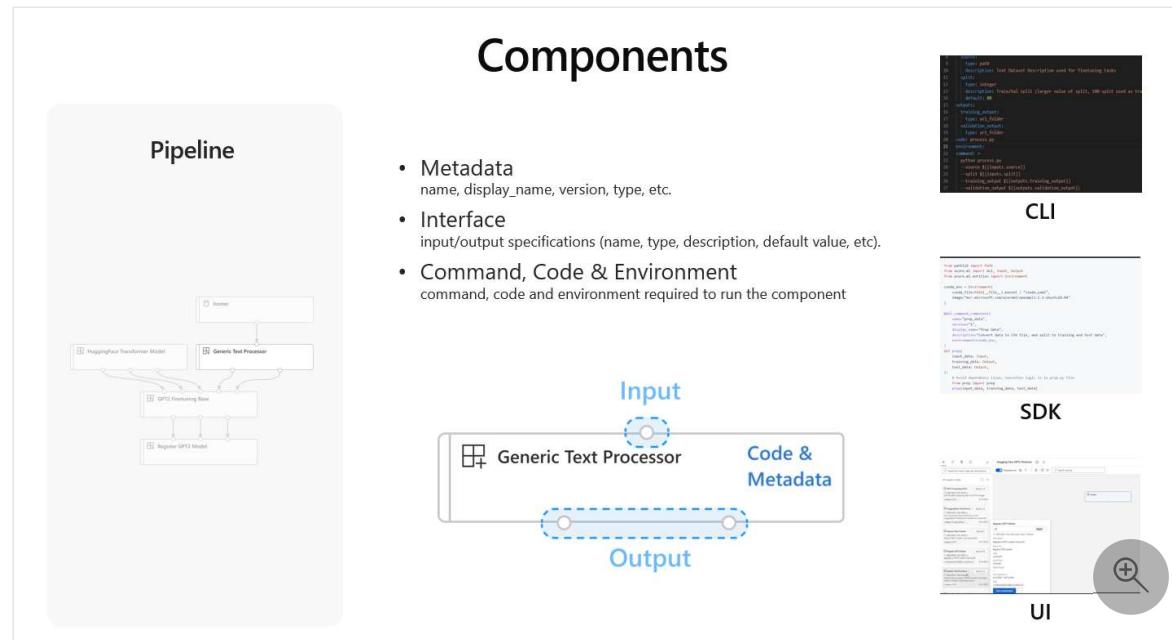
Article • 02/24/2023 • 3 minutes to read

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) 

An Azure Machine Learning component is a self-contained piece of code that does one step in a machine learning pipeline. A component is analogous to a function - it has a name, inputs, outputs, and a body. Components are the building blocks of the [Azure Machine Learning pipelines](#).

A component consists of three parts:

- Metadata: name, display_name, version, type, etc.
 - Interface: input/output specifications (name, type, description, default value, etc.).
 - Command, Code & Environment: command, code and environment required to run the component.



Why should I use a component?

It's a good engineering practice to build a machine learning pipeline to split a complete machine learning task into a multi-step workflow. Such that, everyone can work on the specific step independently. In Azure Machine Learning, a component represents one

reusable step in a pipeline. Components are designed to help improve the productivity of pipeline building. Specifically, components offer:

- **Well-defined interface:** Components require a well-defined interface (input and output). The interface allows the user to build steps and connect steps easily. The interface also hides the complex logic of a step and removes the burden of understanding how the step is implemented.
- **Share and reuse:** As the building blocks of a pipeline, components can be easily shared and reused across pipelines, workspaces, and subscriptions. Components built by one team can be discovered and used by another team.
- **Version control:** Components are versioned. The component producers can keep improving components and publish new versions. Consumers can use specific component versions in their pipelines. This gives them compatibility and reproducibility.

Unit testable: A component is a self-contained piece of code. It's easy to write unit test for a component.

Component and Pipeline

A machine learning pipeline is the workflow for a full machine learning task.

Components are the building blocks of a machine learning pipeline. When you're thinking of a component, it must be under the context of pipeline.

To build components, the first thing is to define the machine learning pipeline. This requires breaking down the full machine learning task into a multi-step workflow. Each step is a component. For example, considering a simple machine learning task of using historical data to train a sales forecasting model, you may want to build a sequential workflow with data processing, model training, and model evaluation steps. For complex tasks, you may want to further break down. For example, split one single data processing step into data ingestion, data cleaning, data pre-processing, and feature engineering steps.

Once the steps in the workflow are defined, the next thing is to specify how each step is connected in the pipeline. For example, to connect your data processing step and model training step, you may want to define a data processing component to output a folder that contains the processed data. A training component takes a folder as input and outputs a folder that contains the trained model. These inputs and outputs definition will become part of your component interface definition.

Now, it's time to develop the code of executing a step. You can use your preferred languages (python, R, etc.). The code must be able to be executed by a shell command. During the development, you may want to add a few inputs to control how this step is going to be executed. For example, for a training step, you may like to add learning rate, number of epochs as the inputs to control the training. These additional inputs plus the inputs and outputs required to connect with other steps are the interface of the component. The argument of a shell command is used to pass inputs and outputs to the code. The environment to execute the command and the code needs to be specified. The environment could be a curated Azure Machine Learning environment, a docker image or a conda environment.

Finally, you can package everything including code, cmd, environment, input, outputs, metadata together into a component. Then connects these components together to build pipelines for your machine learning workflow. One component can be used in multiple pipelines.

To learn more about how to build a component, see:

- How to [build a component using Azure Machine Learning CLI v2](#).
- How to [build a component using Azure Machine Learning SDK v2](#).

Next steps

- [Define component with the Azure Machine Learning CLI v2](#).
- [Define component with the Azure Machine Learning SDK v2](#).
- [Define component with Designer](#).
- [Component CLI v2 YAML reference](#).
- [What is Azure Machine Learning Pipeline?](#).
- Try out [CLI v2 component example ↗](#).
- Try out [Python SDK v2 component example ↗](#).

Additional resources

Documentation

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[How to do hyperparameter sweep in pipeline - Azure Machine Learning](#)

How to use sweep to do hyperparameter tuning in Azure Machine Learning pipeline using CLI v2 and Python SDK

[Upgrade model management to SDK v2 - Azure Machine Learning](#)

Upgrade model management from v1 to v2 of Azure Machine Learning SDK

[azureml.data package - Azure Machine Learning Python](#)

Contains modules supporting data representation for Datastore and Dataset in Azure Machine Learning. This package contains core functionality supporting Datastore and Dataset classes in the core package. Datastore objects contain connection information to Azure storage services that can...

[Tutorial: Upload data and train a model \(SDK v1\) - Azure Machine Learning](#)

How to upload and use your own data in a remote training job, with SDK v1. This is part 3 of a three-part getting-started series.

[Upgrade parallel run step to SDK v2 - Azure Machine Learning](#)

Upgrade parallel run step from v1 to v2 of Azure Machine Learning SDK

[azureml.pipeline.core.PublishedPipeline class - Azure Machine Learning Python](#)

Represents a Pipeline to be submitted without the Python code which constructed it. In addition, a PublishedPipeline can be used to resubmit a Pipeline with different PipelineParameter values and inputs.

[Show 5 more](#)

Training

Learning paths and modules

[Orchestrate machine learning with pipelines - Training](#)

Orchestrate machine learning with pipelines

Git integration for Azure Machine Learning

Article • 11/20/2022 • 6 minutes to read

[Git](#) is a popular version control system that allows you to share and collaborate on your projects.

Azure Machine Learning fully supports Git repositories for tracking work - you can clone repositories directly onto your shared workspace file system, use Git on your local workstation, or use Git from a CI/CD pipeline.

When submitting a job to Azure Machine Learning, if source files are stored in a local git repository then information about the repo is tracked as part of the training process.

Since Azure Machine Learning tracks information from a local git repo, it isn't tied to any specific central repository. Your repository can be cloned from GitHub, GitLab, Bitbucket, Azure DevOps, or any other git-compatible service.

💡 Tip

Use Visual Studio Code to interact with Git through a graphical user interface. To connect to an Azure Machine Learning remote compute instance using Visual Studio Code, see [Connect to an Azure Machine Learning compute instance in Visual Studio Code \(preview\)](#)

For more information on Visual Studio Code version control features, see [Using Version Control in VS Code](#) and [Working with GitHub in VS Code](#).

Clone Git repositories into your workspace file system

Azure Machine Learning provides a shared file system for all users in the workspace. To clone a Git repository into this file share, we recommend that you create a compute instance & [open a terminal](#). Once the terminal is opened, you have access to a full Git client and can clone and work with Git via the Git CLI experience.

We recommend that you clone the repository into your user directory so that others will not make collisions directly on your working branch.

Tip

There is a performance difference between cloning to the local file system of the compute instance or cloning to the mounted filesystem (mounted as the `~/cloudfiles/code` directory). In general, cloning to the local filesystem will have better performance than to the mounted filesystem. However, the local filesystem is lost if you delete and recreate the compute instance. The mounted filesystem is kept if you delete and recreate the compute instance.

You can clone any Git repository you can authenticate to (GitHub, Azure Repos, BitBucket, etc.)

For more information about cloning, see the guide on [how to use Git CLI](#).

Authenticate your Git Account with SSH

Generate a new SSH key

1. [Open the terminal window](#) in the Azure Machine Learning Notebook Tab.
2. Paste the text below, substituting in your email address.

```
Bash
```

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private rsa key pair.
```

3. When you're prompted to "Enter a file in which to save the key" press Enter. This accepts the default file location.
4. Verify that the default location is '/home/azureuser/.ssh' and press enter.
Otherwise specify the location '/home/azureuser/.ssh'.

Tip

Make sure the SSH key is saved in '/home/azureuser/.ssh'. This file is saved on the compute instance is only accessible by the owner of the Compute Instance

```
> Enter a file in which to save the key (/home/azureuser/.ssh/id_rsa):  
[Press enter]
```

5. At the prompt, type a secure passphrase. We recommend you add a passphrase to your SSH key for added security

```
> Enter passphrase (empty for no passphrase): [Type a passphrase]  
> Enter same passphrase again: [Type passphrase again]
```

Add the public key to Git Account

1. In your terminal window, copy the contents of your public key file. If you renamed the key, replace id_rsa.pub with the public key file name.

```
Bash
```

```
cat ~/.ssh/id_rsa.pub
```

💡 Tip

Copy and Paste in Terminal

- Windows: `Ctrl-Insert` to copy and use `Ctrl-Shift-v` or `Shift-Insert` to paste.
- Mac OS: `Cmd-c` to copy and `Cmd-v` to paste.
- FireFox/IE may not support clipboard permissions properly.

2. Select and copy the SSH key output to your clipboard.
3. Next, follow the steps to add the SSH key to your preferred account type:

- [GitHub ↗](#)
- [GitLab ↗](#)
- [Azure DevOps Start at Step 2.](#)

- BitBucket[↗]. Follow Step 4.

Clone the Git repository with SSH

1. Copy the SSH Git clone URL from the Git repo.
2. Paste the url into the `git clone` command below, to use your SSH Git repo URL.
This will look something like:

```
Bash
```

```
git clone git@example.com:GitUser/azureml-example.git
Cloning into 'azureml-example'...
```

You will see a response like:

```
Bash
```

```
The authenticity of host 'example.com (192.30.255.112)' can't be
established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of
known hosts.
```

SSH may display the server's SSH fingerprint and ask you to verify it. You should verify that the displayed fingerprint matches one of the fingerprints in the SSH public keys page.

SSH displays this fingerprint when it connects to an unknown host to protect you from [man-in-the-middle attacks](#). Once you accept the host's fingerprint, SSH will not prompt you again unless the fingerprint changes.

3. When you are asked if you want to continue connecting, type `yes`. Git will clone the repo and set up the origin remote to connect with SSH for future Git commands.

Track code that comes from Git repositories

When you submit a training job from the Python SDK or Machine Learning CLI, the files needed to train the model are uploaded to your workspace. If the `git` command is available on your development environment, the upload process uses it to check if the files are stored in a git repository. If so, then information from your git repository is also

uploaded as part of the training job. This information is stored in the following properties for the training job:

Property	Git command used to get the value	Description
<code>azureml.git.repository_uri</code>	<code>git ls-remote --get-url</code>	The URI that your repository was cloned from.
<code>mlflow.source.git.repoURL</code>	<code>git ls-remote --get-url</code>	The URI that your repository was cloned from.
<code>azureml.git.branch</code>	<code>git symbolic-ref --short HEAD</code>	The active branch when the job was submitted.
<code>mlflow.source.git.branch</code>	<code>git symbolic-ref --short HEAD</code>	The active branch when the job was submitted.
<code>azureml.git.commit</code>	<code>git rev-parse HEAD</code>	The commit hash of the code that was submitted for the job.
<code>mlflow.source.git.commit</code>	<code>git rev-parse HEAD</code>	The commit hash of the code that was submitted for the job.
<code>azureml.git.dirty</code>	<code>git status --porcelain .</code>	<code>True</code> , if the branch/commit is dirty; otherwise, <code>false</code> .

This information is sent for jobs that use an estimator, machine learning pipeline, or script run.

If your training files are not located in a git repository on your development environment, or the `git` command is not available, then no git-related information is tracked.

💡 Tip

To check if the git command is available on your development environment, open a shell session, command prompt, PowerShell or other command line interface and type the following command:

```
git --version
```

If installed, and in the path, you receive a response similar to `git version 2.4.1`. For more information on installing git on your development environment, see the

Git website [↗](#).

View the logged information

The git information is stored in the properties for a training job. You can view this information using the Azure portal or Python SDK.

Azure portal

1. From the [studio portal](#) [↗](#), select your workspace.
2. Select **Jobs**, and then select one of your experiments.
3. Select one of the jobs from the **Display name** column.
4. Select **Outputs + logs**, and then expand the **logs** and **azureml** entries. Select the link that begins with `###_azure`.

The logged information contains text similar to the following JSON:

JSON

```
"properties": {  
    "_azureml.ComputeTargetType": "batchai",  
    "ContentSnapshotId": "5ca66406-cbac-4d7d-bc95-f5a51dd3e57e",  
    "azureml.git.repository_uri":  
        "git@github.com:azure/machinelearningnotebooks",  
        "mlflow.source.git.repoURL":  
    "git@github.com:azure/machinelearningnotebooks",  
        "azureml.git.branch": "master",  
        "mlflow.source.git.branch": "master",  
        "azureml.git.commit": "4d2b93784676893f8e346d5f0b9fb894a9cf0742",  
        "mlflow.source.git.commit": "4d2b93784676893f8e346d5f0b9fb894a9cf0742",  
        "azureml.git.dirty": "True",  
        "AzureML.DerivedImageName":  
    "azureml/azureml_9d3568242c6bfef9631879915768deaf",  
    "ProcessInfoFile": "azureml-logs/process_info.json",  
    "ProcessStatusFile": "azureml-logs/process_status.json"  
}
```

View properties

After submitting a training run, a **Job** object is returned. The `properties` attribute of this object contains the logged git information. For example, the following code retrieves the commit hash:

Python SDK

APPLIES TO:  Python SDK azure-ai-ml v2 (current) 

Python

```
job.properties["azureml.git.commit"]
```

Next steps

- Access a compute instance terminal in your workspace

What is Responsible AI?

Article • 11/09/2022 • 7 minutes to read

APPLIES TO:  Azure CLI ml extension v1  Python SDK azureml v1

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (current) ↗

Responsible Artificial Intelligence (Responsible AI) is an approach to developing, assessing, and deploying AI systems in a safe, trustworthy, and ethical way. AI systems are the product of many decisions made by those who develop and deploy them. From system purpose to how people interact with AI systems, Responsible AI can help proactively guide these decisions toward more beneficial and equitable outcomes. That means keeping people and their goals at the center of system design decisions and respecting enduring values like fairness, reliability, and transparency.

Microsoft has developed a [Responsible AI Standard](#) ↗. It's a framework for building AI systems according to six principles: fairness, reliability and safety, privacy and security, inclusiveness, transparency, and accountability. For Microsoft, these principles are the cornerstone of a responsible and trustworthy approach to AI, especially as intelligent technology becomes more prevalent in products and services that people use every day.

This article demonstrates how Azure Machine Learning supports tools for enabling developers and data scientists to implement and operationalize the six principles.



Fairness and inclusiveness

AI systems should treat everyone fairly and avoid affecting similarly situated groups of people in different ways. For example, when AI systems provide guidance on medical treatment, loan applications, or employment, they should make the same recommendations to everyone who has similar symptoms, financial circumstances, or professional qualifications.

Fairness and inclusiveness in Azure Machine Learning: The [fairness assessment](#) component of the [Responsible AI dashboard](#) enables data scientists and developers to assess model fairness across sensitive groups defined in terms of gender, ethnicity, age, and other characteristics.

Reliability and safety

To build trust, it's critical that AI systems operate reliably, safely, and consistently. These systems should be able to operate as they were originally designed, respond safely to unanticipated conditions, and resist harmful manipulation. How they behave and the variety of conditions they can handle reflect the range of situations and circumstances that developers anticipated during design and testing.

Reliability and safety in Azure Machine Learning: The [error analysis](#) component of the [Responsible AI dashboard](#) enables data scientists and developers to:

- Get a deep understanding of how failure is distributed for a model.
- Identify cohorts (subsets) of data with a higher error rate than the overall benchmark.

These discrepancies might occur when the system or model underperforms for specific demographic groups or for infrequently observed input conditions in the training data.

Transparency

When AI systems help inform decisions that have tremendous impacts on people's lives, it's critical that people understand how those decisions were made. For example, a bank might use an AI system to decide whether a person is creditworthy. A company might use an AI system to determine the most qualified candidates to hire.

A crucial part of transparency is *interpretability*: the useful explanation of the behavior of AI systems and their components. Improving interpretability requires stakeholders to comprehend how and why AI systems function the way they do. The stakeholders can then identify potential performance issues, fairness issues, exclusionary practices, or unintended outcomes.

Transparency in Azure Machine Learning: The [model interpretability](#) and [counterfactual what-if](#) components of the [Responsible AI dashboard](#) enable data scientists and developers to generate human-understandable descriptions of the predictions of a model.

The model interpretability component provides multiple views into a model's behavior:

- *Global explanations.* For example, what features affect the overall behavior of a loan allocation model?
- *Local explanations.* For example, why was a customer's loan application approved or rejected?
- *Model explanations for a selected cohort of data points.* For example, what features affect the overall behavior of a loan allocation model for low-income applicants?

The counterfactual what-if component enables understanding and debugging a machine learning model in terms of how it reacts to feature changes and perturbations.

Azure Machine Learning also supports a [Responsible AI scorecard](#). The scorecard is a customizable PDF report that developers can easily configure, generate, download, and share with their technical and non-technical stakeholders to educate them about their datasets and models health, achieve compliance, and build trust. This scorecard can also be used in audit reviews to uncover the characteristics of machine learning models.

Privacy and security

As AI becomes more prevalent, protecting privacy and securing personal and business information are becoming more important and complex. With AI, privacy and data security require close attention because access to data is essential for AI systems to make accurate and informed predictions and decisions about people. AI systems must comply with privacy laws that:

- Require transparency about the collection, use, and storage of data.
- Mandate that consumers have appropriate controls to choose how their data is used.

Privacy and security in Azure Machine Learning: Azure Machine Learning enables administrators and developers to [create a secure configuration that complies](#) with their companies' policies. With Azure Machine Learning and the Azure platform, users can:

- Restrict access to resources and operations by user account or group.
- Restrict incoming and outgoing network communications.
- Encrypt data in transit and at rest.
- Scan for vulnerabilities.

- Apply and audit configuration policies.

Microsoft has also created two open-source packages that can enable further implementation of privacy and security principles:

- [SmartNoise](#): Differential privacy is a set of systems and practices that help keep the data of individuals safe and private. In machine learning solutions, differential privacy might be required for regulatory compliance. SmartNoise is an open-source project (co-developed by Microsoft) that contains components for building differentially private systems that are global.
- [Counterfit](#): Counterfit is an open-source project that comprises a command-line tool and generic automation layer to allow developers to simulate cyberattacks against AI systems. Anyone can download the tool and deploy it through Azure Cloud Shell to run in a browser, or deploy it locally in an Anaconda Python environment. It can assess AI models hosted in various cloud environments, on-premises, or in the edge. The tool is agnostic to AI models and supports various data types, including text, images, or generic input.

Accountability

The people who design and deploy AI systems must be accountable for how their systems operate. Organizations should draw upon industry standards to develop accountability norms. These norms can ensure that AI systems aren't the final authority on any decision that affects people's lives. They can also ensure that humans maintain meaningful control over otherwise highly autonomous AI systems.

Accountability in Azure Machine Learning: [Machine learning operations \(MLOps\)](#) is based on DevOps principles and practices that increase the efficiency of AI workflows. Azure Machine Learning provides the following MLOps capabilities for better accountability of your AI systems:

- Register, package, and deploy models from anywhere. You can also track the associated metadata that's required to use the model.
- Capture the governance data for the end-to-end machine learning lifecycle. The logged lineage information can include who is publishing models, why changes were made, and when models were deployed or used in production.
- Notify and alert on events in the machine learning lifecycle. Examples include experiment completion, model registration, model deployment, and data drift detection.
- Monitor applications for operational issues and issues related to machine learning. Compare model inputs between training and inference, explore model-specific

metrics, and provide monitoring and alerts on your machine learning infrastructure.

Besides the MLOps capabilities, the [Responsible AI scorecard](#) in Azure Machine Learning creates accountability by enabling cross-stakeholder communications. The scorecard also creates accountability by empowering developers to configure, download, and share their model health insights with their technical and non-technical stakeholders about AI data and model health. Sharing these insights can help build trust.

The machine learning platform also enables decision-making by informing business decisions through:

- Data-driven insights, to help stakeholders understand causal treatment effects on an outcome, by using historical data only. For example, "How would a medicine affect a patient's blood pressure?" These insights are provided through the [causal inference](#) component of the [Responsible AI dashboard](#).
- Model-driven insights, to answer users' questions (such as "What can I do to get a different outcome from your AI next time?") so they can take action. Such insights are provided to data scientists through the [counterfactual what-if](#) component of the [Responsible AI dashboard](#).

Next steps

- For more information on how to implement Responsible AI in Azure Machine Learning, see [Responsible AI dashboard](#).
- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in your Responsible AI dashboard.
- Learn about the [Responsible AI Standard](#) ↗ for building AI systems according to six key principles.

Assess AI systems by using the Responsible AI dashboard

Article • 02/22/2023 • 11 minutes to read

Implementing Responsible AI in practice requires rigorous engineering. But rigorous engineering can be tedious, manual, and time-consuming without the right tooling and infrastructure.

The Responsible AI dashboard provides a single interface to help you implement Responsible AI in practice effectively and efficiently. It brings together several mature Responsible AI tools in the areas of:

- [Model performance and fairness assessment ↗](#)
- Data exploration
- [Machine learning interpretability ↗](#)
- [Error analysis ↗](#)
- [Counterfactual analysis and perturbations ↗](#)
- [Causal inference ↗](#)

The dashboard offers a holistic assessment and debugging of models so you can make informed data-driven decisions. Having access to all of these tools in one interface empowers you to:

- Evaluate and debug your machine learning models by identifying model errors and fairness issues, diagnosing why those errors are happening, and informing your mitigation steps.
- Boost your data-driven decision-making abilities by addressing questions such as:
"What is the minimum change that users can apply to their features to get a different outcome from the model?"
"What is the causal effect of reducing or increasing a feature (for example, red meat consumption) on a real-world outcome (for example, diabetes progression)?"

You can customize the dashboard to include only the subset of tools that are relevant to your use case.

The Responsible AI dashboard is accompanied by a [PDF scorecard](#). The scorecard enables you to export Responsible AI metadata and insights into your data and models. You can then share them offline with the product and compliance stakeholders.

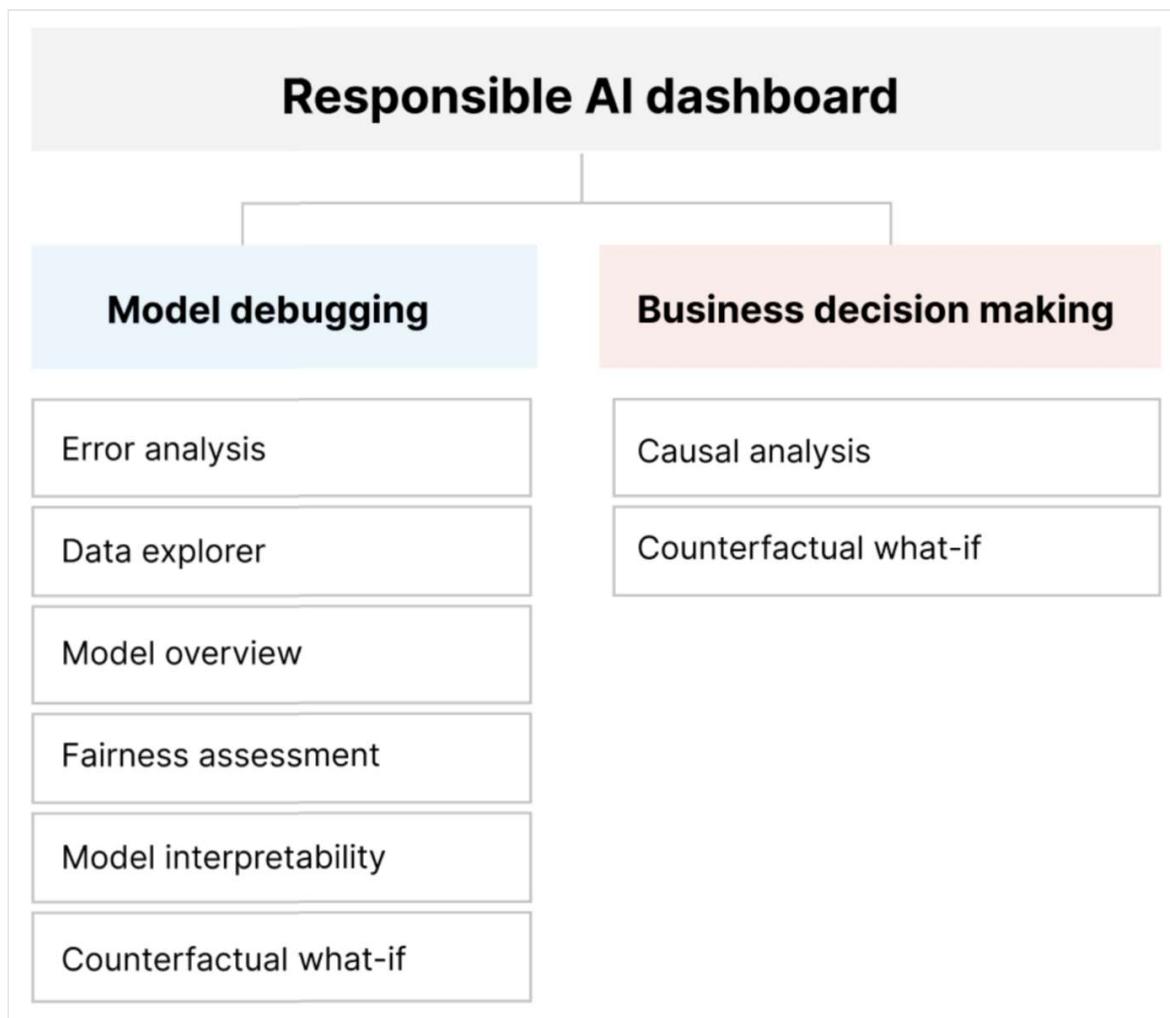
Responsible AI dashboard components

The Responsible AI dashboard brings together, in a comprehensive view, various new and pre-existing tools. The dashboard integrates these tools with [Azure Machine Learning CLI v2](#), [Azure Machine Learning Python SDK v2](#), and [Azure Machine Learning studio](#). The tools include:

- [Data analysis](#), to understand and explore your dataset distributions and statistics.
- [Model overview and fairness assessment](#), to evaluate the performance of your model and evaluate your model's group fairness issues (how your model's predictions affect diverse groups of people).
- [Error analysis](#), to view and understand how errors are distributed in your dataset.
- [Model interpretability](#) (importance values for aggregate and individual features), to understand your model's predictions and how those overall and individual predictions are made.
- [Counterfactual what-if](#), to observe how feature perturbations would affect your model predictions while providing the closest data points with opposing or different model predictions.
- [Causal analysis](#), to use historical data to view the causal effects of treatment features on real-world outcomes.

Together, these tools will help you debug machine learning models, while informing your data-driven and model-driven business decisions. The following diagram shows how you can incorporate them into your AI lifecycle to improve your models and get solid data insights.

Responsible AI dashboard



Model debugging

Assessing and debugging machine learning models is critical for model reliability, interpretability, fairness, and compliance. It helps determine how and why AI systems behave the way they do. You can then use this knowledge to improve model performance. Conceptually, model debugging consists of three stages:

1. **Identify**, to understand and recognize model errors and/or fairness issues by addressing the following questions:

"What kinds of errors does my model have?"

"In what areas are errors most prevalent?"

2. **Diagnose**, to explore the reasons behind the identified errors by addressing:

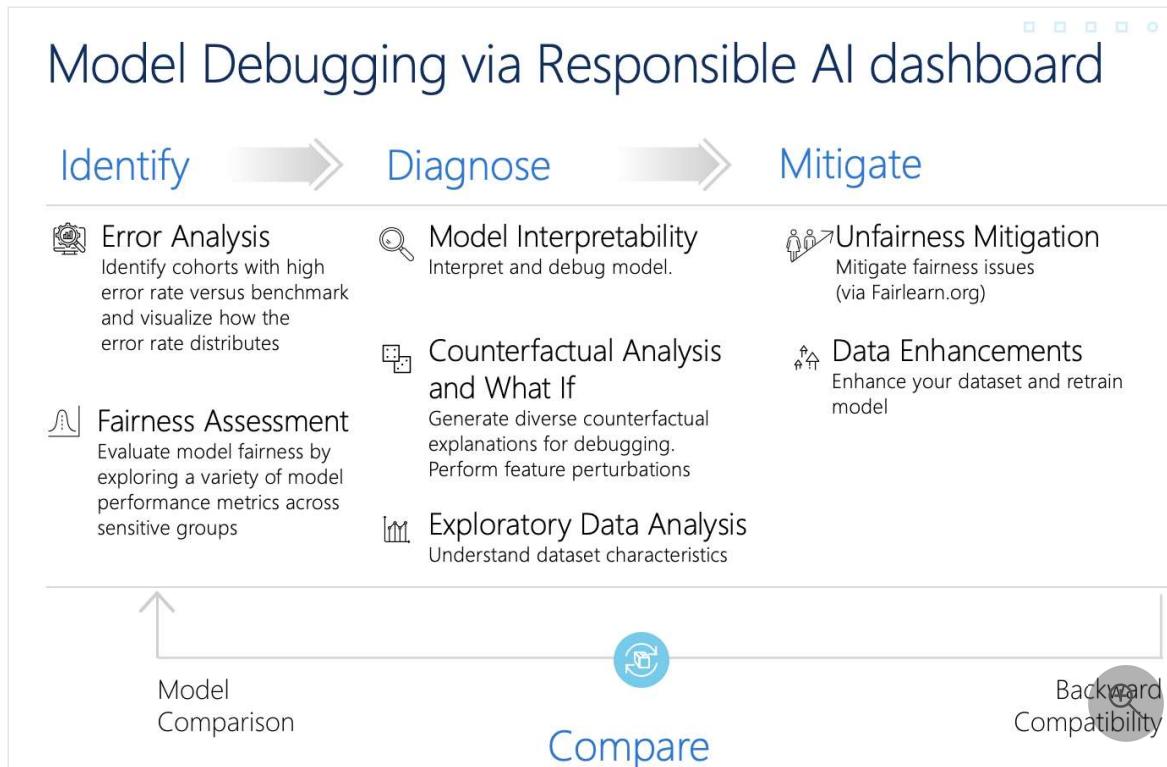
"What are the causes of these errors?"

"Where should I focus my resources to improve my model?"

3. **Mitigate**, to use the identification and diagnosis insights from previous stages to take targeted mitigation steps and address questions such as:

"How can I improve my model?"

"What social or technical solutions exist for these issues?"



The following table describes when to use Responsible AI dashboard components to support model debugging:

Stage	Component	Description
Identify	Error analysis	<p>The error analysis component helps you get a deeper understanding of model failure distribution and quickly identify erroneous cohorts (subgroups) of data.</p> <p>The capabilities of this component in the dashboard come from the Error Analysis package.</p>
Identify	Fairness analysis	<p>The fairness component defines groups in terms of sensitive attributes such as sex, race, and age. It then assesses how your model predictions affect these groups and how you can mitigate disparities. It evaluates the performance of your model by exploring the distribution of your prediction values and the values of your model performance metrics across the groups.</p> <p>The capabilities of this component in the dashboard come from the Fairlearn package.</p>

Stage	Component	Description
Identify	Model overview	The model overview component aggregates model assessment metrics in a high-level view of model prediction distribution for better investigation of its performance. This component also enables group fairness assessment by highlighting the breakdown of model performance across sensitive groups.
Diagnose	Data analysis	Data analysis visualizes datasets based on predicted and actual outcomes, error groups, and specific features. You can then identify issues of overrepresentation and underrepresentation, along with seeing how data is clustered in the dataset.
Diagnose	Model interpretability	<p>The interpretability component generates human-understandable explanations of the predictions of a machine learning model. It provides multiple views into a model's behavior:</p> <ul style="list-style-type: none"> - Global explanations (for example, which features affect the overall behavior of a loan allocation model) - Local explanations (for example, why an applicant's loan application was approved or rejected) <p>The capabilities of this component in the dashboard come from the InterpretML package.</p>
Diagnose	Counterfactual analysis and what-if	<p>This component consists of two functionalities for better error diagnosis:</p> <ul style="list-style-type: none"> - Generating a set of examples in which minimal changes to a particular point alter the model's prediction. That is, the examples show the closest data points with opposite model predictions. - Enabling interactive and custom what-if perturbations for individual data points to understand how the model reacts to feature changes. <p>The capabilities of this component in the dashboard come from the DiCE package.</p>

Mitigation steps are available via standalone tools such as [Fairlearn](#). For more information, see the [unfairness mitigation algorithms](#).

Responsible decision-making

Decision-making is one of the biggest promises of machine learning. The Responsible AI dashboard can help you make informed business decisions through:

- Data-driven insights, to further understand causal treatment effects on an outcome by using historical data only. For example:

"How would a medicine affect a patient's blood pressure?"

"How would providing promotional values to certain customers affect revenue?"

These insights are provided through the [causal inference](#) component of the dashboard.

- Model-driven insights, to answer users' questions (such as "What can I do to get a different outcome from your AI next time?") so they can take action. These insights are provided to data scientists through the [counterfactual what-if](#) component.

Decision Making via Responsible AI dashboard

Understand data

Inform Actions

 Exploratory-Data-Analysis
Understand dataset characteristics

 Causal Inference
Understand the causal impact of your features on real-world outcomes

 Counterfactual Analysis
Generate diverse counterfactual explanations for informing end users

Exploratory data analysis, causal inference, and counterfactual analysis capabilities can help you make informed model-driven and data-driven decisions responsibly.

These components of the Responsible AI dashboard support responsible decision-making:

- **Data analysis:** You can reuse the data analysis component here to understand data distributions and to identify overrepresentation and underrepresentation. Data exploration is a critical part of decision making, because it isn't feasible to make informed decisions about a cohort that's underrepresented in the data.
- **Causal inference:** The causal inference component estimates how a real-world outcome changes in the presence of an intervention. It also helps construct promising interventions by simulating feature responses to various interventions and creating rules to determine which population cohorts would benefit from a particular intervention. Collectively, these functionalities allow you to apply new policies and effect real-world change.

The capabilities of this component come from the [EconML](#) package, which estimates heterogeneous treatment effects from observational data via machine learning.

- **Counterfactual analysis:** You can reuse the counterfactual analysis component here to generate minimum changes applied to a data point's features that lead to opposite model predictions. For example: Taylor would have obtained the loan approval from the AI if they earned \$10,000 more in annual income and had two fewer credit cards open.

Providing this information to users informs their perspective. It educates them on how they can take action to get the desired outcome from the AI in the future.

The capabilities of this component come from the [DiCE](#) package.

Reasons for using the Responsible AI dashboard

Although progress has been made on individual tools for specific areas of Responsible AI, data scientists often need to use various tools to holistically evaluate their models and data. For example: they might have to use model interpretability and fairness assessment together.

If data scientists discover a fairness issue with one tool, they then need to jump to a different tool to understand what data or model factors lie at the root of the issue before taking any steps on mitigation. The following factors further complicate this challenging process:

- There's no central location to discover and learn about the tools, extending the time it takes to research and learn new techniques.
- The different tools don't communicate with each other. Data scientists must wrangle the datasets, models, and other metadata as they pass them between the tools.
- The metrics and visualizations aren't easily comparable, and the results are hard to share.

The Responsible AI dashboard challenges this status quo. It's a comprehensive yet customizable tool that brings together fragmented experiences in one place. It enables you to seamlessly onboard to a single customizable framework for model debugging and data-driven decision-making.

By using the Responsible AI dashboard, you can create dataset cohorts, pass those cohorts to all of the supported components, and observe your model health for your identified cohorts. You can further compare insights from all supported components across a variety of prebuilt cohorts to perform disaggregated analysis and find the blind spots of your model.

When you're ready to share those insights with other stakeholders, you can extract them easily by using the [Responsible AI PDF scorecard](#). Attach the PDF report to your compliance reports, or share it with colleagues to build trust and get their approval.

Ways to customize the Responsible AI dashboard

The Responsible AI dashboard's strength lies in its customizability. It empowers users to design tailored, end-to-end model debugging and decision-making workflows that address their particular needs.

Need some inspiration? Here are some examples of how the dashboard's components can be put together to analyze scenarios in diverse ways:

Responsible AI dashboard flow	Use case
Model overview > error analysis > data analysis	To identify model errors and diagnose them by understanding the underlying data distribution
Model overview > fairness assessment > data analysis	To identify model fairness issues and diagnose them by understanding the underlying data distribution
Model overview > error analysis > counterfactuals analysis and what-if	To diagnose errors in individual instances with counterfactual analysis (minimum change to lead to a different model prediction)
Model overview > data analysis	To understand the root cause of errors and fairness issues introduced via data imbalances or lack of representation of a particular data cohort
Model overview > interpretability	To diagnose model errors through understanding how the model has made its predictions
Data analysis > causal inference	To distinguish between correlations and causations in the data or decide the best treatments to apply to get a positive outcome
Interpretability > causal inference	To learn whether the factors that the model has used for prediction-making have any causal effect on the real-world outcome
Data analysis > counterfactuals analysis and what-if	To address customers' questions about what they can do next time to get a different outcome from an AI system

People who should use the Responsible AI dashboard

The following people can use the Responsible AI dashboard, and its corresponding [Responsible AI scorecard](#), to build trust with AI systems:

- Machine learning professionals and data scientists who are interested in debugging and improving their machine learning models before deployment
- Machine learning professionals and data scientists who are interested in sharing their model health records with product managers and business stakeholders to build trust and receive deployment permissions
- Product managers and business stakeholders who are reviewing machine learning models before deployment
- Risk officers who are reviewing machine learning models to understand fairness and reliability issues
- Providers of AI solutions who want to explain model decisions to users or help them improve the outcome
- Professionals in heavily regulated spaces who need to review machine learning models with regulators and auditors

Supported scenarios and limitations

- The Responsible AI dashboard currently supports regression and classification (binary and multi-class) models trained on tabular structured data.
- The Responsible AI dashboard currently supports MLflow models that are registered in Azure Machine Learning with a sklearn (scikit-learn) flavor only. The scikit-learn models should implement `predict()/predict_proba()` methods, or the model should be wrapped within a class that implements `predict()/predict_proba()` methods. The models must be loadable in the component environment and must be pickleable.
- The Responsible AI dashboard currently visualizes up to 5K of your data points on the dashboard UI. You should downsample your dataset to 5K or less before passing it to the dashboard.
- The dataset inputs to the Responsible AI dashboard must be pandas DataFrames in Parquet format. NumPy and SciPy sparse data is currently not supported.
- The Responsible AI dashboard currently supports numeric or categorical features. For categorical features, the user has to explicitly specify the feature names.
- The Responsible AI dashboard currently doesn't support datasets with more than 10K columns.
- The Responsible AI dashboard currently doesn't support AutoML MLFlow model.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
 - Learn how to generate a [Responsible AI scorecard](#) based on the insights observed on the Responsible AI dashboard.
-

Additional resources

Documentation

[Generate a Responsible AI insights in the studio UI - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with no-code experience in the Azure Machine Learning studio UI.

[Share Responsible AI insights and make data-driven decisions with Azure Machine Learning Responsible AI scorecard - Azure Machine Learning](#)

Learn about how to use the Responsible AI scorecard to share responsible AI insights from your machine learning models and make data-driven decisions with non-technical and technical stakeholders.

[Generate a Responsible AI insights with YAML and Python - Azure Machine Learning](#)

Learn how to generate a Responsible AI insights with Python and YAML in Azure Machine Learning.

[Use the Responsible AI dashboard in Azure Machine Learning studio - Azure Machine Learning](#)

Learn how to use the various tools and visualization charts in the Responsible AI dashboard in Azure Machine Learning.

[Tutorial: AutoML- train object detection model \(v1\) - Azure Machine Learning](#)

Train an object detection model to identify if an image contains certain objects with automated ML and the Azure Machine Learning Python SDK automated ML. (v1)

[Use Responsible AI scorecard \(preview\) in Azure Machine Learning - Azure Machine Learning](#)

Share insights with non-technical business stakeholders by exporting a PDF Responsible AI scorecard from Azure Machine Learning.

[Prepare data for computer vision tasks - Azure Machine Learning](#)

Image data preparation for Azure Machine Learning automated ML to train computer vision models on classification, object detection, and segmentation

[Algorithm & component reference \(v2\) - Azure Machine Learning](#)

Learn about the Azure Machine Learning designer components that you can use to create your own machine learning projects. (v2)

[Show 5 more](#)

Training

Learning certificate

Microsoft Certified: Azure AI Engineer Associate - Certifications

Azure AI engineers build, manage, and deploy AI solutions that leverage Azure Cognitive Services and Azure Applied AI services.

Share Responsible AI insights using the Responsible AI scorecard (preview)

Article • 11/09/2022 • 2 minutes to read

Our Responsible AI dashboard is designed for machine learning professionals and data scientists to explore and evaluate model insights and inform their data-driven decisions. While it can help you implement Responsible AI practically in your machine learning lifecycle, there are some needs left unaddressed:

- There often exists a gap between the technical Responsible AI tools (designed for machine-learning professionals) and the ethical, regulatory, and business requirements that define the production environment.
- While an end-to-end machine learning life cycle includes both technical and non-technical stakeholders in the loop, there's little support to enable an effective multi-stakeholder alignment, helping technical experts get timely feedback and direction from the non-technical stakeholders.
- AI regulations make it essential to be able to share model and data insights with auditors and risk officers for auditability purposes.

One of the biggest benefits of using the Azure Machine Learning ecosystem is related to the archival of model and data insights in the Azure Machine Learning Run History (for quick reference in future). As a part of that infrastructure and to accompany machine learning models and their corresponding Responsible AI dashboards, we introduce the Responsible AI scorecard to empower ML professionals to generate and share their data and model health records easily.

Who should use a Responsible AI scorecard?

- If you're a data scientist or a machine learning professional, after training your model and generating its corresponding Responsible AI dashboard(s) for assessment and decision-making purposes, you can extract those learnings via our PDF scorecard and share the report easily with your technical and non-technical stakeholders to build trust and gain their approval for deployment.
- If you're a product manager, business leader, or an accountable stakeholder on an AI product, you can pass your desired model performance and fairness target values such as your target accuracy, target error rate, etc., to your data science team, asking them to generate this scorecard with respect to your identified target values and whether your model meets them. That can provide guidance into whether the model should be deployed or further improved.

Next steps

- Learn how to generate the Responsible AI dashboard and scorecard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Learn more about how the Responsible AI dashboard and scorecard in this [tech community blog post](#).

Model interpretability

Article • 11/09/2022 • 7 minutes to read

This article describes methods you can use for model interpretability in Azure Machine Learning.

Important

With the release of the Responsible AI dashboard, which includes model interpretability, we recommend that you migrate to the new experience, because the older SDK v1 preview model interpretability dashboard will no longer be actively maintained.

Why model interpretability is important to model debugging

When you're using machine learning models in ways that affect people's lives, it's critically important to understand what influences the behavior of models.

Interpretability helps answer questions in scenarios such as:

- Model debugging: Why did my model make this mistake? How can I improve my model?
- Human-AI collaboration: How can I understand and trust the model's decisions?
- Regulatory compliance: Does my model satisfy legal requirements?

The interpretability component of the [Responsible AI dashboard](#) contributes to the "diagnose" stage of the model lifecycle workflow by generating human-understandable descriptions of the predictions of a machine learning model. It provides multiple views into a model's behavior:

- Global explanations: For example, what features affect the overall behavior of a loan allocation model?
- Local explanations: For example, why was a customer's loan application approved or rejected?

You can also observe model explanations for a selected cohort as a subgroup of data points. This approach is valuable when, for example, you're assessing fairness in model predictions for individuals in a particular demographic group. The **Local explanation** tab of this component also represents a full data visualization, which is great for general

eyeballing of the data and looking at differences between correct and incorrect predictions of each cohort.

The capabilities of this component are founded by the [InterpretML](#) package, which generates model explanations.

Use interpretability when you need to:

- Determine how trustworthy your AI system's predictions are by understanding what features are most important for the predictions.
- Approach the debugging of your model by understanding it first and identifying whether the model is using healthy features or merely false correlations.
- Uncover potential sources of unfairness by understanding whether the model is basing predictions on sensitive features or on features that are highly correlated with them.
- Build user trust in your model's decisions by generating local explanations to illustrate their outcomes.
- Complete a regulatory audit of an AI system to validate models and monitor the impact of model decisions on humans.

How to interpret your model

In machine learning, *features* are the data fields you use to predict a target data point. For example, to predict credit risk, you might use data fields for age, account size, and account age. Here, age, account size, and account age are features. Feature importance tells you how each data field affects the model's predictions. For example, although you might use age heavily in the prediction, account size and account age might not affect the prediction values significantly. Through this process, data scientists can explain resulting predictions in ways that give stakeholders visibility into the model's most important features.

By using the classes and methods in the Responsible AI dashboard and by using SDK v2 and CLI v2, you can:

- Explain model prediction by generating feature-importance values for the entire model (global explanation) or individual data points (local explanation).
- Achieve model interpretability on real-world datasets at scale.
- Use an interactive visualization dashboard to discover patterns in your data and its explanations at training time.

By using the classes and methods in the SDK v1, you can:

- Explain model prediction by generating feature-importance values for the entire model or individual data points.
- Achieve model interpretability on real-world datasets at scale during training and inference.
- Use an interactive visualization dashboard to discover patterns in your data and its explanations at training time.

 **Note**

Model interpretability classes are made available through the SDK v1 package. For more information, see [Install SDK packages for Azure Machine Learning](#) and `azureml.interpret`.

Supported model interpretability techniques

The Responsible AI dashboard and `azureml-interpret` use the interpretability techniques that were developed in [Interpret-Community](#), an open-source Python package for training interpretable models and helping to explain opaque-box AI systems. Opaque-box models are those for which we have no information about their internal workings.

Interpret-Community serves as the host for the following supported explainers, and currently supports the interpretability techniques presented in the next sections.

Supported in Responsible AI dashboard in Python SDK v2 and CLI v2

Interpretability technique	Description	Type
Mimic Explainer (Global Surrogate) + SHAP tree	<p>Mimic Explainer is based on the idea of training global surrogate models to mimic opaque-box models. A global surrogate model is an intrinsically interpretable model that's trained to approximate the predictions of any opaque-box model as accurately as possible.</p> <p>Data scientists can interpret the surrogate model to draw conclusions about the opaque-box model. The Responsible AI dashboard uses LightGBM (<code>LGBMExplainableModel</code>), paired with the SHAP (SHapley Additive exPlanations) Tree Explainer, which is a specific explainer to trees and ensembles of trees. The combination of LightGBM and SHAP tree provides model-agnostic global and local explanations of your machine learning models.</p>	Model-agnostic

Supported in Python SDK v1

Interpretability technique	Description	Type
SHAP Tree Explainer	The SHAP Tree Explainer, which focuses on a polynomial, time-fast, SHAP value-estimation algorithm that's specific to <i>trees and ensembles of trees</i> .	Model-specific
SHAP Deep Explainer	Based on the explanation from SHAP, Deep Explainer is a "high-speed approximation algorithm for SHAP values in deep learning models that builds on a connection with DeepLIFT described in the SHAP NIPS paper . <i>TensorFlow</i> models and <i>Keras</i> models using the <i>TensorFlow</i> back end are supported (there's also preliminary support for PyTorch)."	Model-specific
SHAP Linear Explainer	The SHAP Linear Explainer computes SHAP values for a <i>linear model</i> , optionally accounting for inter-feature correlations.	Model-specific
SHAP Kernel Explainer	The SHAP Kernel Explainer uses a specially weighted local linear regression to estimate SHAP values for <i>any model</i> .	Model-agnostic
Mimic Explainer (Global Surrogate)	Mimic Explainer is based on the idea of training global surrogate models to mimic opaque-box models. A global surrogate model is an intrinsically interpretable model that's trained to approximate the predictions of <i>any opaque-box model</i> as accurately as possible. Data scientists can interpret the surrogate model to draw conclusions about the opaque-box model. You can use one of the following interpretable models as your surrogate model: LightGBM (<code>LGBMExplainableModel</code>), Linear Regression (<code>LinearExplainableModel</code>), Stochastic Gradient Descent explainable model (<code>SGDExplainableModel</code>), or Decision Tree (<code>DecisionTreeExplainableModel</code>).	Model-agnostic
Permutation Feature Importance Explainer	Permutation Feature Importance (PFI) is a technique used to explain classification and regression models that's inspired by Breiman's Random Forests paper (see section 10). At a high level, the way it works is by randomly shuffling data one feature at a time for the entire dataset and calculating how much the performance metric of interest changes. The larger the change, the more important that feature is. PFI can explain the overall behavior of <i>any underlying model</i> but doesn't explain individual predictions.	Model-agnostic

Besides the interpretability techniques described above, we support another SHAP-based explainer, called Tabular Explainer. Depending on the model, Tabular Explainer uses one of the supported SHAP explainers:

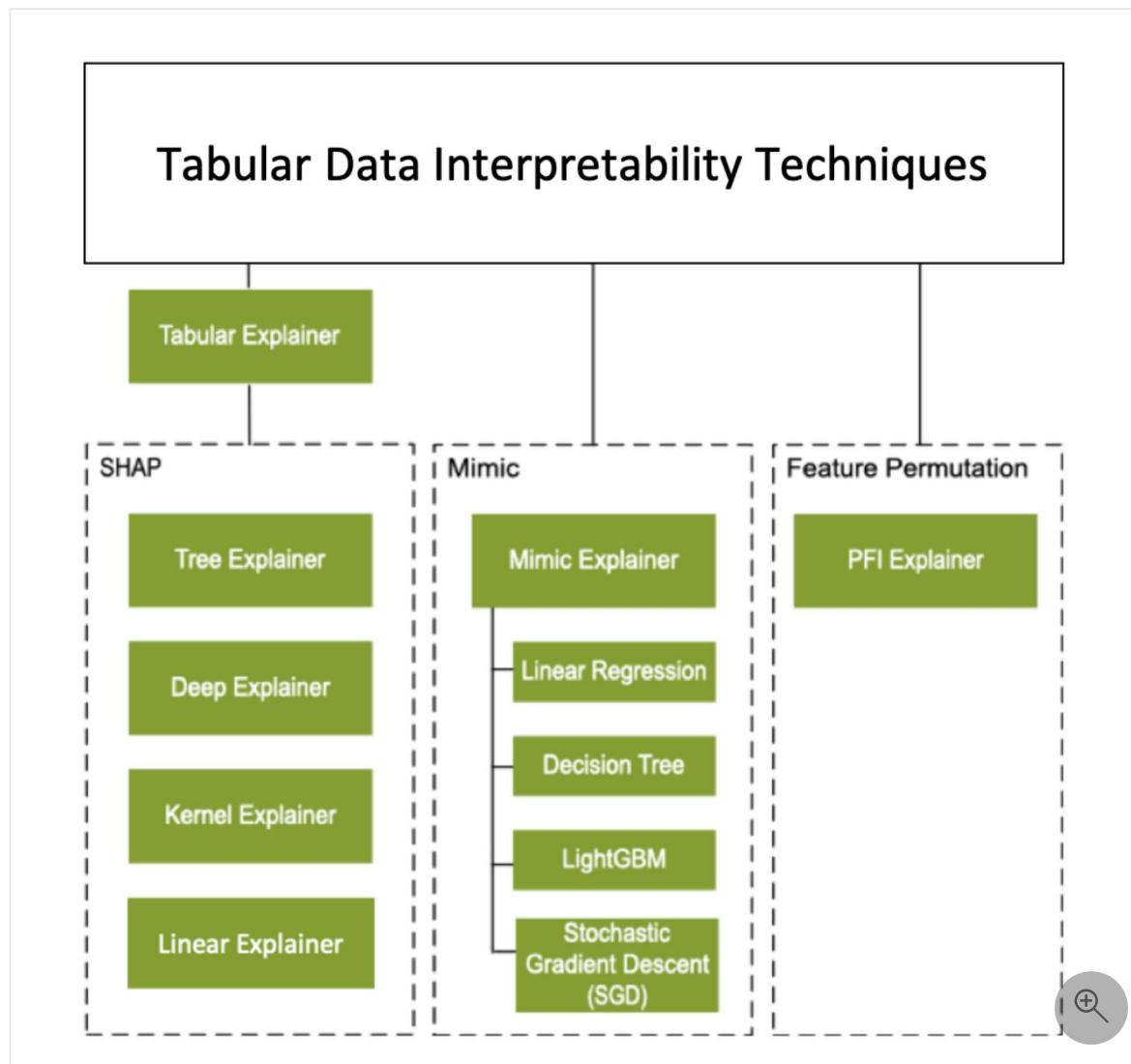
- Tree Explainer for all tree-based models
- Deep Explainer for deep neural network (DNN) models

- Linear Explainer for linear models
- Kernel Explainer for all other models

Tabular Explainer has also made significant feature and performance enhancements over the direct SHAP explainers:

- **Summarization of the initialization dataset:** When speed of explanation is most important, we summarize the initialization dataset and generate a small set of representative samples. This approach speeds up the generation of overall and individual feature importance values.
- **Sampling the evaluation data set:** If you pass in a large set of evaluation samples but don't actually need all of them to be evaluated, you can set the sampling parameter to `true` to speed up the calculation of overall model explanations.

The following diagram shows the current structure of supported explainers:



Supported machine learning models

The `azureml.interpret` package of the SDK supports models that are trained with the following dataset formats:

- `numpy.array`
- `pandas.DataFrame`
- `iml.datatypes.DenseData`
- `scipy.sparse.csr_matrix`

The explanation functions accept both models and pipelines as input. If a model is provided, it must implement the prediction function `predict` or `predict_proba` that conforms to the Scikit convention. If your model doesn't support this, you can wrap it in a function that generates the same outcome as `predict` or `predict_proba` in Scikit and use that wrapper function with the selected explainer.

If you provide a pipeline, the explanation function assumes that the running pipeline script returns a prediction. When you use this wrapping technique, `azureml.interpret` can support models that are trained via PyTorch, TensorFlow, and Keras deep learning frameworks as well as classic machine learning models.

Local and remote compute target

The `azureml.interpret` package is designed to work with both local and remote compute targets. If you run the package locally, the SDK functions won't contact any Azure services.

You can run the explanation remotely on Azure Machine Learning Compute and log the explanation info into the Azure Machine Learning Run History Service. After this information is logged, reports and visualizations from the explanation are readily available on Azure Machine Learning studio for analysis.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI v2 and SDK v2](#) or the [Azure Machine Learning studio UI](#).
- Explore the [supported interpretability visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.
- Learn how to enable [interpretability for automated machine learning models](#).

Model performance and fairness

Article • 11/09/2022 • 5 minutes to read

This article describes methods that you can use to understand your model performance and fairness in Azure Machine Learning.

What is machine learning fairness?

Artificial intelligence and machine learning systems can display unfair behavior. One way to define unfair behavior is by its harm, or its impact on people. AI systems can give rise to many types of harm. To learn more, see the [NeurIPS 2017 keynote by Kate Crawford](#).

Two common types of AI-caused harms are:

- **Harm of allocation:** An AI system extends or withholds opportunities, resources, or information for certain groups. Examples include hiring, school admissions, and lending, where a model might be better at picking good candidates among a specific group of people than among other groups.
- **Harm of quality-of-service:** An AI system doesn't work as well for one group of people as it does for another. For example, a voice recognition system might fail to work as well for women as it does for men.

To reduce unfair behavior in AI systems, you have to assess and mitigate these harms. The *model overview* component of the [Responsible AI dashboard](#) contributes to the identification stage of the model lifecycle by generating model performance metrics for your entire dataset and your identified cohorts of data. It generates these metrics across subgroups identified in terms of sensitive features or sensitive attributes.

Note

Fairness is a socio-technical challenge. Quantitative fairness metrics don't capture many aspects of fairness, such as justice and due process. Also, many quantitative fairness metrics can't all be satisfied simultaneously.

The goal of the Fairlearn open-source package is to enable humans to assess the impact and mitigation strategies. Ultimately, it's up to the humans who build AI and machine learning models to make trade-offs that are appropriate for their scenarios.

In this component of the Responsible AI dashboard, fairness is conceptualized through an approach known as *group fairness*. This approach asks: "Which groups of individuals are at risk for experiencing harm?" The term *sensitive features* suggests that the system designer should be sensitive to these features when assessing group fairness.

During the assessment phase, fairness is quantified through *disparity metrics*. These metrics can evaluate and compare model behavior across groups either as ratios or as differences. The Responsible AI dashboard supports two classes of disparity metrics:

- **Disparity in model performance:** These sets of metrics calculate the disparity (difference) in the values of the selected performance metric across subgroups of data. Here are a few examples:
 - Disparity in accuracy rate
 - Disparity in error rate
 - Disparity in precision
 - Disparity in recall
 - Disparity in mean absolute error (MAE)
- **Disparity in selection rate:** This metric contains the difference in selection rate (favorable prediction) among subgroups. An example of this is disparity in loan approval rate. Selection rate means the fraction of data points in each class classified as 1 (in binary classification) or distribution of prediction values (in regression).

The fairness assessment capabilities of this component come from the [Fairlearn](#) package. Fairlearn provides a collection of model fairness assessment metrics and unfairness mitigation algorithms.

Note

A fairness assessment is not a purely technical exercise. The Fairlearn open-source package can identify quantitative metrics to help you assess the fairness of a model, but it won't perform the assessment for you. You must perform a qualitative analysis to evaluate the fairness of your own models. The sensitive features noted earlier are an example of this kind of qualitative analysis.

Parity constraints for mitigating unfairness

After you understand your model's fairness issues, you can use the mitigation algorithms in the [Fairlearn](#) open-source package to mitigate those issues. These

algorithms support a set of constraints on the predictor's behavior called *parity constraints* or criteria.

Parity constraints require some aspects of the predictor's behavior to be comparable across the groups that sensitive features define (for example, different races). The mitigation algorithms in the Fairlearn open-source package use such parity constraints to mitigate the observed fairness issues.

ⓘ Note

The unfairness mitigation algorithms in the Fairlearn open-source package can provide suggested mitigation strategies to reduce unfairness in a machine learning model, but those strategies don't eliminate unfairness. Developers might need to consider other parity constraints or criteria for their machine learning models. Developers who use Azure Machine Learning must determine for themselves if the mitigation sufficiently reduces unfairness in their intended use and deployment of machine learning models.

The Fairlearn package supports the following types of parity constraints:

Parity constraint	Purpose	Machine learning task
Demographic parity	Mitigate allocation harms	Binary classification, regression
Equalized odds	Diagnose allocation and quality-of-service harms	Binary classification
Equal opportunity	Diagnose allocation and quality-of-service harms	Binary classification
Bounded group loss	Mitigate quality-of-service harms	Regression

Mitigation algorithms

The Fairlearn open-source package provides two types of unfairness mitigation algorithms:

- **Reduction:** These algorithms take a standard black-box machine learning estimator (for example, a LightGBM model) and generate a set of retrained models by using a sequence of reweighted training datasets.

For example, applicants of a certain gender might be upweighted or downweighted to retrain models and reduce disparities across gender groups. Users can then pick a model that provides the best trade-off between accuracy (or another performance metric) and disparity, based on their business rules and cost calculations.

- **Post-processing:** These algorithms take an existing classifier and a sensitive feature as input. They then derive a transformation of the classifier's prediction to enforce the specified fairness constraints. The biggest advantage of one post-processing algorithm, threshold optimization, is its simplicity and flexibility because it doesn't need to retrain the model.

Algorithm	Description	Machine learning task	Sensitive features	Supported parity constraints	Algorithm type
<code>ExponentiatedGradient</code>	Black-box approach to fair classification described in A Reductions Approach to Fair Classification .	Binary classification	Categorical	Demographic parity, equalized odds	Reduction
<code>GridSearch</code>	Black-box approach described in A Reductions Approach to Fair Classification .	Binary classification	Binary	Demographic parity, equalized odds	Reduction

Algorithm	Description	Machine learning task	Sensitive features	Supported parity constraints	Algorithm type
GridSearch	<p>Black-box approach that implements a grid-search variant of fair regression with the algorithm for bounded group loss described in Fair Regression: Quantitative Definitions and Reduction-based Algorithms ↗.</p>	Regression	Binary	Bounded group loss	Reduction
ThresholdOptimizer	<p>Postprocessing algorithm based on the paper Equality of Opportunity in Supervised Learning ↗. This technique takes as input an existing classifier and a sensitive feature. Then, it derives a monotone transformation of the classifier's prediction to enforce the specified parity constraints.</p>	Binary classification	Categorical	Demographic parity, equalized odds	Post-processing

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the supported model overview and fairness assessment visualizations of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.
- Learn how to use the components by checking out Fairlearn's [GitHub repository](#), [user guide](#), [examples](#), and [sample notebooks](#).

Make data-driven policies and influence decision-making

Article • 11/09/2022 • 3 minutes to read

Machine learning models are powerful in identifying patterns in data and making predictions. But they offer little support for estimating how the real-world outcome changes in the presence of an intervention.

Practitioners have become increasingly focused on using historical data to inform their future decisions and business interventions. For example, how would the revenue be affected if a corporation pursued a new pricing strategy? Would a new medication improve a patient's condition, all else equal?

The *causal inference* component of the [Responsible AI dashboard](#) addresses these questions by estimating the effect of a feature on an outcome of interest on average, across a population or a cohort, and on an individual level. It also helps construct promising interventions by simulating feature responses to various interventions and creating rules to determine which population cohorts would benefit from an intervention. Collectively, these functionalities allow decision-makers to apply new policies and effect real-world change.

The capabilities of this component come from the [EconML](#) package. It estimates heterogeneous treatment effects from observational data via the [double machine learning](#) technique.

Use causal inference when you need to:

- Identify the features that have the most direct effect on your outcome of interest.
- Decide what overall treatment policy to take to maximize real-world impact on an outcome of interest.
- Understand how individuals with certain feature values would respond to a particular treatment policy.

How are causal inference insights generated?

Note

Only historical data is required to generate causal insights. The causal effects computed based on the treatment features are purely a data property. So, a trained model is optional when you're computing the causal effects.

Double machine learning is a method for estimating heterogeneous treatment effects when all potential confounders/controls (factors that simultaneously had a direct effect on the treatment decision in the collected data and the observed outcome) are observed but either of the following problems exists:

- There are too many for classical statistical approaches to be applicable. That is, they're *high-dimensional*.
- Their effect on the treatment and outcome can't be satisfactorily modeled by parametric functions. That is, they're *non-parametric*.

You can use machine learning techniques to address both problems. For an example, see [Chernozhukov2016](#).

Double machine learning reduces the problem by first estimating two predictive tasks:

- Predicting the outcome from the controls
- Predicting the treatment from the controls

Then the method combines these two predictive models in a final-stage estimation to create a model of the heterogeneous treatment effect. This approach allows for arbitrary machine learning algorithms to be used for the two predictive tasks while maintaining many favorable statistical properties related to the final model. These properties include small mean squared error, asymptotic normality, and construction of confidence intervals.

What other tools does Microsoft provide for causal inference?

- [Project Azua](#) provides a novel framework that focuses on end-to-end causal inference.

Azua's DECI (deep end-to-end causal inference) technology is a single model that can simultaneously do causal discovery and causal inference. The user provides data, and the model can output the causal relationships among all variables.

By itself, this approach can provide insights into the data. It enables the calculation of metrics such as individual treatment effect (ITE), average treatment effect (ATE), and conditional average treatment effect (CATE). You can then use these calculations to make optimal decisions.

The framework is scalable for large data, in terms of both the number of variables and the number of data points. It can also handle missing data entries with mixed

statistical types.

- [EconML](#) powers the back end of the Responsible AI dashboard's causal inference component. It's a Python package that applies machine learning techniques to estimate individualized causal responses from observational or experimental data.

The suite of estimation methods in EconML represents the latest advances in causal machine learning. By incorporating individual machine learning steps into interpretable causal models, these methods improve the reliability of what-if predictions and make causal analysis quicker and easier for a broad set of users.

- [DoWhy](#) is a Python library that aims to spark causal thinking and analysis. DoWhy provides a principled four-step interface for causal inference that focuses on explicitly modeling causal assumptions and validating them as much as possible.

The key feature of DoWhy is its state-of-the-art refutation API that can automatically test causal assumptions for any estimation method. It makes inference more robust and accessible to non-experts.

DoWhy supports estimation of the average causal effect for back-door, front-door, instrumental variable, and other identification methods. It also supports estimation of the CATE through an integration with the EconML library.

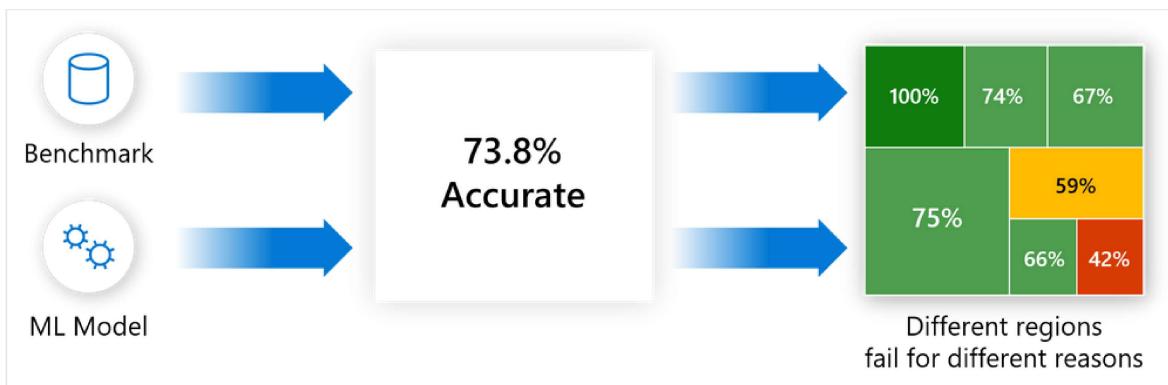
Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the [supported causal inference visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

Assess errors in machine learning models

Article • 11/09/2022 • 2 minutes to read

One of the biggest challenges with current model-debugging practices is using aggregate metrics to score models on a benchmark dataset. Model accuracy might not be uniform across subgroups of data, and there might be input cohorts for which the model fails more often. The direct consequences of these failures are a lack of reliability and safety, the appearance of fairness issues, and a loss of trust in machine learning altogether.



Error analysis moves away from aggregate accuracy metrics. It exposes the distribution of errors to developers in a transparent way, and it enables them to identify and diagnose errors efficiently.

The error analysis component of the [Responsible AI dashboard](#) provides machine learning practitioners with a deeper understanding of model failure distribution and helps them quickly identify erroneous cohorts of data. This component identifies the cohorts of data with a higher error rate versus the overall benchmark error rate. It contributes to the identification stage of the model lifecycle workflow through:

- A decision tree that reveals cohorts with high error rates.
- A heatmap that visualizes how input features affect the error rate across cohorts.

Discrepancies in errors might occur when the system underperforms for specific demographic groups or infrequently observed input cohorts in the training data.

The capabilities of this component come from the [Error Analysis](#) package, which generates model error profiles.

Use error analysis when you need to:

- Gain a deep understanding of how model failures are distributed across a dataset and across several input and feature dimensions.
- Break down the aggregate performance metrics to automatically discover erroneous cohorts in order to inform your targeted mitigation steps.

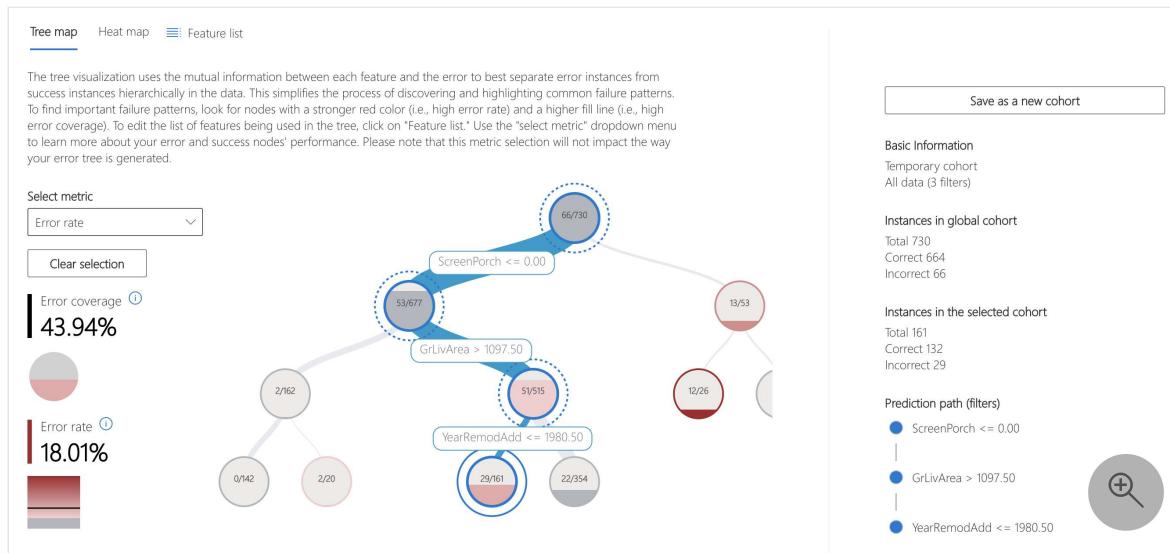
Error tree

Often, error patterns are complex and involve more than one or two features.

Developers might have difficulty exploring all possible combinations of features to discover hidden data pockets with critical failures.

To alleviate the burden, the binary tree visualization automatically partitions the benchmark data into interpretable subgroups that have unexpectedly high or low error rates. In other words, the tree uses the input features to maximally separate model error from success. For each node that defines a data subgroup, users can investigate the following information:

- **Error rate:** A portion of instances in the node for which the model is incorrect. It's shown through the intensity of the red color.
- **Error coverage:** A portion of all errors that fall into the node. It's shown through the fill rate of the node.
- **Data representation:** The number of instances in each node of the error tree. It's shown through the thickness of the incoming edge to the node, along with the total number of instances in the node.



Error heatmap

The view slices the data based on a one-dimensional or two-dimensional grid of input features. Users can choose the input features of interest for analysis.

The heatmap visualizes cells with high error by using a darker red color to bring the user's attention to those regions. This feature is especially beneficial when the error themes are different across partitions, which happens often in practice. In this error identification view, the analysis is highly guided by the users and their knowledge or hypotheses of what features might be most important for understanding failures.



Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the [supported error analysis visualizations](#).

- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

Understand your datasets

Article • 11/09/2022 • 2 minutes to read

Machine learning models "learn" from historical decisions and actions captured in training data. As a result, their performance in real-world scenarios is heavily influenced by the data they're trained on. When feature distribution in a dataset is skewed, it can cause a model to incorrectly predict data points that belong to an underrepresented group or to be optimized along an inappropriate metric.

For example, while a model was training an AI system for predicting house prices, the training set was representing 75 percent of newer houses that had less than median prices. As a result, it was much less accurate in successfully identifying more expensive historic houses. The fix was to add older and expensive houses to the training data and augment the features to include insights about historical value. That data augmentation improved results.

The data analysis component of the [Responsible AI dashboard](#) helps visualize datasets based on predicted and actual outcomes, error groups, and specific features. It helps you identify issues of overrepresentation and underrepresentation and to see how data is clustered in the dataset. Data visualizations consist of aggregate plots or individual data points.

When to use data analysis

Use data analysis when you need to:

- Explore your dataset statistics by selecting different filters to slice your data into different dimensions (also known as cohorts).
- Understand the distribution of your dataset across different cohorts and feature groups.
- Determine whether your findings related to fairness, error analysis, and causality (derived from other dashboard components) are a result of your dataset's distribution.
- Decide in which areas to collect more data to mitigate errors that come from representation issues, label noise, feature noise, label bias, and similar factors.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).

- Explore the supported data analysis visualizations of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

Counterfactuals analysis and what-if

Article • 11/09/2022 • 2 minutes to read

What-if counterfactuals address the question of what the model would predict if you changed the action input. They enable understanding and debugging of a machine learning model in terms of how it reacts to input (feature) changes.

Standard interpretability techniques approximate a machine learning model or rank features by their predictive importance. By contrast, counterfactual analysis "interrogates" a model to determine what changes to a particular data point would flip the model decision.

Such an analysis helps in disentangling the impact of correlated features in isolation. It also helps you get a more nuanced understanding of how much of a feature change is needed to see a model decision flip for classification models and a decision change for regression models.

The *counterfactual analysis and what-if* component of the [Responsible AI dashboard](#) has two functions:

- Generate a set of examples with minimal changes to a particular point such that they change the model's prediction (showing the closest data points with opposite model predictions).
- Enable users to generate their own what-if perturbations to understand how the model reacts to feature changes.

One of the top differentiators of the Responsible AI dashboard's counterfactual analysis component is the fact that you can identify which features to vary and their permissible ranges for valid and logical counterfactual examples.

The capabilities of this component come from the [DiCE](#) package.

Use what-if counterfactuals when you need to:

- Examine fairness and reliability criteria as a decision evaluator by perturbing sensitive attributes such as gender and ethnicity, and then observing whether model predictions change.
- Debug specific input instances in depth.
- Provide solutions to users and determine what they can do to get a desirable outcome from the model.

How are counterfactual examples generated?

To generate counterfactuals, DiCE implements a few model-agnostic techniques. These methods apply to any opaque-box classifier or regressor. They're based on sampling nearby points to an input point, while optimizing a loss function based on proximity (and optionally, sparsity, diversity, and feasibility). Currently supported methods are:

- [Randomized search](#): This method samples points randomly near a query point and returns counterfactuals as points whose predicted label is the desired class.
- [Genetic search](#): This method samples points by using a genetic algorithm, given the combined objective of optimizing proximity to the query point, changing as few features as possible, and seeking diversity among the generated counterfactuals.
- [KD tree search](#): This algorithm returns counterfactuals from the training dataset. It constructs a KD tree over the training data points based on a distance function and then returns the closest points to a particular query point that yields the desired predicted label.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLIV2 and SDKv2](#) or [studio UI](#).
- Explore the [supported counterfactual analysis and what-if perturbation visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

What is an Azure Machine Learning workspace?

Article • 02/24/2023 • 5 minutes to read

The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. The workspace keeps a history of all training runs, including logs, metrics, output, and a snapshot of your scripts. You use this information to determine which training run produces the best model.

Once you have a model you like, you register it with the workspace. You then use the registered model and scoring scripts to deploy to an [online endpoint](#) as a REST-based HTTP endpoint.

Taxonomy

- A workspace can contain [Azure Machine Learning compute instances](#), cloud resources configured with the Python environment necessary to run Azure Machine Learning.
- [User roles](#) enable you to share your workspace with other users, teams, or projects.
- [Compute targets](#) are used to run your experiments.
- When you create the workspace, [associated resources](#) are also created for you.
- Jobs are training runs you use to build your models. You can organize your jobs into Experiments.
- [Pipelines](#) are reusable workflows for training and retraining your model.
- [Data assets](#) aid in management of the data you use for model training and pipeline creation.
- Once you have a model you want to deploy, you create a registered model.
- Use the registered model and a scoring script to create an [online endpoint](#).

Tools for workspace interaction

You can interact with your workspace in the following ways:

- On the web:
 - Azure Machine Learning studio ↗
 - Azure Machine Learning designer
- In any Python environment with the [Azure Machine Learning SDK for Python](#) ↗.
- On the command line using the Azure Machine Learning [CLI extension](#)
- [Azure Machine Learning VS Code Extension](#)

Machine learning with a workspace

Machine learning tasks read and/or write artifacts to your workspace.

- Run an experiment to train a model - writes job run results to the workspace.
- Use automated ML to train a model - writes training results to the workspace.
- Register a model in the workspace.
- Deploy a model - uses the registered model to create a deployment.
- Create and run reusable workflows.
- View machine learning artifacts such as jobs, pipelines, models, deployments.
- Track and monitor models.

Workspace management

You can also perform the following workspace management tasks:

Workspace management task	Portal	Studio	Python SDK	Azure CLI	VS Code
Create a workspace	✓	✓	✓	✓	✓
Manage workspace access	✓			✓	
Create and manage compute resources	✓	✓	✓	✓	✓
Create a compute instance		✓	✓	✓	✓

Warning

Moving your Azure Machine Learning workspace to a different subscription, or moving the owning subscription to a new tenant, is not supported. Doing so may cause errors.

Create a workspace

There are multiple ways to create a workspace:

- Use [Azure Machine Learning studio](#) to quickly create a workspace with default settings.
- Use the [Azure portal](#) for a point-and-click interface with more options.
- Use the [Azure Machine Learning SDK for Python](#) to create a workspace on the fly from Python scripts or Jupyter notebooks.
- Use an [Azure Resource Manager template](#) or the [Azure Machine Learning CLI](#) when you need to automate or customize the creation with corporate security standards.
- If you work in Visual Studio Code, use the [VS Code extension](#).

 **Note**

The workspace name is case-insensitive.

Sub resources

These sub resources are the main resources that are made in the Azure Machine Learning workspace.

- VMs: provide computing power for your Azure Machine Learning workspace and are an integral part in deploying and training models.
- Load Balancer: a network load balancer is created for each compute instance and compute cluster to manage traffic even while the compute instance/cluster is stopped.
- Virtual Network: these help Azure resources communicate with one another, the internet, and other on-premises networks.
- Bandwidth: encapsulates all outbound data transfers across regions.

Associated resources

When you create a new workspace, it automatically creates several Azure resources that are used by the workspace:

- [Azure Storage account](#): Is used as the default datastore for the workspace. Jupyter notebooks that are used with your Azure Machine Learning compute instances are stored here as well.

 **Important**

By default, the storage account is a general-purpose v1 account. You can [upgrade this to general-purpose v2](#) after the workspace has been created. Do not enable hierarchical namespace on the storage account after upgrading to general-purpose v2.

To use an existing Azure Storage account, it cannot be of type BlobStorage or a premium account (Premium_LRS and Premium_GRS). It also cannot have a hierarchical namespace (used with Azure Data Lake Storage Gen2). Neither premium storage nor hierarchical namespaces are supported with the *default* storage account of the workspace. You can use premium storage or hierarchical namespace with *non-default* storage accounts.

- [Azure Container Registry](#): Registers docker containers that are used for the following components:
 - [Azure Machine Learning environments](#) when training and deploying models
 - AutoML when deploying
 - [Data profiling](#)

To minimize costs, ACR is **lazy-loaded** until images are needed.

Note

If your subscription setting requires adding tags to resources under it, Azure Container Registry (ACR) created by Azure Machine Learning will fail, since we cannot set tags to ACR.

- [Azure Application Insights](#): Stores monitoring and diagnostics information. For more information, see [Monitor online endpoints](#).

Note

You can delete the Application Insights instance after cluster creation if you want. Deleting it limits the information gathered from the workspace, and may make it more difficult to troubleshoot problems. **If you delete the Application Insights instance created by the workspace, you cannot re-create it without deleting and recreating the workspace.**

- [Azure Key Vault](#): Stores secrets that are used by compute targets and other sensitive information that's needed by the workspace.

Note

You can instead use existing Azure resource instances when you create the workspace with the [Python SDK](#) or the [Azure Machine Learning CLI using an ARM template](#).

Next steps

To learn more about planning a workspace for your organization's requirements, see [Organize and set up Azure Machine Learning](#).

To get started with Azure Machine Learning, see:

- [What is Azure Machine Learning?](#)
- [Create and manage a workspace](#)
- [Recover a workspace after deletion \(soft-delete\)](#)
- [Tutorial: Get started with Azure Machine Learning](#)
- [Tutorial: Create your first classification model with automated machine learning](#)
- [Tutorial: Predict automobile price with the designer](#)

Additional resources

Documentation

[Quickstart: Create workspace resources - Azure Machine Learning](#)

Create an Azure Machine Learning workspace and cloud resources that can be used to train machine learning models.

[Monitoring Azure Machine Learning - Azure Machine Learning](#)

Learn how to use Azure Monitor to view, analyze, and create alerts on metrics from Azure Machine Learning.

[What is an Azure Machine Learning compute instance? - Azure Machine Learning](#)

Learn about the Azure Machine Learning compute instance, a fully managed cloud-based workstation.

[Manage and optimize costs - Azure Machine Learning](#)

Learn tips to optimize your cost when building machine learning models in Azure Machine Learning

[Service limits - Azure Machine Learning](#)

Service limits used for capacity planning and maximum limits on requests and responses for Azure Machine Learning.

[Quickstart: Interactive Data Wrangling with Apache Spark \(preview\) - Azure Machine Learning](#)

Learn how to perform interactive data wrangling with Apache Spark in Azure Machine Learning

[Create and manage a compute instance - Azure Machine Learning](#)

Learn how to create and manage an Azure Machine Learning compute instance. Use as your development environment, or as compute target for dev/test purposes.

[Search for assets - Azure Machine Learning](#)

Find your Azure Machine Learning assets with search

[Show 5 more](#)

Training

Learning paths and modules

[Create Azure Machine Learning resources with the CLI \(v2\) - Training](#)

Create Azure Machine Learning resources with the CLI (v2)

Learning certificate

[Microsoft Certified: Azure Data Scientist Associate - Certifications](#)

Azure data scientists have subject matter expertise in applying data science and machine learning to implement and run machine learning workloads on Azure.

What are Azure Machine Learning environments?

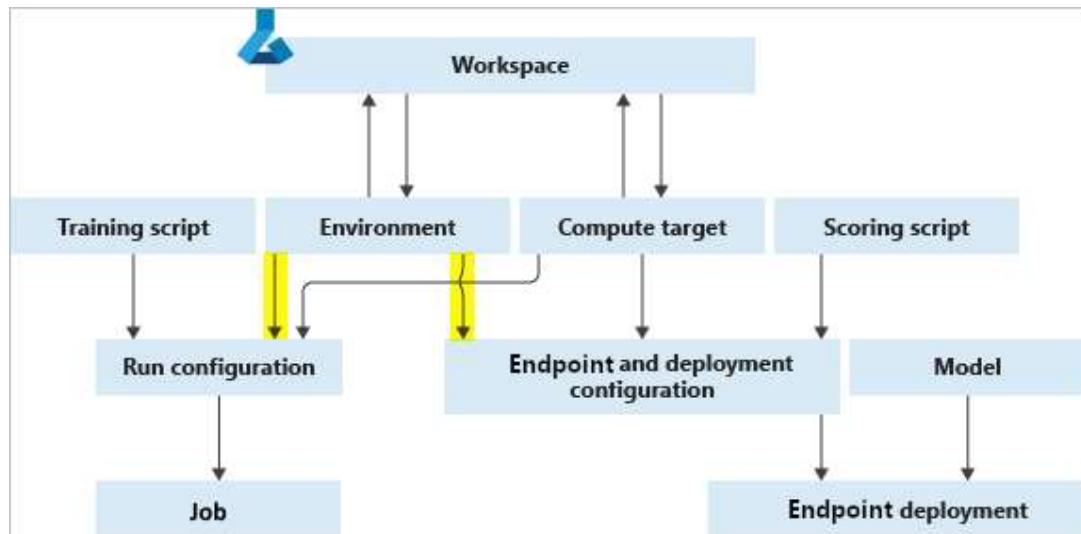
Article • 02/24/2023 • 6 minutes to read

Azure Machine Learning environments are an encapsulation of the environment where your machine learning training happens. They specify the Python packages, environment variables, and software settings around your training and scoring scripts. They also specify runtimes (Python, Spark, or Docker). The environments are managed and versioned entities within your Machine Learning workspace that enable reproducible, auditable, and portable machine learning workflows across a variety of compute targets.

You can use an `Environment` object on your local compute to:

- Develop your training script.
- Reuse the same environment on Azure Machine Learning Compute for model training at scale.
- Deploy your model with that same environment.
- Revisit the environment in which an existing model was trained.

The following diagram illustrates how you can use a single `Environment` object in both your job configuration (for training) and your inference and deployment configuration (for web service deployments).



The environment, compute target and training script together form the job configuration: the full specification of a training job.

Types of environments

Environments can broadly be divided into three categories: *curated*, *user-managed*, and *system-managed*.

Curated environments are provided by Azure Machine Learning and are available in your workspace by default. Intended to be used as is, they contain collections of Python packages and settings to help you get started with various machine learning frameworks. These pre-created environments also allow for faster deployment time. For a full list, see the [curated environments article](#).

In user-managed environments, you're responsible for setting up your environment and installing every package that your training script needs on the compute target. Also be sure to include any dependencies needed for model deployment.

You use system-managed environments when you want [conda](#) to manage the Python environment for you. A new conda environment is materialized from your conda specification on top of a base docker image.

Create and manage environments

You can create environments from clients like the Azure Machine Learning Python SDK, Azure Machine Learning CLI, Environments page in Azure Machine Learning studio, and [VS Code extension](#). Every client allows you to customize the base image, Dockerfile, and Python layer if needed.

For specific code samples, see the "Create an environment" section of [How to use environments](#).

Environments are also easily managed through your workspace, which allows you to:

- Register environments.
- Fetch environments from your workspace to use for training or deployment.
- Create a new instance of an environment by editing an existing one.
- View changes to your environments over time, which ensures reproducibility.
- Build Docker images automatically from your environments.

"Anonymous" environments are automatically registered in your workspace when you submit an experiment. They will not be listed but may be retrieved by version.

For code samples, see the "Manage environments" section of [How to use environments](#).

Environment building, caching, and reuse

Azure Machine Learning builds environment definitions into Docker images and conda environments. It also caches the environments so they can be reused in subsequent training jobs and service endpoint deployments. Running a training script remotely requires the creation of a Docker image, but a local job can use a conda environment directly.

Submitting a job using an environment

When you first submit a remote job using an environment, the Azure Machine Learning service invokes an [ACR Build Task](#) on the Azure Container Registry (ACR) associated with the Workspace. The built Docker image is then cached on the Workspace ACR. Curated environments are backed by Docker images that are cached in Global ACR. At the start of the job execution, the image is retrieved by the compute target from the relevant ACR.

For local jobs, a Docker or conda environment is created based on the environment definition. The scripts are then executed on the target compute - a local runtime environment or local Docker engine.

Building environments as Docker images

If the image for a particular environment definition doesn't already exist in the workspace ACR, a new image will be built. The image build consists of two steps:

1. Downloading a base image, and executing any Docker steps
2. Building a conda environment according to conda dependencies specified in the environment definition.

The second step is optional, and the environment may instead come from the Docker build context or base image. In this case you're responsible for installing any Python packages, by including them in your base image, or specifying custom Docker steps. You're also responsible for specifying the correct location for the Python executable. It is also possible to use a [custom Docker base image](#).

Image caching and reuse

If you use the same environment definition for another job, Azure Machine Learning reuses the cached image from the Workspace ACR to save time.

To view the details of a cached image, check the Environments page in Azure Machine Learning studio or use [MLClient.environments](#) to get and inspect the environment.

To determine whether to reuse a cached image or build a new one, Azure Machine Learning computes a [hash value](#) from the environment definition and compares it to the hashes of existing environments. The hash is based on the environment definition's:

- Base image
- Custom docker steps
- Python packages
- Spark packages

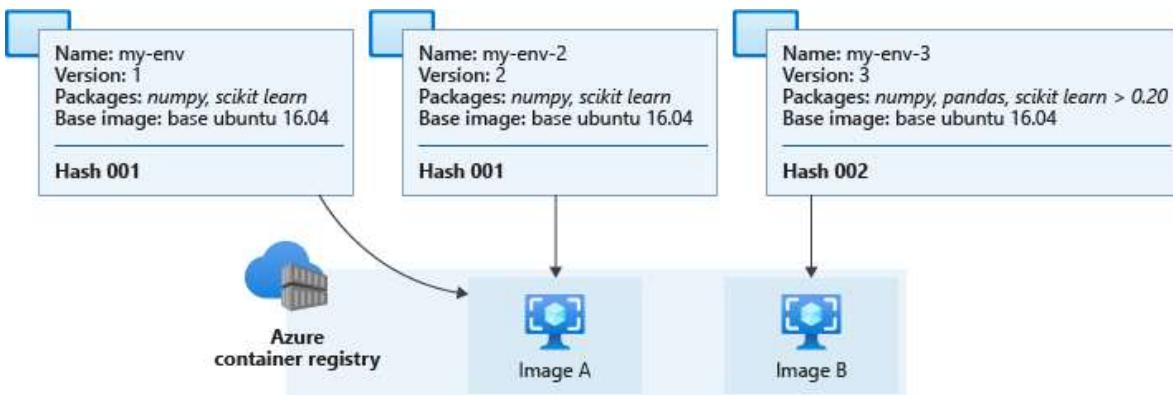
The hash isn't affected by the environment name or version. If you rename your environment or create a new one with the same settings and packages as another environment, then the hash value will remain the same. However, environment definition changes like adding or removing a Python package or changing a package version will result cause the resulting hash value to change. Changing the order of dependencies or channels in an environment will also change the hash and require a new image build. Similarly, any change to a curated environment will result in the creation of a new "non-curated" environment.

 **Note**

You will not be able to submit any local changes to a curated environment without changing the name of the environment. The prefixes "AzureML-" and "Microsoft" are reserved exclusively for curated environments, and your job submission will fail if the name starts with either of them.

The environment's computed hash value is compared with those in the Workspace and global ACR, or on the compute target (local jobs only). If there is a match then the cached image is pulled and used, otherwise an image build is triggered.

The following diagram shows three environment definitions. Two of them have different names and versions but identical base images and Python packages, which results in the same hash and corresponding cached image. The third environment has different Python packages and versions, leading to a different hash and cached image.



Actual cached images in your workspace ACR will have names like

`azureml/azureml_e9607b2514b066c851012848913ba19f` with the hash appearing at the end.

ⓘ Important

- If you create an environment with an unpinned package dependency (for example, `numpy`), the environment uses the package version that was *available when the environment was created*. Any future environment that uses a matching definition will use the original version.

To update the package, specify a version number to force an image rebuild.

An example of this would be changing `numpy` to `numpy==1.18.1`. New dependencies--including nested ones--will be installed, and they might break a previously working scenario.

- Using an unpinned base image like `mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04` in your environment definition results in rebuilding the image every time the `:latest` tag is updated. This helps the image receive the latest patches and system updates.zzs

Image patching

Microsoft is responsible for patching the base images for known security vulnerabilities. Updates for supported images are released every two weeks, with a commitment of no unpatched vulnerabilities older than 30 days in the latest version of the image. Patched images are released with a new immutable tag and the `:latest` tag is updated to the latest version of the patched image.

If you provide your own images, you are responsible for updating them.

For more information on the base images, see the following links:

- Azure Machine Learning base images [GitHub repository](#).
- Use a custom container to deploy a model to an online endpoint

Next steps

- Learn how to [create and use environments](#) in Azure Machine Learning.
 - See the Python SDK reference documentation for the [environment class](#).
-

Additional resources

Documentation

[Python SDK release notes - Azure Machine Learning](#)

Learn about the latest updates to Azure Machine Learning Python SDK.

[Use software environments CLI v1 - Azure Machine Learning](#)

Create and manage environments for model training and deployment with CLI v1. Manage Python packages and other settings for the environment.

[Migrate logging from SDK v1 to MLflow - Azure Machine Learning](#)

Comparison of SDK v1 logging APIs and MLflow tracking

[Create a Training Job with the job creation UI - Azure Machine Learning](#)

Learn how to use the job creation UI in Azure Machine Learning studio to create a training job.

[azureml.pipeline.core.PublishedPipeline class - Azure Machine Learning Python](#)

Represents a Pipeline to be submitted without the Python code which constructed it. In addition, a PublishedPipeline can be used to resubmit a Pipeline with different PipelineParameter values and inputs.

[Customize outputs in batch deployments - Azure Machine Learning](#)

Learn how create deployments that generate custom outputs and files.

[CLI \(v2\) mltable YAML schema - Azure Machine Learning](#)

Reference documentation for the CLI (v2) MLTable YAML schema.

[Show 4 more](#)

What is an Azure Machine Learning compute instance?

Article • 01/23/2023 • 7 minutes to read

An Azure Machine Learning compute instance is a managed cloud-based workstation for data scientists. Each compute instance has only one owner, although you can share files between multiple compute instances.

Compute instances make it easy to get started with Azure Machine Learning development and provide management and enterprise readiness capabilities for IT administrators.

Use a compute instance as your fully configured and managed development environment in the cloud for machine learning. They can also be used as a compute target for training and inferencing for development and testing purposes.

For compute instance Jupyter functionality to work, ensure that web socket communication isn't disabled. Ensure your network allows websocket connections to *.instances.azureml.net and *.instances.azureml.ms.

Important

Items marked (preview) in this article are currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Why use a compute instance?

A compute instance is a fully managed cloud-based workstation optimized for your machine learning development environment. It provides the following benefits:

Key benefits	Description
--------------	-------------

Key benefits	Description
Productivity	<p>You can build and deploy models using integrated notebooks and the following tools in Azure Machine Learning studio:</p> <ul style="list-style-type: none"> - Jupyter - JupyterLab - VS Code (preview) <p>Compute instance is fully integrated with Azure Machine Learning workspace and studio. You can share notebooks and data with other data scientists in the workspace.</p>
Managed & secure	<p>Reduce your security footprint and add compliance with enterprise security requirements. Compute instances provide robust management policies and secure networking configurations such as:</p> <ul style="list-style-type: none"> - Autoprovisioning from Resource Manager templates or Azure Machine Learning SDK - Azure role-based access control (Azure RBAC) - Virtual network support - Azure policy to disable SSH access - Azure policy to enforce creation in a virtual network - Auto-shutdown/auto-start based on schedule - TLS 1.2 enabled
Preconfigured for ML	<p>Save time on setup tasks with pre-configured and up-to-date ML packages, deep learning frameworks, GPU drivers.</p>
Fully customizable	<p>Broad support for Azure VM types including GPUs and persisted low-level customization such as installing packages and drivers makes advanced scenarios a breeze. You can also use setup scripts to automate customization</p>

- Secure your compute instance with [No public IP](#).
- The compute instance is also a secure training compute target similar to [compute clusters](#), but it's single node.
- You can [create a compute instance](#) yourself, or an administrator can [create a compute instance on your behalf](#).
- You can also [use a setup script \(preview\)](#) for an automated way to customize and configure the compute instance as per your needs.
- To save on costs, [create a schedule](#) to automatically start and stop the compute instance, or [enable idle shutdown](#)

Tools and environments

Azure Machine Learning compute instance enables you to author, train, and deploy models in a fully integrated notebook experience in your workspace.

You can run Jupyter notebooks in VS Code [using compute instance as the remote server](#) with no SSH needed. You can also enable VS Code integration through [remote SSH extension](#).

You can [install packages](#) and [add kernels](#) to your compute instance.

Following tools and environments are already installed on the compute instance:

General tools & environments	Details
Drivers	CUDA cuDNN NVIDIA Blob FUSE
Intel MPI library	
Azure CLI	
Azure Machine Learning samples	
Docker	
Nginx	
NCCL 2.0	
Protobuf	

R tools & environments	Details
R kernel	

You can [Add RStudio or Posit Workbench \(formerly RStudio Workbench\)](#) when you create the instance.

PYTHON tools & environments	Details
Anaconda Python	
Jupyter and extensions	
Jupyterlab and extensions	

PYTHON tools & environments	Details
Azure Machine Learning SDK for Python from PyPI	<p>Includes most of the azureml extra packages. To see the full list, open a terminal window on your compute instance and run</p> <pre>conda list -n azureml_py36 azureml*</pre>
Other PyPI packages	jupyter tensorboard nbconvert notebook Pillow
Conda packages	cython numpy ipykernel scikit-learn matplotlib tqdm joblib nodejs
Deep learning packages	PyTorch TensorFlow Keras Horovod MLflow pandas-ml scrapbook
ONNX packages	keras2onnx onnx onnxconverter-common skl2onnx onnxmltools
Azure Machine Learning Python samples	

Python packages are all installed in the **Python 3.8 - AzureML** environment. Compute instance has Ubuntu 20.04 as the base OS.

Accessing files

Notebooks and Python scripts are stored in the default storage account of your workspace in Azure file share. These files are located under your “User files” directory. This storage makes it easy to share notebooks between compute instances. The storage account also keeps your notebooks safely preserved when you stop or delete a compute instance.

The Azure file share account of your workspace is mounted as a drive on the compute instance. This drive is the default working directory for Jupyter, Jupyter Labs, RStudio, and Posit Workbench. This means that the notebooks and other files you create in Jupyter, JupyterLab, RStudio, or Posit are automatically stored on the file share and available to use in other compute instances as well.

The files in the file share are accessible from all compute instances in the same workspace. Any changes to these files on the compute instance will be reliably persisted back to the file share.

You can also clone the latest Azure Machine Learning samples to your folder under the user files directory in the workspace file share.

Writing small files can be slower on network drives than writing to the compute instance local disk itself. If you're writing many small files, try using a directory directly on the compute instance, such as a `/tmp` directory. Note these files won't be accessible from other compute instances.

Don't store training data on the notebooks file share. You can use the `/tmp` directory on the compute instance for your temporary data. However, don't write large files of data on the OS disk of the compute instance. OS disk on compute instance has 128-GB capacity. You can also store temporary training data on temporary disk mounted on `/mnt`. Temporary disk size is based on the VM size chosen and can store larger amounts of data if a higher size VM is chosen. You can also mount [datastores and datasets](#). Any software packages you install are saved on the OS disk of compute instance. Note customer managed key encryption is currently not supported for OS disk. The OS disk for compute instance is encrypted with Microsoft-managed keys.

Create

Follow the steps in the [Quickstart: Create workspace resources you need to get started with Azure Machine Learning](#) to create a basic compute instance.

For more options, see [create a new compute instance](#).

As an administrator, you can [create a compute instance for others in the workspace \(preview\)](#).

You can also [use a setup script \(preview\)](#) for an automated way to customize and configure the compute instance.

Other ways to create a compute instance:

- Directly from the integrated notebooks experience.
- From Azure Resource Manager template. For an example template, see the [create an Azure Machine Learning compute instance template](#).
- With [Azure Machine Learning SDK](#)
- From the [CLI extension for Azure Machine Learning](#)

The dedicated cores per region per VM family quota and total regional quota, which applies to compute instance creation, is unified and shared with Azure Machine Learning training compute cluster quota. Stopping the compute instance doesn't release quota to ensure you'll be able to restart the compute instance. Don't stop the compute instance through the OS terminal by doing a sudo shutdown.

Compute instance comes with P10 OS disk. Temp disk type depends on the VM size chosen. Currently, it isn't possible to change the OS disk type.

Compute target

Compute instances can be used as a [training compute target](#) similar to Azure Machine Learning [compute training clusters](#). But a compute instance has only a single node, while a compute cluster can have more nodes.

A compute instance:

- Has a job queue.
- Runs jobs securely in a virtual network environment, without requiring enterprises to open up SSH port. The job executes in a containerized environment and packages your model dependencies in a Docker container.
- Can run multiple small jobs in parallel (preview). One job per core can run in parallel while the rest of the jobs are queued.
- Supports single-node multi-GPU [distributed training](#) jobs

You can use compute instance as a local inferencing deployment target for test/debug scenarios.

💡 Tip

The compute instance has 120GB OS disk. If you run out of disk space and get into an unusable state, please clear at least 5 GB disk space on OS disk (mounted on /)

through the compute instance terminal by removing files/folders and then do `sudo reboot`. Temporary disk will be freed after restart; you do not need to clear space on temp disk manually. To access the terminal go to compute list page or compute instance details page and click on **Terminal** link. You can check available disk space by running `df -h` on the terminal. Clear at least 5 GB space before doing `sudo reboot`. Please do not stop or restart the compute instance through the Studio until 5 GB disk space has been cleared. Auto shutdowns, including scheduled start or stop as well as idle shutdowns(preview), will not work if the CI disk is full.

Next steps

- [Quickstart: Create workspace resources you need to get started with Azure Machine Learning.](#)
 - [Tutorial: Train your first ML model](#) shows how to use a compute instance with an integrated notebook.
-

Additional resources

Documentation

[Distributed GPU training guide \(SDK v2\) - Azure Machine Learning](#)

Learn the best practices for performing distributed training with Azure Machine Learning SDK (v2) supported frameworks, such as MPI, Horovod, DeepSpeed, PyTorch, TensorFlow, and InfiniBand.

[How to use studio UI to build and debug Machine Learning pipelines - Azure Machine Learning](#)

Learn how to build, debug, clone, and compare V2 pipeline with the studio UI.

[Manage and optimize costs - Azure Machine Learning](#)

Learn tips to optimize your cost when building machine learning models in Azure Machine Learning

[Create and manage a compute instance with CLI v1 - Azure Machine Learning](#)

Learn how to create and manage an Azure Machine Learning compute instance with CLI v1. Use as your development environment, or as compute target for dev/test purposes.

[Export or delete workspace data - Azure Machine Learning](#)

Learn how to export or delete your workspace with the Azure Machine Learning studio.

[Use automated ML in ML pipelines - Azure Machine Learning](#)

The AutoMLStep allows you to use automated machine learning in your pipelines.

[Profile model memory and CPU usage \(v1\) - Azure Machine Learning](#)

Use CLI (v1) or SDK (v1) to profile your model before deployment. Profiling determines the memory and CPU usage of your model.

[Create compute clusters - Azure Machine Learning](#)

Learn how to create compute clusters in your Azure Machine Learning workspace. Use the compute cluster as a compute target for training or inference.

[Show 5 more](#)

Training

Learning paths and modules

[Work with Compute in Azure Machine Learning - Training](#)

Work with Compute in Azure Machine Learning

What are compute targets in Azure Machine Learning?

Article • 12/29/2022 • 8 minutes to read

A *compute target* is a designated compute resource or environment where you run your training script or host your service deployment. This location might be your local machine or a cloud-based compute resource. Using compute targets makes it easy for you to later change your compute environment without having to change your code.

In a typical model development lifecycle, you might:

1. Start by developing and experimenting on a small amount of data. At this stage, use your local environment, such as a local computer or cloud-based virtual machine (VM), as your compute target.
2. Scale up to larger data, or do [distributed training](#) by using one of these [training compute targets](#).
3. After your model is ready, deploy it to a web hosting environment with one of these [deployment compute targets](#).

The compute resources you use for your compute targets are attached to a [workspace](#). Compute resources other than the local machine are shared by users of the workspace.

Training compute targets

Azure Machine Learning has varying support across different compute targets. A typical model development lifecycle starts with development or experimentation on a small amount of data. At this stage, use a local environment like your local computer or a cloud-based VM. As you scale up your training on larger datasets or perform [distributed training](#), use Azure Machine Learning compute to create a single- or multi-node cluster that autoscales each time you submit a job. You can also attach your own compute resource, although support for different scenarios might vary.

Compute targets can be reused from one training job to the next. For example, after you attach a remote VM to your workspace, you can reuse it for multiple jobs. For machine learning pipelines, use the appropriate [pipeline step](#) for each compute target.

You can use any of the following resources for a training compute target for most jobs. Not all resources can be used for automated machine learning, machine learning pipelines, or designer. Azure Databricks can be used as a training resource for local runs and machine learning pipelines, but not as a remote target for other training.

Training targets	Automated machine learning	Machine learning pipelines	Azure Machine Learning designer
Local computer	Yes		
Azure Machine Learning compute cluster	Yes	Yes	Yes
Azure Machine Learning compute instance	Yes (through SDK)	Yes	Yes
Azure Machine Learning Kubernetes	Yes	Yes	Yes
Remote VM	Yes	Yes	
Apache Spark pools (preview)	Yes (SDK local mode only)	Yes	
Azure Databricks	Yes (SDK local mode only)	Yes	
Azure Data Lake Analytics		Yes	
Azure HDInsight		Yes	
Azure Batch		Yes	

💡 Tip

The compute instance has 120GB OS disk. If you run out of disk space, **use the terminal** to clear at least 1-2 GB before you **stop or restart** the compute instance.

Compute targets for inference

When performing inference, Azure Machine Learning creates a Docker container that hosts the model and associated resources needed to use it. This container is then used in a compute target.

The compute target you use to host your model will affect the cost and availability of your deployed endpoint. Use this table to choose an appropriate compute target.

Compute target	Used for	GPU support	Description

Compute target	Used for	GPU support	Description
Local web service	Testing/debugging		Use for limited testing and troubleshooting. Hardware acceleration depends on use of libraries in the local system.
Azure Machine Learning endpoints	Real-time inference	Yes	Fully managed computes for real-time (managed online endpoints) and batch scoring (batch endpoints) on serverless compute.
	Batch inference		
Azure Machine Learning Kubernetes	Real-time inference	Yes	Run inferencing workloads on on-premises, cloud, and edge Kubernetes clusters.
	Batch inference		
Azure Container Instances (SDK/CLI v1 only)	Real-time inference		Use for low-scale CPU-based workloads that require less than 48 GB of RAM. Doesn't require you to manage a cluster.
	Recommended for dev/test purposes only.		Supported in the designer.

ⓘ Note

When choosing a cluster SKU, first scale up and then scale out. Start with a machine that has 150% of the RAM your model requires, profile the result and find a machine that has the performance you need. Once you've learned that, increase the number of machines to fit your need for concurrent inference.

ⓘ Note

Container instances require the SDK or CLI v1 and are suitable only for small models less than 1 GB in size.

Learn [where and how to deploy your model to a compute target](#).

Azure Machine Learning compute (managed)

A managed compute resource is created and managed by Azure Machine Learning. This compute is optimized for machine learning workloads. Azure Machine Learning compute clusters and [compute instances](#) are the only managed computes.

You can create Azure Machine Learning compute instances or compute clusters from:

- [Azure Machine Learning studio](#).
- The Python SDK and the Azure CLI:
 - [Compute instance](#).
 - [Compute cluster](#).
- An Azure Resource Manager template. For an example template, see [Create an Azure Machine Learning compute cluster ↗](#).

When created, these compute resources are automatically part of your workspace, unlike other kinds of compute targets.

Capability	Compute cluster	Compute instance
Single- or multi-node cluster	✓	Single node cluster
Autoscales each time you submit a job	✓	
Automatic cluster management and job scheduling	✓	✓
Support for both CPU and GPU resources	✓	✓

! Note

To avoid charges when the compute is idle:

- For compute *cluster* make sure the minimum number of nodes is set to 0.
- For a compute *instance*, enable idle shutdown.

Supported VM series and sizes

! Note

H-series virtual machine series will be retired on August 31, 2022. Create compute instance and compute clusters with alternate VM sizes. Existing compute instances and clusters with H-series virtual machines will not work after August 31, 2022.

When you select a node size for a managed compute resource in Azure Machine Learning, you can choose from among select VM sizes available in Azure. Azure offers a range of sizes for Linux and Windows for different workloads. To learn more, see [VM types and sizes](#).

There are a few exceptions and limitations to choosing a VM size:

- Some VM series aren't supported in Azure Machine Learning.
- There are some VM series, such as GPUs and other special SKUs, which may not initially appear in your list of available VMs. But you can still use them, once you request a quota change. For more information about requesting quotas, see [Request quota increases](#). See the following table to learn more about supported series.

Supported VM series	Category	Supported by
DDSv4	General purpose	Compute clusters and instance
Dv2	General purpose	Compute clusters and instance
Dv3	General purpose	Compute clusters and instance
DSv2	General purpose	Compute clusters and instance
DSv3	General purpose	Compute clusters and instance
EAv4	Memory optimized	Compute clusters and instance
Ev3	Memory optimized	Compute clusters and instance
ESv3	Memory optimized	Compute clusters and instance
FSv2	Compute optimized	Compute clusters and instance
FX	Compute optimized	Compute clusters
H	High performance compute	Compute clusters and instance
HB	High performance compute	Compute clusters and instance
HBv2	High performance compute	Compute clusters and instance
HBv3	High performance compute	Compute clusters and instance
HC	High performance compute	Compute clusters and instance
LSv2	Storage optimized	Compute clusters and instance
M	Memory optimized	Compute clusters and instance
NC	GPU	Compute clusters and instance
NC Promo	GPU	Compute clusters and instance
NCv2	GPU	Compute clusters and instance
NCv3	GPU	Compute clusters and instance

Supported VM series	Category	Supported by
ND	GPU	Compute clusters and instance
NDv2	GPU	Compute clusters and instance
NV	GPU	Compute clusters and instance
NVv3	GPU	Compute clusters and instance
NCasT4_v3	GPU	Compute clusters and instance
NDasrA100_v4	GPU	Compute clusters and instance

While Azure Machine Learning supports these VM series, they might not be available in all Azure regions. To check whether VM series are available, see [Products available by region](#).

ⓘ Note

Azure Machine Learning doesn't support all VM sizes that Azure Compute supports. To list the available VM sizes, use one of the following methods:

- [REST API](#)
- The [Azure CLI extension 2.0 for machine learning](#) command, `az ml compute list-sizes`.

If using the GPU-enabled compute targets, it is important to ensure that the correct CUDA drivers are installed in the training environment. Use the following table to determine the correct CUDA version to use:

GPU Architecture	Azure VM Series	Supported CUDA versions
Ampere	NDA100_v4	11.0+
Turing	NCT4_v3	10.0+
Volta	NCv3, NDv2	9.0+
Pascal	NCv2, ND	9.0+
Maxwell	NV, NVv3	9.0+
Kepler	NC, NC Promo	9.0+

In addition to ensuring the CUDA version and hardware are compatible, also ensure that the CUDA version is compatible with the version of the machine learning framework you

are using:

- For PyTorch, you can check the compatibility by visiting [Pytorch's previous versions page](#).
- For Tensorflow, you can check the compatibility by visiting [Tensorflow's build from source page](#).

Compute isolation

Azure Machine Learning compute offers VM sizes that are isolated to a specific hardware type and dedicated to a single customer. Isolated VM sizes are best suited for workloads that require a high degree of isolation from other customers' workloads for reasons that include meeting compliance and regulatory requirements. Utilizing an isolated size guarantees that your VM will be the only one running on that specific server instance.

The current isolated VM offerings include:

- Standard_M128ms
- Standard_F72s_v2
- Standard_NC24s_v3
- Standard_NC24rs_v3*

*RDMA capable

To learn more about isolation, see [Isolation in the Azure public cloud](#).

Unmanaged compute

An unmanaged compute target is *not* managed by Azure Machine Learning. You create this type of compute target outside Azure Machine Learning and then attach it to your workspace. Unmanaged compute resources can require additional steps for you to maintain or to improve performance for machine learning workloads.

Azure Machine Learning supports the following unmanaged compute types:

- Remote virtual machines
- Azure HDInsight
- Azure Databricks
- Azure Data Lake Analytics

- Azure Synapse Spark pool (preview)

 **Tip**

Currently this requires the Azure Machine Learning SDK v1.

- Kubernetes

For more information, see [Manage compute resources](#).

Next steps

Learn how to:

- [Deploy your model to a compute target](#)
-

Additional resources

Documentation

[azureml.contrib.pipeline.steps.parallel_run_config.ParallelRunConfig class - Azure Machine Learning Python](#)

Defines configuration for a ParallelRunStep object. Note This package, azureml-contrib-pipeline-steps, has been deprecated and moved to azureml-pipeline-steps. Please use the ParallelRunConfig class from new package. For an example of using ParallelRunStep, see the notebook....

[azureml.core.compute.ComputeTarget class - Azure Machine Learning Python](#)

Abstract parent class for all compute targets managed by Azure Machine Learning. A compute target is a designated compute resource/environment where you run your training script or host your service deployment. This location may be your local machine or a cloud-based compute resource....

[How to view AutoML model training code - Azure Machine Learning AutoML](#)

How to view model training code for an automated ML trained model and explanation of each stage.

[Apply Transformation: Component Reference - Azure Machine Learning](#)

Learn how to use the Apply Transformation component in Azure Machine Learning to modify an input dataset based on a previously computed transformation.

[azureml.core.compute package - Azure Machine Learning Python](#)

This package contains classes used to manage compute targets in Azure Machine Learning. For more information about choosing compute targets for training and deployment, see What are compute targets in Azure Machine Learning?

[Import data into the designer - Azure Machine Learning](#)

Learn how to import data into Azure Machine Learning designer using Azure Machine Learning

datasets and the Import Data component.

[Configure authentication for models deployed as web services - Azure Machine Learning](#)

Learn how to configure authentication for machine learning models deployed to web services in Azure Machine Learning.

[Execute Python Script in the designer - Azure Machine Learning](#)

Learn how to use the Execute Python Script model in Azure Machine Learning designer to run custom operations written in Python.

[Show 5 more](#)

Enterprise security and governance for Azure Machine Learning

Article • 11/15/2022 • 5 minutes to read

In this article, you'll learn about security and governance features available for Azure Machine Learning. These features are useful for administrators, DevOps, and MLOps who want to create a secure configuration that is compliant with your companies policies. With Azure Machine Learning and the Azure platform, you can:

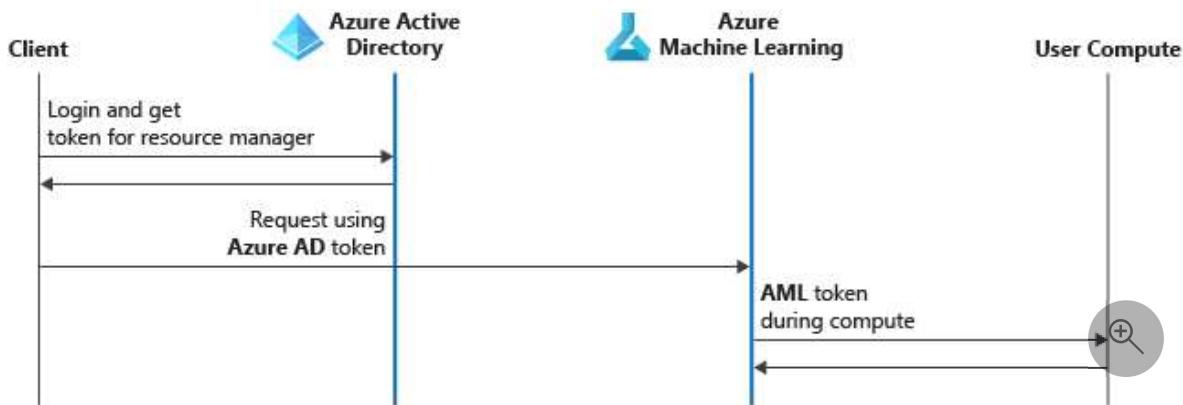
- Restrict access to resources and operations by user account or groups
- Restrict incoming and outgoing network communications
- Encrypt data in transit and at rest
- Scan for vulnerabilities
- Apply and audit configuration policies

Restrict access to resources and operations

Azure Active Directory (Azure AD) is the identity service provider for Azure Machine Learning. It allows you to create and manage the security objects (user, group, service principal, and managed identity) that are used to *authenticate* to Azure resources. Multi-factor authentication is supported if Azure AD is configured to use it.

Here's the authentication process for Azure Machine Learning using multi-factor authentication in Azure AD:

1. The client signs in to Azure AD and gets an Azure Resource Manager token.
2. The client presents the token to Azure Resource Manager and to all Azure Machine Learning.
3. Azure Machine Learning provides a Machine Learning service token to the user compute target (for example, Azure Machine Learning compute cluster). This token is used by the user compute target to call back into the Machine Learning service after the job is complete. The scope is limited to the workspace.



Each workspace has an associated system-assigned [managed identity](#) that has the same name as the workspace. This managed identity is used to securely access resources used by the workspace. It has the following Azure RBAC permissions on associated resources:

Resource	Permissions
Workspace	Contributor
Storage account	Storage Blob Data Contributor
Key vault	Access to all keys, secrets, certificates
Azure Container Registry	Contributor
Resource group that contains the workspace	Contributor

The system-assigned managed identity is used for internal service-to-service authentication between Azure Machine Learning and other Azure resources. The identity token is not accessible to users and cannot be used by them to gain access to these resources. Users can only access the resources through [Azure Machine Learning control and data plane APIs](#), if they have sufficient RBAC permissions.

We don't recommend that admins revoke the access of the managed identity to the resources mentioned in the preceding table. You can restore access by using the [resync keys operation](#).

Note

If your Azure Machine Learning workspaces has compute targets (compute cluster, compute instance, Azure Kubernetes Service, etc.) that were created **before May 14th, 2021**, you may also have an additional Azure Active Directory account. The account name starts with `Microsoft-AzureML-Support-App-` and has contributor-level access to your subscription for every workspace region.

If your workspace does not have an Azure Kubernetes Service (AKS) attached, you can safely delete this Azure AD account.

If your workspace has attached AKS clusters, *and they were created before May 14th, 2021, do not delete this Azure AD account*. In this scenario, you must first delete and recreate the AKS cluster before you can delete the Azure AD account.

You can provision the workspace to use user-assigned managed identity, and grant the managed identity additional roles, for example to access your own Azure Container Registry for base Docker images. You can also configure managed identities for use with Azure Machine Learning compute cluster. This managed identity is independent of workspace managed identity. With a compute cluster, the managed identity is used to access resources such as secured datastores that the user running the training job may not have access to. For more information, see [Use managed identities for access control](#).

Tip

There are some exceptions to the use of Azure AD and Azure RBAC within Azure Machine Learning:

- You can optionally enable **SSH** access to compute resources such as Azure Machine Learning compute instance and compute cluster. SSH access is based on public/private key pairs, not Azure AD. SSH access is not governed by Azure RBAC.
- You can authenticate to models deployed as online endpoints using **key** or **token**-based authentication. Keys are static strings, while tokens are retrieved using an Azure AD security object. For more information, see [How to authenticate online endpoints](#).

For more information, see the following articles:

- [Authentication for Azure Machine Learning workspace](#)
- [Manage access to Azure Machine Learning](#)
- [Connect to storage services](#)
- [Use Azure Key Vault for secrets when training](#)
- [Use Azure AD managed identity with Azure Machine Learning](#)

Network security and isolation

To restrict network access to Azure Machine Learning resources, you can use [Azure Virtual Network \(VNet\)](#). VNets allow you to create network environments that are partially, or fully, isolated from the public internet. This reduces the attack surface for your solution, as well as the chances of data exfiltration.

You might use a virtual private network (VPN) gateway to connect individual clients, or your own network, to the VNet.

The Azure Machine Learning workspace can use [Azure Private Link](#) to create a private endpoint behind the VNet. This provides a set of private IP addresses that can be used to access the workspace from within the VNet. Some of the services that Azure Machine Learning relies on can also use Azure Private Link, but some rely on network security groups or user-defined routing.

For more information, see the following documents:

- [Virtual network isolation and privacy overview](#)
- [Secure workspace resources](#)
- [Secure training environment](#)
- [Secure inference environment](#)
- [Use studio in a secured virtual network](#)
- [Use custom DNS](#)
- [Configure firewall](#)

Data encryption

Azure Machine Learning uses a variety of compute resources and data stores on the Azure platform. To learn more about how each of these supports data encryption at rest and in transit, see [Data encryption with Azure Machine Learning](#).

Data exfiltration prevention (preview)

Azure Machine Learning has several inbound and outbound network dependencies. Some of these dependencies can expose a data exfiltration risk by malicious agents within your organization. These risks are associated with the outbound requirements to Azure Storage, Azure Front Door, and Azure Monitor. For recommendations on mitigating this risk, see the [Azure Machine Learning data exfiltration prevention](#) article.

Vulnerability scanning

Microsoft Defender for Cloud provides unified security management and advanced threat protection across hybrid cloud workloads. For Azure machine learning, you should enable scanning of your [Azure Container Registry](#) resource and Azure Kubernetes Service resources. For more information, see [Azure Container Registry image scanning by Defender for Cloud](#) and [Azure Kubernetes Services integration with Defender for Cloud](#).

Audit and manage compliance

[Azure Policy](#) is a governance tool that allows you to ensure that Azure resources are compliant with your policies. You can set policies to allow or enforce specific configurations, such as whether your Azure Machine Learning workspace uses a private endpoint. For more information on Azure Policy, see the [Azure Policy documentation](#). For more information on the policies specific to Azure Machine Learning, see [Audit and manage compliance with Azure Policy](#).

Next steps

- [Azure Machine Learning best practices for enterprise security](#)
- [Use Azure Machine Learning with Azure Firewall](#)
- [Use Azure Machine Learning with Azure Virtual Network](#)
- [Data encryption at rest and in transit](#)
- [Build a real-time recommendation API on Azure](#)

Network traffic flow when using a secured workspace

Article • 02/24/2023 • 10 minutes to read

When your Azure Machine Learning workspace and associated resources are secured in an Azure Virtual Network, it changes the network traffic between resources. Without a virtual network, network traffic flows over the public internet or within an Azure data center. Once a virtual network (VNet) is introduced, you may also want to harden network security. For example, blocking inbound and outbound communications between the VNet and public internet. However, Azure Machine Learning requires access to some resources on the public internet. For example, Azure Resource Management is used for deployments and management operations.

This article lists the required traffic to/from the public internet. It also explains how network traffic flows between your client development environment and a secured Azure Machine Learning workspace in the following scenarios:

- Using Azure Machine Learning **studio** to work with:
 - Your workspace
 - AutoML
 - Designer
 - Datasets and datastores

💡 Tip

Azure Machine Learning studio is a web-based UI that runs partially in your web browser, and makes calls to Azure services to perform tasks such as training a model, using designer, or viewing datasets. Some of these calls use a different communication flow than if you are using the SDK, CLI, REST API, or VS Code.

- Using Azure Machine Learning **studio**, **SDK**, **CLI**, or **REST API** to work with:
 - Compute instances and clusters
 - Azure Kubernetes Service
 - Docker images managed by Azure Machine Learning

💡 Tip