Microsoft Certified: Azure AI Fundamentals

Mastering the basics in AI can help you jump-start your career and get ready to dive deeper into the other technical opportunities Azure offers. Artificial intelligence (AI) opens doors into possibilities that might have seemed like science fiction only yesterday. Using AI, you can build solutions, improve your apps and advance technology in many fields, including healthcare, financial management, and environmental protection, to name just a few.

The Microsoft Certified: Azure AI Fundamentals certification could be a great fit for you if you'd like to:

- Prove that you have the AI skills it takes to build a better world. Earning your **Azure AI Fundamentals** certification can supply the foundation you need to build your career and demonstrate your knowledge of common AI and machine learning workloads—and what Azure services can solve for them.
- Validate your foundational knowledge of machine learning and AI concepts, along with related Azure services.

It is recommended to have some general programming knowledge or experience, but it is not required. You can use Azure AI fundamentals to reinforce your basics for other Azure role-based certifications, like Azure Data Scientist Associate, Azure AI Engineer Associate, or Azure Developer Associate, but it’s not a prerequisite for any of them.

To ensure you are prepared for the exam, we recommend:

- Fully understanding the skills measured.
- Studying the relevant self-paced content on Microsoft Learn, or attending a Microsoft Azure Virtual Training Day: AI Fundamentals, or signing up for an instructor-led training event with a Microsoft Learning Partner.
- Taking the practice exam to validate your knowledge and understanding of the exam experience.
- Getting a trial subscription and giving it a try.
- Checking out Master the basics with the Azure AI Fundamentals certification to learn more about this certification and how to get ready.

What's next?

Register for your exam! After you pass the exam and earn your certification, celebrate your certification badge and skills on social platforms such as LinkedIn. To find out more, visit: use and share certification badges.

Depending on your goals, you may choose to master the basics with the Azure Data Fundamentals certification, level up with the Azure AI Engineer Associate, Azure Data Scientist Associate, or Azure Developer Associate certifications, or find the right Microsoft Azure certification for you, based on your profession (or the one you aspire to).

Important

The English language version of this certification was updated on April 29, 2022. Visit the **Exam AI-900 page** to download the study guide.

Job role: AI Engineer, Developer, Data Scientist, Student

FUNDAMENTALS CERTIFICATION

Microsoft Certified: Azure AI Fundamentals

Skills measured

- This list contains the skills measured on the exam associated with this certification. For more detailed information, visit the exam details page and download the study guide.
- Describe Artificial Intelligence workloads and considerations
- Describe fundamental principles of machine learning on Azure
- Describe features of computer vision workloads on Azure
- Describe features of Natural Language Processing (NLP) workloads on Azure

Part 1/6

Microsoft Azure AI Fundamentals: Get started with artificial intelligence

Part 1/6_1/1

Introduction to AI

AI enables us to build amazing software that can improve health care, enable people to overcome physical disadvantages, empower smart infrastructure, create incredible entertainment experiences, and even save the planet!

Watch the following video to see some ways that AI can be used.

What is AI?

Simply put, AI is the creation of software that imitates human behaviors and capabilities. Key workloads include:

- **Machine learning** - This is often the foundation for an AI system, and is the way we "teach" a computer model to make prediction and draw conclusions from data.
- **Anomaly detection** - The capability to automatically detect errors or unusual activity in a system.
- **Computer vision** - The capability of software to interpret the world visually through cameras, video, and images.
- **Natural language processing** - The capability for a computer to interpret written or spoken language, and respond in kind.
- **Knowledge mining** - The capability to extract information from large volumes of often unstructured data to create a searchable knowledge store.

Understand machine learning

Machine Learning is the foundation for most AI solutions.

Let's start by looking at a real-world example of how machine learning can be used to solve a difficult problem.

Sustainable farming techniques are essential to maximize food production while protecting a fragile environment. *The Yield*, an agricultural technology company based in Australia, uses sensors, data and machine learning to help farmers make informed decisions related to weather, soil and plant conditions.

View the following video to learn more.

You can find out more about how the Yield is using machine learning to feed the world without wrecking the planet here.

How machine learning works

So how do machines learn?

The answer is, from data. In today's world, we create huge volumes of data as we go about our everyday lives. From the text messages, emails, and social media posts we send to the photographs and videos we take on our phones, we generate massive amounts of information. More data still is created by millions of sensors in our homes, cars, cities, public transport infrastructure, and factories.

Data scientists can use all of that data to train machine learning models that can make predictions and inferences based on the relationships they find in the data.

For example, suppose an environmental conservation organization wants volunteers to identify and catalog different species of wildflower using a phone app. The following animation shows how machine learning can be used to enable this scenario.

1. A team of botanists and scientists collect data on wildflower samples.
2. The team labels the samples with the correct species.
3. The labeled data is processed using an algorithm that finds relationships between the features of the samples and the labeled species.

4. The results of the algorithm are encapsulated in a model.
5. When new samples are found by volunteers, the model can identify the correct species label.

Machine learning in Microsoft Azure

Microsoft Azure provides the **Azure Machine Learning** service - a cloud-based platform for creating, managing, and publishing machine learning models. Azure Machine Learning provides the following features and capabilities:

Feature	Capability
Automated machine learning	This feature enables non-experts to quickly create an effective machine learning model from data.
Azure Machine Learning designer	A graphical interface enabling no-code development of machine learning solutions.
Data and compute management	Cloud-based data storage and compute resources that professional data scientists can use to run data experiment code at scale.
Pipelines	Data scientists, software engineers, and IT operations professionals can define pipelines to orchestrate model training, deployment, and management tasks.

Understand anomaly detection

Imagine you're creating a software system to monitor credit card transactions and detect unusual usage patterns that might indicate fraud. Or an application that tracks activity in an automated production line and identifies failures. Or a racing car telemetry system that uses sensors to proactively warn engineers about potential mechanical failures before they happen.

These kinds of scenario can be addressed by using *anomaly detection* - a machine learning based technique that analyzes data over time and identifies unusual changes.

Let's explore how anomaly detection might help in the racing car scenario.

1. Sensors in the car collect telemetry, such as engine revolutions, brake temperature, and so on.
2. An anomaly detection model is trained to understand expected fluctuations in the telemetry measurements over time.
3. If a measurement occurs outside of the normal expected range, the model reports an anomaly that can be used to alert the race engineer to call the driver in for a pit stop to fix the issue before it forces retirement from the race.

Anomaly detection in Microsoft Azure

In Microsoft Azure, the **Anomaly Detector** service provides an application programming interface (API) that developers can use to create anomaly detection solutions.

To learn more, view the Anomaly Detector service web site.

Understand computer vision

Computer Vision is an area of AI that deals with visual processing. Let's explore some of the possibilities that computer vision brings.

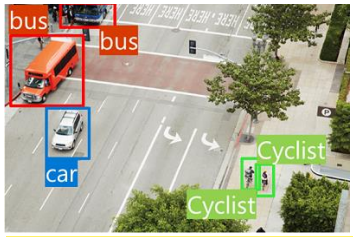
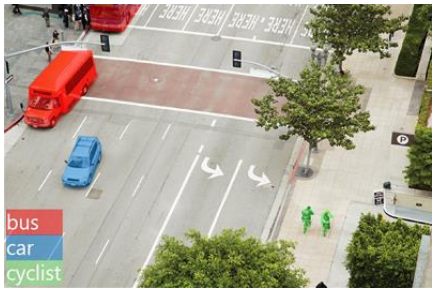
The **Seeing AI** app is a great example of the power of computer vision. Designed for the blind and low vision community, the Seeing AI app harnesses the power of AI to open up the visual world and describe nearby people, text and objects.

View the following video to learn more about Seeing AI.

To find out more, check out the Seeing AI web page.

Computer Vision models and capabilities

Most computer vision solutions are based on machine learning models that can be applied to visual input from cameras, videos, or images. The following table describes common computer vision tasks.

Task	Description
Image classification	 Image classification involves training a machine learning model to classify images based on their contents. For example, in a traffic monitoring solution you might use an image classification model to classify images based on the type of vehicle they contain, such as taxis, buses, cyclists, and so on.
Object detection	 Object detection machine learning models are trained to classify individual objects within an image, and identify their location with a bounding box. For example, a traffic monitoring solution might use object detection to identify the location of different classes of vehicle.
Semantic segmentation	 Semantic segmentation is an advanced machine learning technique in which individual pixels in the image are classified according to the object to which they belong. For example, a traffic monitoring solution might overlay traffic images with "mask" layers to highlight different vehicles using specific colors.
Image analysis	 You can create solutions that combine machine learning models with advanced image analysis techniques to extract information from images, including "tags" that could help catalog the image or even descriptive captions that summarize the scene shown in the image.
Face detection, analysis, and recognition	 Face detection is a specialized form of object detection that locates human faces in an image. This can be combined with classification and facial geometry analysis techniques to infer details such as age and emotional state; and even recognize individuals based on their facial features.

Task	Description
Optical character recognition (OCR)	 Optical character recognition is a technique used to detect and read text in images. You can use OCR to read text in photographs (for example, road signs or store fronts) or to extract information from scanned documents such as letters, invoices, or forms.

Computer vision services in Microsoft Azure

Microsoft Azure provides the following cognitive services to help you create computer vision solutions:

Service	Capabilities
Computer Vision	You can use this service to analyze images and video, and extract descriptions, tags, objects, and text.
Custom Vision	Use this service to train custom image classification and object detection models using your own images.
Face	The Face service enables you to build face detection and facial recognition solutions.
Form Recognizer	Use this service to extract information from scanned forms and invoices.

Try this

To see an example of a how computer vision can be used to analyze images, follow these steps:

1. Open another browser tab and go to https://aidemos.microsoft.com/computer-vision.
2. Use the demo interface to try each of the steps. For each step, you can select images and review the information returned by the Computer Vision service.

Understand natural language processing

Natural language processing (NLP) is the area of AI that deals with creating software that understands written and spoken language.

NLP enables you to create software that can:

- Analyze and interpret text in documents, email messages, and other sources.
- Interpret spoken language, and synthesize speech responses.
- Automatically translate spoken or written phrases between languages.
- Interpret commands and determine appropriate actions.

For example, *Starship Commander*, is a virtual reality (VR) game from Human Interact, that takes place in a science fiction world. The game uses natural language processing to enable players to control the narrative and interact with in-game characters and starship systems.

Watch the following video to learn more.

Natural language processing in Microsoft Azure

In Microsoft Azure, you can use the following cognitive services to build natural language processing solutions:

Service	Capabilities
Language	Use this service to access features for understanding and analyzing text, training language models that can understand spoken or text-based commands, and building intelligent applications.
Translator	Use this service to translate text between more than 60 languages.
Speech	Use this service to recognize and synthesize speech, and to translate spoken languages.
Azure Bot	This service provides a platform for conversational AI, the capability of a software "agent" to participate in a conversation. Developers can use the *Bot Framework* to create a bot and manage it with Azure Bot Service - integrating back-end services like Language, and connecting to channels for web chat, email, Microsoft Teams, and others.

Try this

To see an example of a how you can use natural language to interact with an AI system, follow these steps:

1. Open another browser tab and go to https://aidemos.microsoft.com/luis/demo.
2. Use the demo interface to control the lighting in the virtual home. You can type instructions, use the microphone button to speak commands, or select any of the suggested phrases to see how the system responds.

Understand knowledge mining

Knowledge mining is the term used to describe solutions that involve extracting information from large volumes of often unstructured data to create a searchable knowledge store.

Knowledge mining in Microsoft Azure

One of these knowledge mining solutions is **Azure Cognitive Search**, a private, enterprise, search solution that has tools for building indexes. The indexes can then be used for internal only use, or to enable searchable content on public facing internet assets.

Azure Cognitive Search can utilize the built-in AI capabilities of Azure Cognitive Services such as image processing, content extraction, and natural language processing to perform knowledge mining of documents. The product's AI capabilities makes it possible to index previously unsearchable documents and to extract and surface insights from large amounts of data quickly.

Challenges and risks with AI

Artificial Intelligence is a powerful tool that can be used to greatly benefit the world. However, like any tool, it must be used responsibly.

The following table shows some of the potential challenges and risks facing an AI application developer.

Challenge or Risk	Example
Bias can affect results	A loan-approval model discriminates by gender due to bias in the data with which it was trained
Errors may cause harm	An autonomous vehicle experiences a system failure and causes a collision
Data could be exposed	A medical diagnostic bot is trained using sensitive patient data, which is stored insecurely
Solutions may not work for everyone	A home automation assistant provides no audio output for visually impaired users
Users must trust a complex system	An AI-based financial tool makes investment recommendations - what are they based on?
Who's liable for AI-driven decisions?	An innocent person is convicted of a crime based on evidence from facial recognition – who's responsible?

Understand responsible AI

At Microsoft, AI software development is guided by a set of six principles, designed to ensure that AI applications provide amazing solutions to difficult problems without any unintended negative consequences.

Fairness

AI systems should treat all people fairly. For example, suppose you create a machine learning model to support a loan approval application for a bank. The model should make predictions of whether or not the loan should be approved without incorporating any bias based on gender, ethnicity, or other factors that might result in an unfair advantage or disadvantage to specific groups of applicants.

Azure Machine Learning includes the capability to interpret models and quantify the extent to which each feature of the data influences the model's prediction. This capability helps data scientists and developers identify and mitigate bias in the model.

For more details about considerations for fairness, watch the following video.

Reliability and safety

AI systems should perform reliably and safely. For example, consider an AI-based software system for an autonomous vehicle; or a machine learning model that diagnoses patient symptoms and recommends prescriptions. Unreliability in these kinds of systems can result in substantial risk to human life.

AI-based software application development must be subjected to rigorous testing and deployment management processes to ensure that they work as expected before release.

For more information about considerations for reliability and safety, watch the following video.

Privacy and security

AI systems should be secure and respect privacy. The machine learning models on which AI systems are based rely on large volumes of data, which may contain personal details that must be kept private. Even after the models are trained and the system is in production, it uses new data to make predictions or take action that may be subject to privacy or security concerns.

For more details about considerations for privacy and security, watch the following video.

Inclusiveness

AI systems should empower everyone and engage people. AI should bring benefits to all parts of society, regardless of physical ability, gender, sexual orientation, ethnicity, or other factors.

For more details about considerations for inclusiveness, watch the following video.

Transparency

AI systems should be understandable. Users should be made fully aware of the purpose of the system, how it works, and what limitations may be expected.

For more details about considerations for transparency, watch the following video.

Accountability

People should be accountable for AI systems. Designers and developers of AI-based solutions should work within a framework of governance and organizational principles that ensure the solution meets ethical and legal standards that are clearly defined.

For more details about considerations for accountability, watch the following video.

The principles of responsible AI can help you understand some of the challenges facing developers as they try to create ethical AI solutions.

Further resources

For more resources to help you put the responsible AI principles into practice, see https://www.microsoft.com/ai/responsible-ai-resources.

Part 2

[Microsoft Azure AI Fundamentals: Explore visual tools for machine learning]

Part 2/6_1/4

Use automated machine learning in Azure Machine Learning

Introduction

Machine Learning is the foundation for most artificial intelligence solutions, and the creation of an intelligent solution often begins with the use of machine learning to train a predictive model using historic data that you have collected.

Azure Machine Learning is a cloud service that you can use to train and manage machine learning models.

In this module, you'll learn how to:

- Identify different kinds of machine learning models.
- Use the automated machine learning capability of Azure Machine Learning to train and deploy a predictive model.

To complete this module, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.microsoft.com.

What is machine learning?

Machine learning is a technique that uses mathematics and statistics to create a model that can predict unknown values.

For example, suppose *Adventure Works Cycles* is a business that rents cycles in a city. The business could use historic data to train a model that predicts daily rental demand in order to make sure sufficient staff and cycles are available.

To do this, Adventure Works could create a machine learning model that takes information about a specific day (the day of week, the anticipated weather conditions, and so on) as an input, and predicts the expected number of rentals as an output.

Mathematically, you can think of machine learning as a way of defining a function (let's call it ***f***) that operates on one or more *features* of something (which we'll call ***x***) to calculate a predicted *label* (***y***) - like this:

$$f(x) = y$$

In this bicycle rental example, the details about a given day (day of the week, weather, and so on) are the *features* (***x***), the number of rentals for that day is the *label* (***y***), and the function (***f***) that calculates the number of rentals based on the information about the day is encapsulated in a machine learning model.

The specific operation that the ***f*** function performs on *x* to calculate *y* depends on a number of factors, including the type of model you're trying to create and the specific algorithm used to train the model. Additionally in most cases, the data used to train the machine learning model requires some pre-processing before model training can be performed.

The following video discusses the various kinds of machine learning model you can create, and the process generally followed to train and use them.

Azure Machine Learning

Training and deploying an effective machine learning model involves a lot of work, much of it time-consuming and resource-intensive. Azure Machine Learning is a cloud-based service that helps simplify some of the tasks and reduce the time it takes to prepare data, train a model, and deploy a predictive service. In the rest of this unit, you'll explore Azure Machine Learning, and in particular its *automated machine learning* capability.

∞

Create an Azure Machine Learning workspace

Data scientists expend a lot of effort exploring and pre-processing data, and trying various types of model-training algorithms to produce accurate models, which is time consuming, and often makes inefficient use of expensive compute hardware.

Azure Machine Learning is a cloud-based platform for building and operating machine learning solutions in Azure. It includes a wide range of features and capabilities that help data scientists prepare data, train models, publish predictive services, and monitor their usage. Most importantly, it helps data scientists increase their efficiency by automating many of the time-consuming tasks associated with training models; and it enables them to use cloud-based compute resources that scale effectively to handle large volumes of data while incurring costs only when actually used.

Create an Azure Machine Learning workspace

To use Azure Machine Learning, you create a *workspace* in your Azure subscription. You can then use this workspace to manage data, compute resources, code, models, and other artifacts related to your machine learning workloads.

Note

This module is one of many that make use of an Azure Machine Learning workspace, including the other modules in the **Microsoft Azure AI Fundamentals: Explore visual tools for machine learning** learning path. If you are using your own Azure subscription, you may consider creating the workspace once and reusing it in other modules. Your Azure subscription will be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription, so we recommend you delete the Azure Machine Learning workspace when it is no longer required.

If you do not already have one, follow these steps to create a workspace:

1. Sign into the Azure portal using your Microsoft credentials.
2. Select **+Create a resource**, search for *Machine Learning*, and create a new **Azure Machine Learning** resource with an *Azure Machine Learning* plan. Use the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Workspace name**: *Enter a unique name for your workspace*
    - **Region**: *Select the closest geographical region*
    - **Storage account**: *Note the default new storage account that will be created for your workspace*
    - **Key vault**: *Note the default new key vault that will be created for your workspace*
    - **Application insights**: *Note the default new application insights resource that will be created for your workspace*
    - **Container registry**: None (*one will be created automatically the first time you deploy a model to a container*)
3. Select **Review + create**. Wait for your workspace to be created (it can take a few minutes). Then go to it in the portal.
4. On the **Overview** page for your workspace, launch Azure Machine Learning studio (or open a new browser tab and navigate to https://ml.azure.com), and sign into Azure Machine Learning studio using your Microsoft account.
5. In Azure Machine Learning studio, select the three lines at the top left to view the various pages in the interface. You can use these pages to manage the resources in your workspace.

You can manage your workspace using the Azure portal, but for data scientists and Machine Learning operations engineers, Azure Machine Learning studio provides a more focused user interface for managing workspace resources.

Create compute resources

After you have created an Azure Machine Learning workspace, you can use it to manage the various assets and resources you need to create machine learning solutions. At its core, Azure Machine Learning is a platform for training and managing machine learning models, for which you need compute on which to run the training process.

Create a compute cluster

Compute targets are cloud-based resources on which you can run model training and data exploration processes.

In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Compute** page (under **Manage**). This is where you manage the compute targets for your data science activities. There are four kinds of compute resource you can create:

- **Compute Instances**: Development workstations that data scientists can use to work with data and models.
- **Compute Clusters**: Scalable clusters of virtual machines for on-demand processing of experiment code.
- **Inference Clusters**: Deployment targets for predictive services that use your trained models.
- **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

Note

Compute instances and clusters are based on standard Azure virtual machine images. For this module, the *Standard_DS11_v2* image is recommended to achieve the optimal balance of cost and performance. If your subscription has a quota that does not include this image, choose an alternative image; but bear in mind that a larger image may incur higher cost and a smaller image may not be sufficient to complete the tasks. Alternatively, ask your Azure administrator to extend your quota.

1. Select the **Compute Clusters** tab, and add a new compute cluster with the following settings. You'll use this to train a machine learning model:
    - **Location**: *Select the same as your workspace. If that location is not listed, choose the one closest to you*
    - **Virtual Machine tier**: Dedicated
    - **Virtual Machine type**: CPU
    - **Virtual Machine size**:
        - Choose **Select from all options**
        - Search for and select **Standard_DS11_v2**
    - Select **Next**
    - **Compute name**: *enter a unique name*
    - **Minimum number of nodes**: 0
    - **Maximum number of nodes**: 2
    - **Idle seconds before scale down**: 120
    - **Enable SSH access**: Unselected
    - Select **Create**

Tip

After you finish the entire module, be sure to follow the **Clean Up** instructions at the end of the module to stop your compute resources. Stop your compute resources to ensure your subscription won't be charged.

The compute cluster will take some time to be created. You can move onto the next unit while you wait.

Explore data

Machine learning models must be trained with existing data. In this case, you'll use a dataset of historical bicycle rental details to train a model that predicts the number of bicycle rentals that should be expected on a given day, based on seasonal and meteorological features.

Create a dataset

In Azure Machine Learning, data for model training and other operations is usually encapsulated in an object called a *dataset*.

1. View the comma-separated data at https://aka.ms/bike-rentals in your web browser.
2. In Azure Machine Learning studio,expand the left pane by selecting the three lines at the top left of the screen. View the **Data** page (under **Assets**). The Data page contains specific data files or tables that you plan to work with in Azure ML. You can create datasets from this page as well.
3. Create a new dataset **from web files**, using the following settings:
    - **Basic Info**:
        - **Web URL**: https://aka.ms/bike-rentals
        - **Name**: bike-rentals
        - **Dataset type**: Tabular
        - **Description**: Bicycle rental data
        - **Skip data validation**: *do not select*
    - **Settings and preview**:
        - **File format**: Delimited
        - **Delimiter**: Comma
        - **Encoding**: UTF-8
        - **Column headers**: Only first file has headers
        - **Skip rows**: None
        - **Dataset contains multi-line data**: *do not select*
    - **Schema**:
        - Include all columns other than **Path**
        - Review the automatically detected types
    - **Confirm details**:
        - Do not profile the dataset after creation
4. After the dataset has been created, open it and view the **Explore** page to see a sample of the data. This data contains historical features and labels for bike rentals.

Citation: *This data is derived from Capital Bikeshare and is used in accordance with the published data license agreement*.

Train a machine learning model

Azure Machine Learning includes an *automated machine learning* capability that automatically tries multiple pre-processing techniques and model-training algorithms in parallel. These automated capabilities use the power of cloud compute to find the best performing supervised machine learning model for your data.

Note

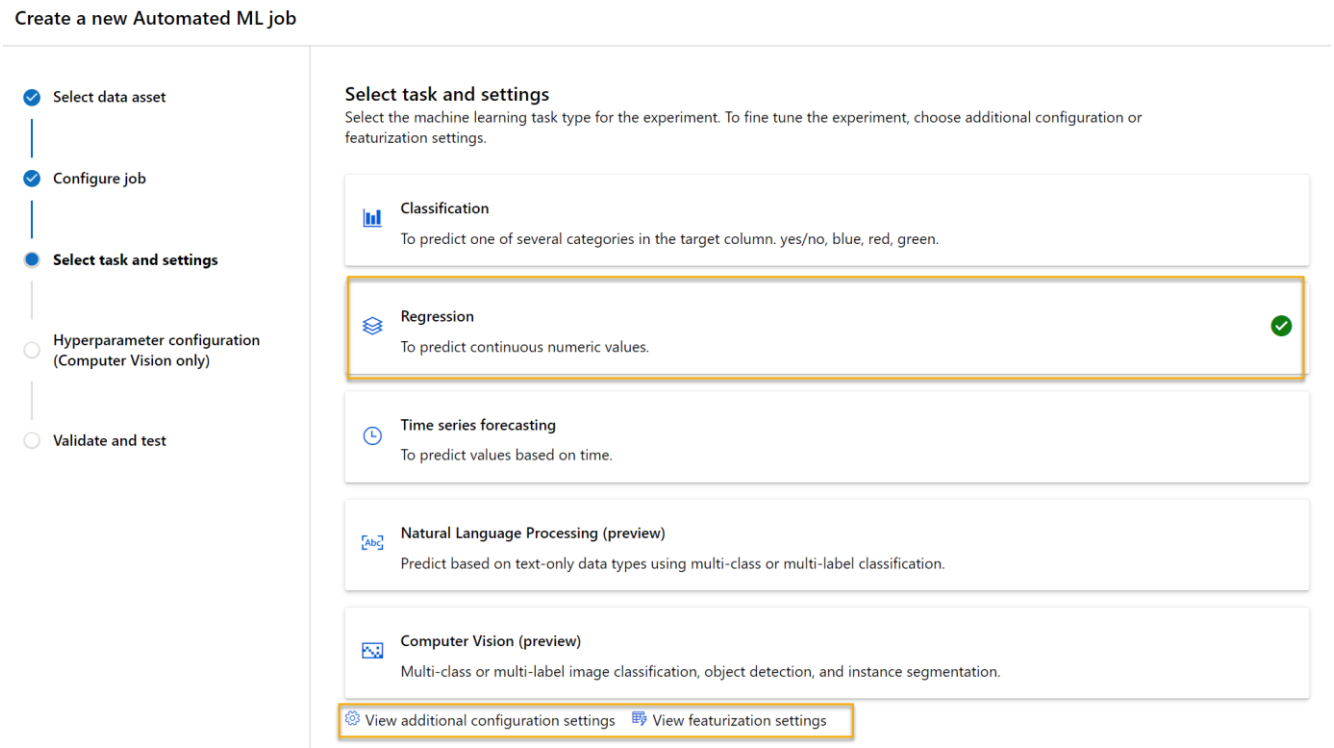
The automated machine learning capability in Azure Machine Learning supports *supervised* machine learning models - in other words, models for which the training data includes known label values. You can use automated machine learning to train models for:

- **Classification** (predicting categories or *classes*)
- **Regression** (predicting numeric values)
- **Time series forecasting** (predicting numeric values at a future point in time)

Run an automated machine learning experiment

In Azure Machine Learning, operations that you run are called *experiments*. Follow the steps to run an experiment that uses automated machine learning to train a regression model that predicts bicycle rentals.

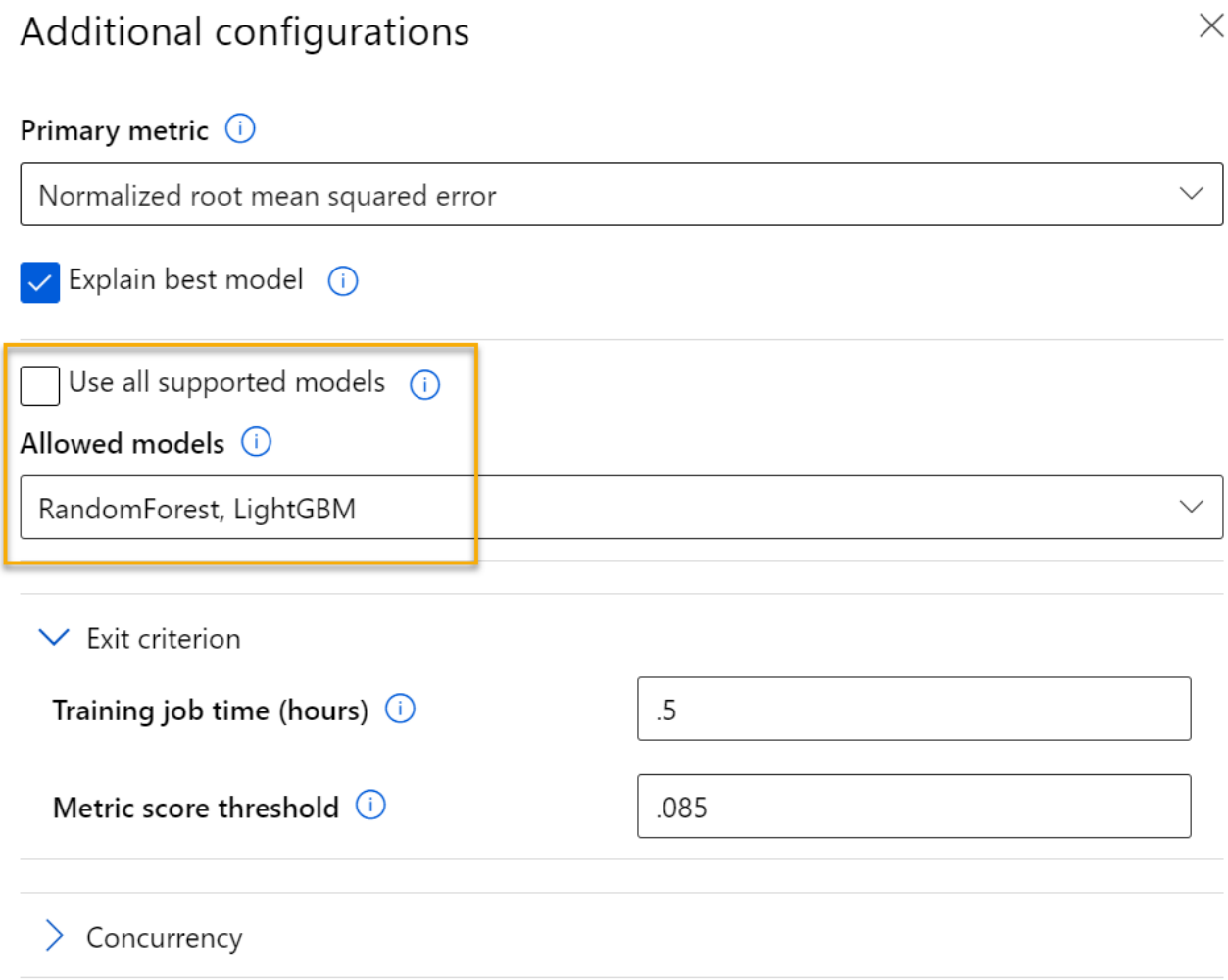
1. In Azure Machine Learning studio, view the **Automated ML** page (under **Author**).
2. Create an Automated ML run with the following settings:
    - **Select dataset**:
        - **Dataset**: bike-rentals
    - **Configure run**:
        - **New experiment name**: mslearn-bike-rental



        - **Target column**: rentals (*this is the label that the model is trained to predict)*
        - **Select compute cluster**: *the compute cluster that you created previously*
    - **Select task and settings**:
        - **Task type**: Regression *(the model predicts a numeric value)*

    Notice under task type there are settings *View additional configuration settings* and *View Featurization settings*. Now configure these settings.

    - **Additional configuration settings:**
        - **Primary metric**: Select **Normalized root mean squared error** *(more about this metric later!)*
        - **Explain best model**: Selected — *this option causes automated machine learning to calculate feature importance for the best model which makes it possible to determine the influence of each feature on the predicted label.*



- **Use all supported models**: Unselected. *You'll restrict the experiment to try only a few specific algorithms.*
- **Allowed models**: *Select only* ***RandomForest*** *and* ***LightGBM*** *— normally you'd want to try as many as possible, but each model added increases the time it takes to run the experiment.*
- **Exit criterion**:
  - **Training job time (hours)**: 0.5 — *ends the experiment after a maximum of 30 minutes.*
  - **Metric score threshold**: 0.085 — *if a model achieves a normalized root mean squared error metric score of 0.085 or less, the experiment ends.*
- **Concurrency**: *do not change*

- **Featurization settings:**
  - **Enable featurization**: Selected — *automatically preprocess the features before training.*

Click **Next** to go to the next selection pane.

- **[Optional] Select the validation and test type**
  - **Validation type**: Auto
  - **Test dataset (preview)**: No test dataset required

3. When you finish submitting the automated ML run details, it starts automatically. Wait for the run status to change from *Preparing* to *Running*.

4. When the run status changes to *Running*, view the **Models** tab and observe as each possible combination of training algorithm and pre-processing steps is tried and the performance of the resulting model is evaluated. The page automatically refreshes periodically, but you can also select ↻ **Refresh**. It might take 10 minutes or so before models start to appear, as the cluster nodes must be initialized before training can begin.
5. Wait for the experiment to finish. It might take a while — now might be a good time for a coffee break!

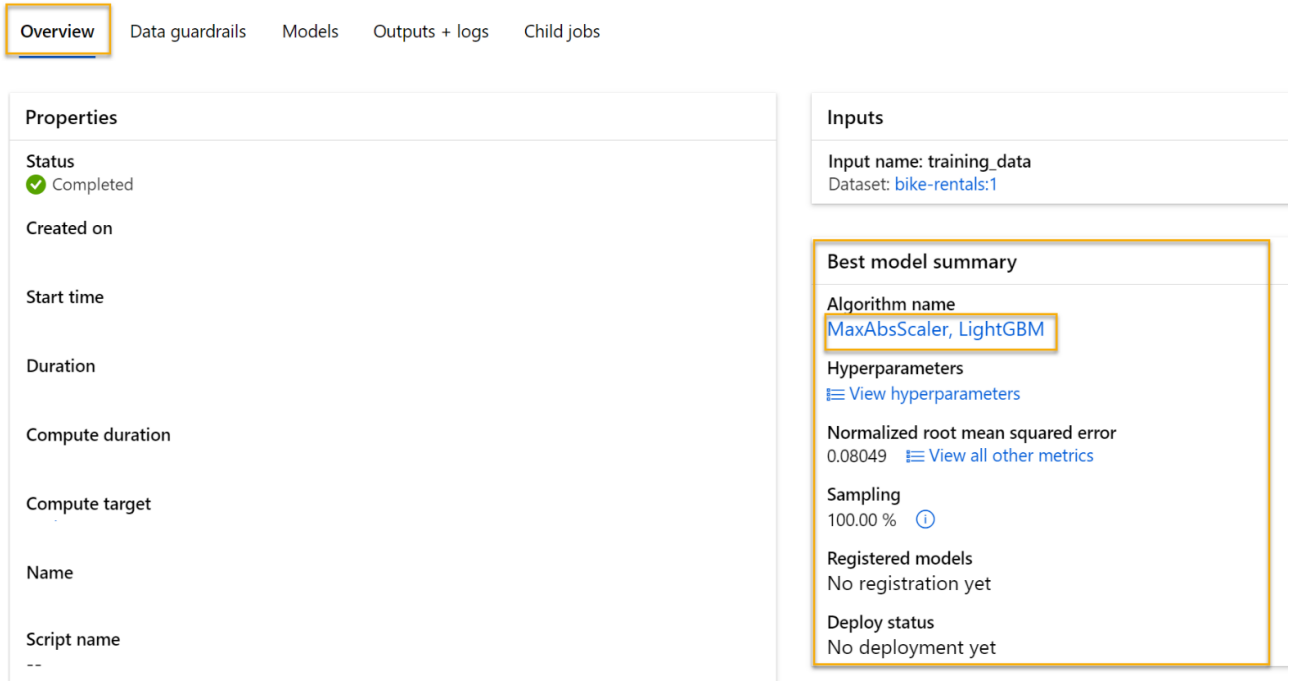
Review the best model

After the experiment has finished you can review the best performing model. In this case, you used exit criteria to stop the experiment. Thus the "best" model the experiment generated might not be the best possible model, just the best one found within the time allowed for this exercise.

1. On the **Overview** tab of the automated machine learning run, note the best model summary.

2. Select the text under **Algorithm name** for the best model to view its details.

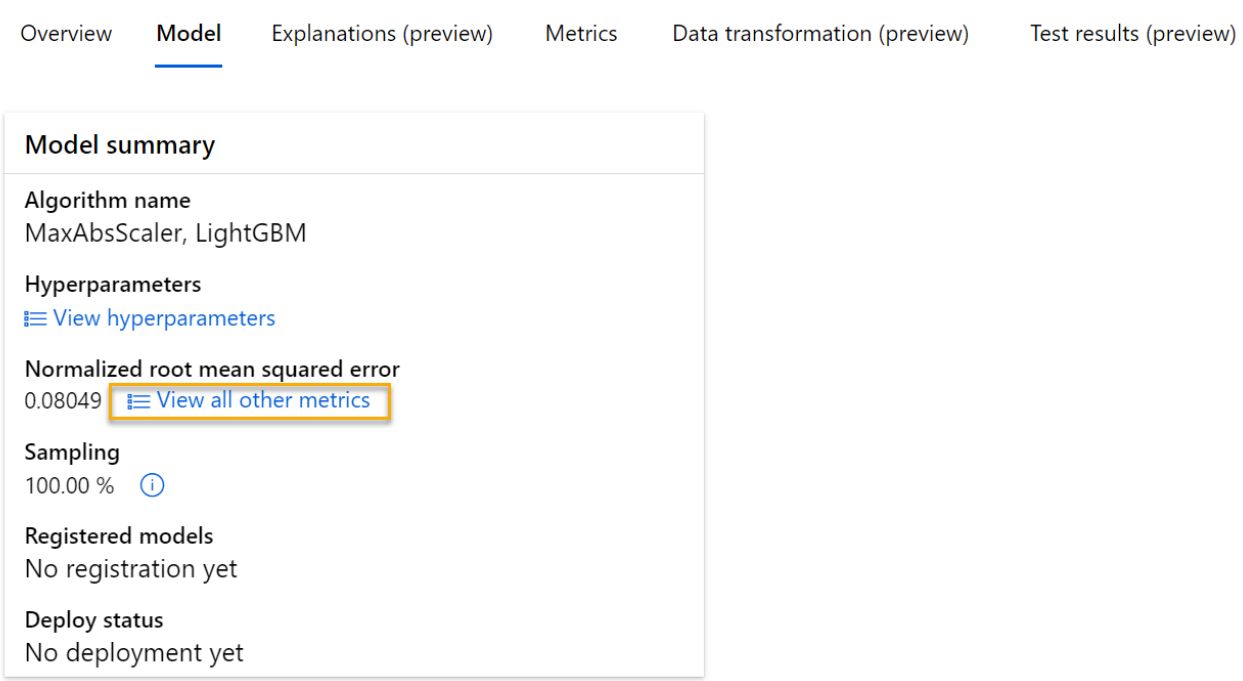
   The best model is identified based on the evaluation metric you specified, *Normalized root mean squared error*.



   A technique called *cross-validation* is used to calculate the evaluation metric. After the model is trained using a portion of the data, the remaining portion is used to iteratively test, or cross-validate, the trained model. The metric is calculated by comparing the predicted value from the test with the actual known value, or label.

   The difference between the predicted and actual value, known as the *residuals*, indicates the amount of *error* in the model. The particular performance metric you used, normalized root mean squared error, is calculated by squaring the errors across all of the test cases, finding the mean of these squares, and then taking the square root. What all of this means is that smaller this value is, the more accurate the model's predictions.

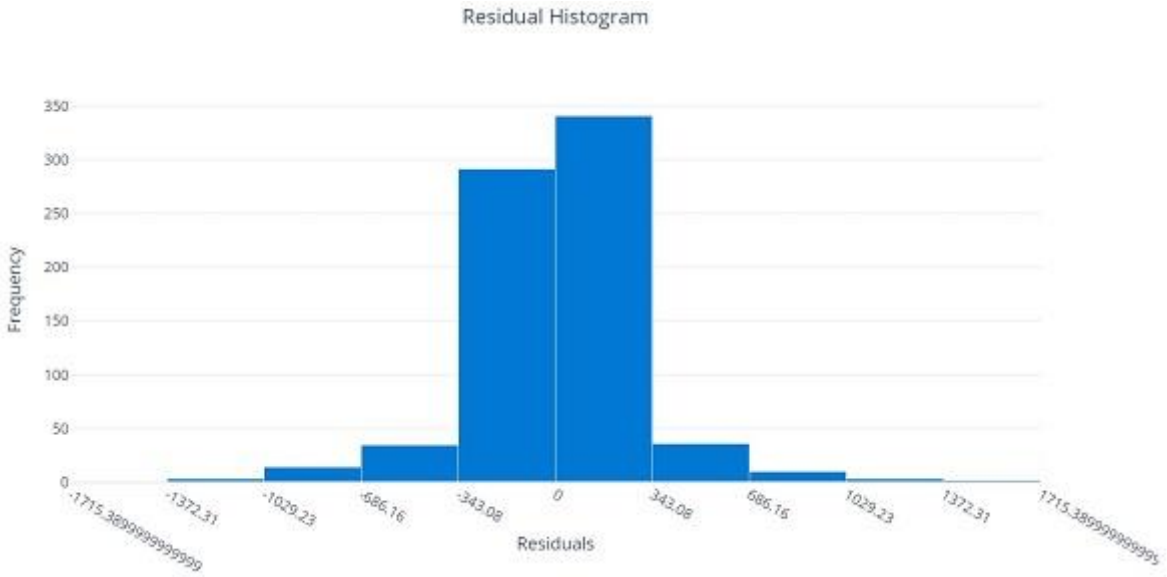
3. Next to the *Normalized root mean squared error* value, select **View all other metrics** to see values of other possible evaluation metrics for a regression model.



4. Select the **Metrics** tab and select the **residuals** and **predicted_true** charts if they are not already selected.

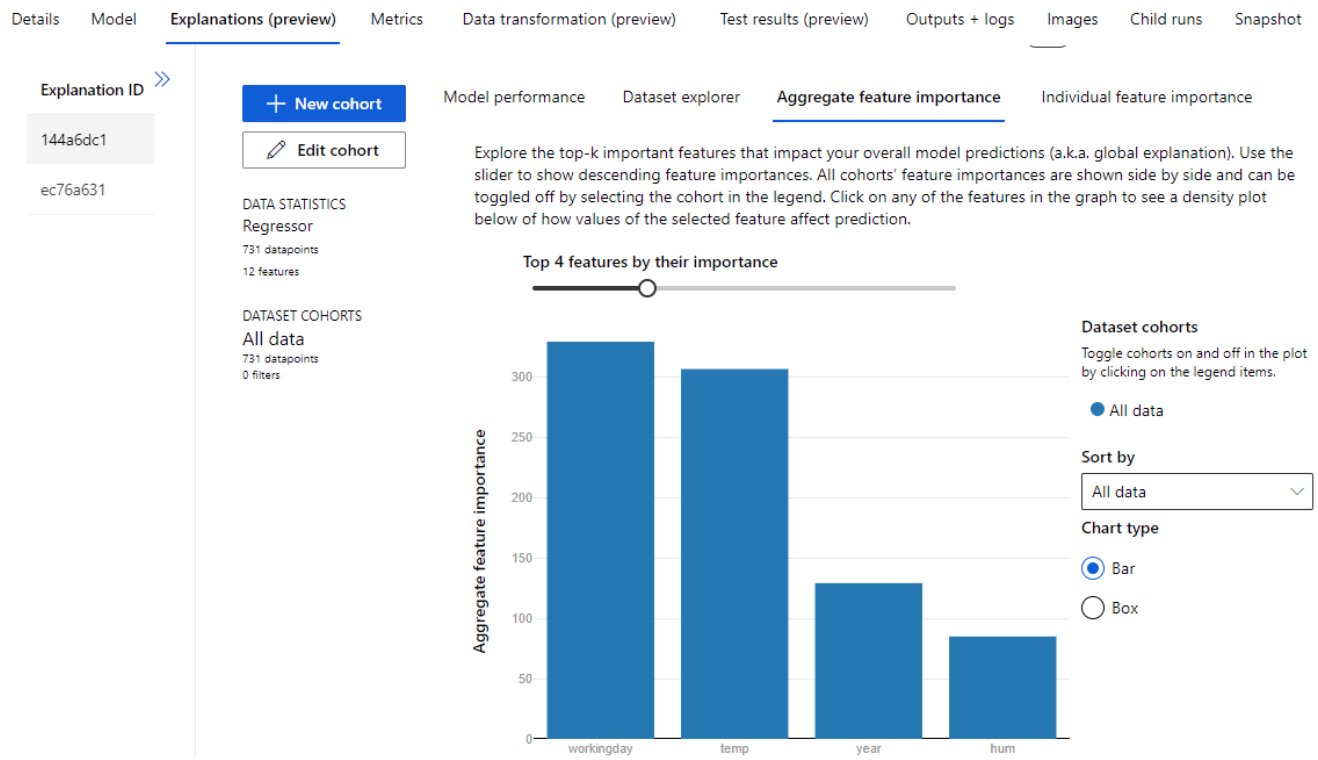
Review the charts which show the performance of the model. The first chart shows the *residuals*, the differences between predicted and actual values, as a histogram, the second chart compares the predicted values against the true values.

The **Residual Histogram** shows the frequency of residual value ranges. Residuals represent variance between predicted and true values that can't be explained by the model, in other words, errors. You should hope to see the most frequently occurring residual values clustered around zero. You want small errors with fewer errors at the extreme ends of the scale.



The **Predicted vs. True** chart should show a diagonal trend in which the predicted value correlates closely to the true value. The dotted line shows how a perfect model should perform. The closer the line of your model's average predicted value is to the dotted line, the better its performance. A histogram below the line chart shows the distribution of true values.

5. Select the **Explanations** tab. Select an explanation ID and then select **Aggregate feature importance**. This chart shows how much each feature in the dataset influences the label prediction, like this:



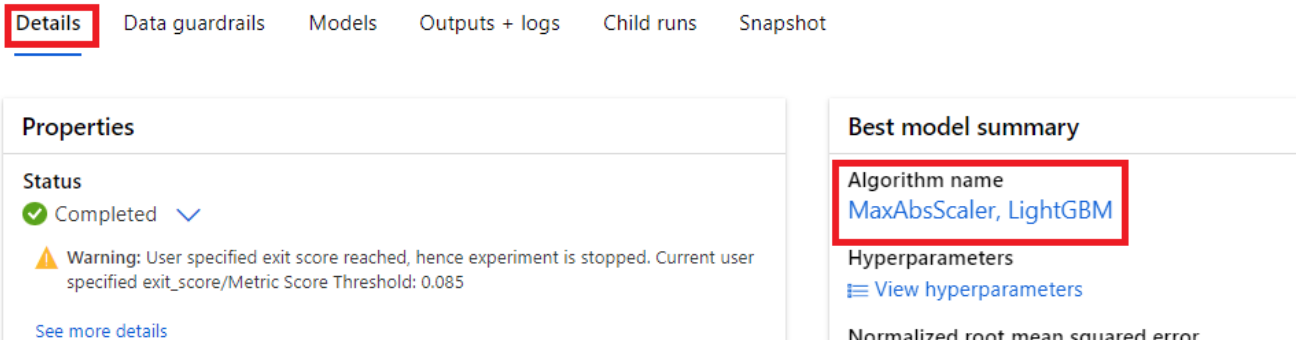
Deploy a model as a service

After you've used automated machine learning to train some models, you can deploy the best performing model as a service for client applications to use.

Deploy a predictive service

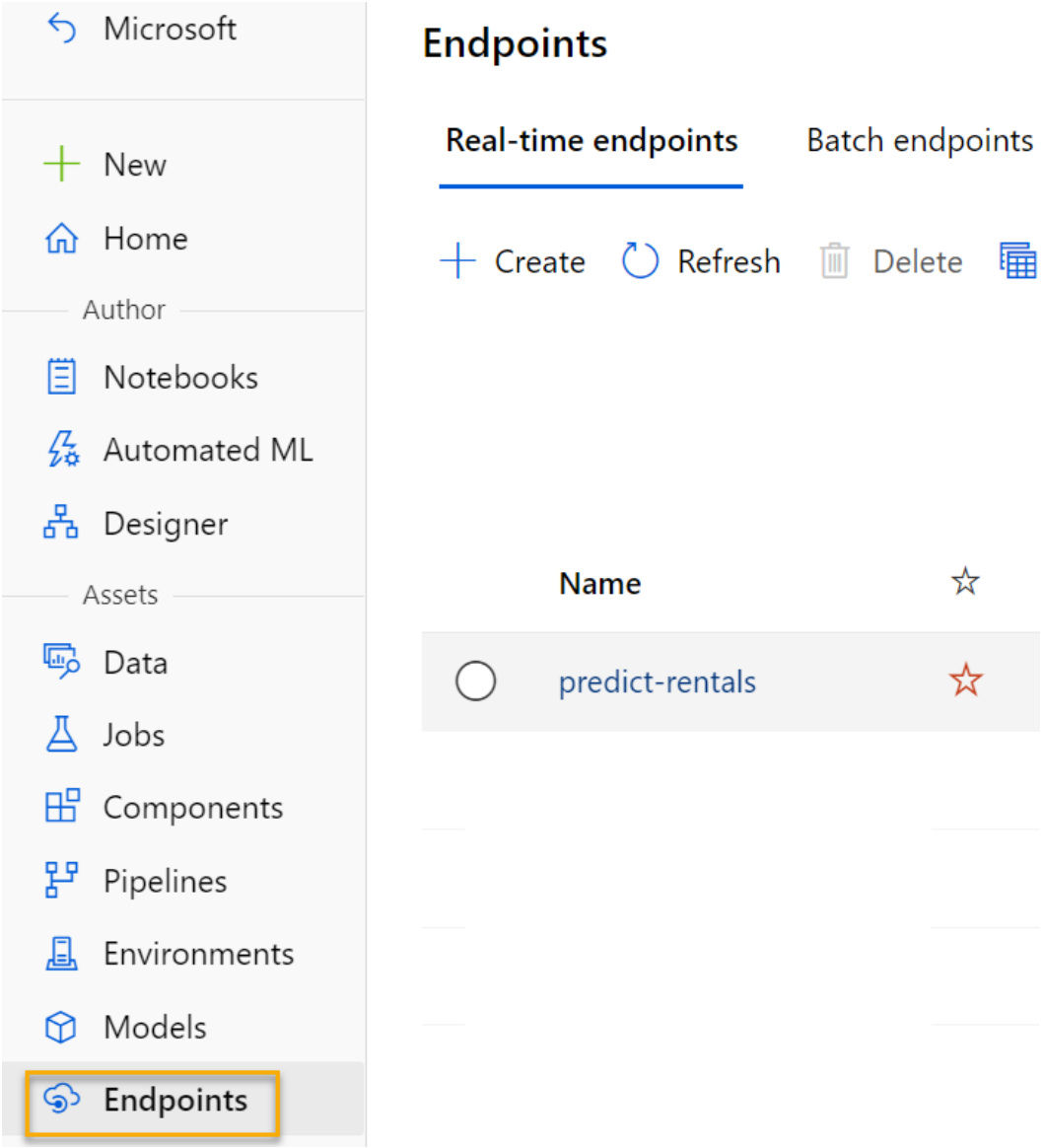
In Azure Machine Learning, you can deploy a service as an Azure Container Instances (ACI) or to an Azure Kubernetes Service (AKS) cluster. For production scenarios, an AKS deployment is recommended, for which you must create an *inference cluster* compute target. In this exercise, you'll use an ACI service, which is a suitable deployment target for testing, and does not require you to create an inference cluster.

1. In Azure Machine Learning studio, on the **Automated ML** page, select the run for your automated machine learning experiment.
2. On the **Details** tab, select the algorithm name for the best model.



3. on the **Model** tab, select the **Deploy** button and use the **Deploy to web service** option to deploy the model with the following settings:
    - **Name**: predict-rentals
    - **Description**: Predict cycle rentals
    - **Compute type**: Azure Container Instance
    - **Enable authentication**: Selected

4. Wait for the deployment to start - this may take a few seconds. Then, in the **Model summary** section, observe the **Deploy status** for the **predict-rentals** service, which should be **Running**. Wait for this status to change to **Successful**, which may take some time. You may need to select **Refresh** periodically.
5. In Azure Machine Learning studio, on the left hand menu, select **Endpoints**.



Test the deployed service

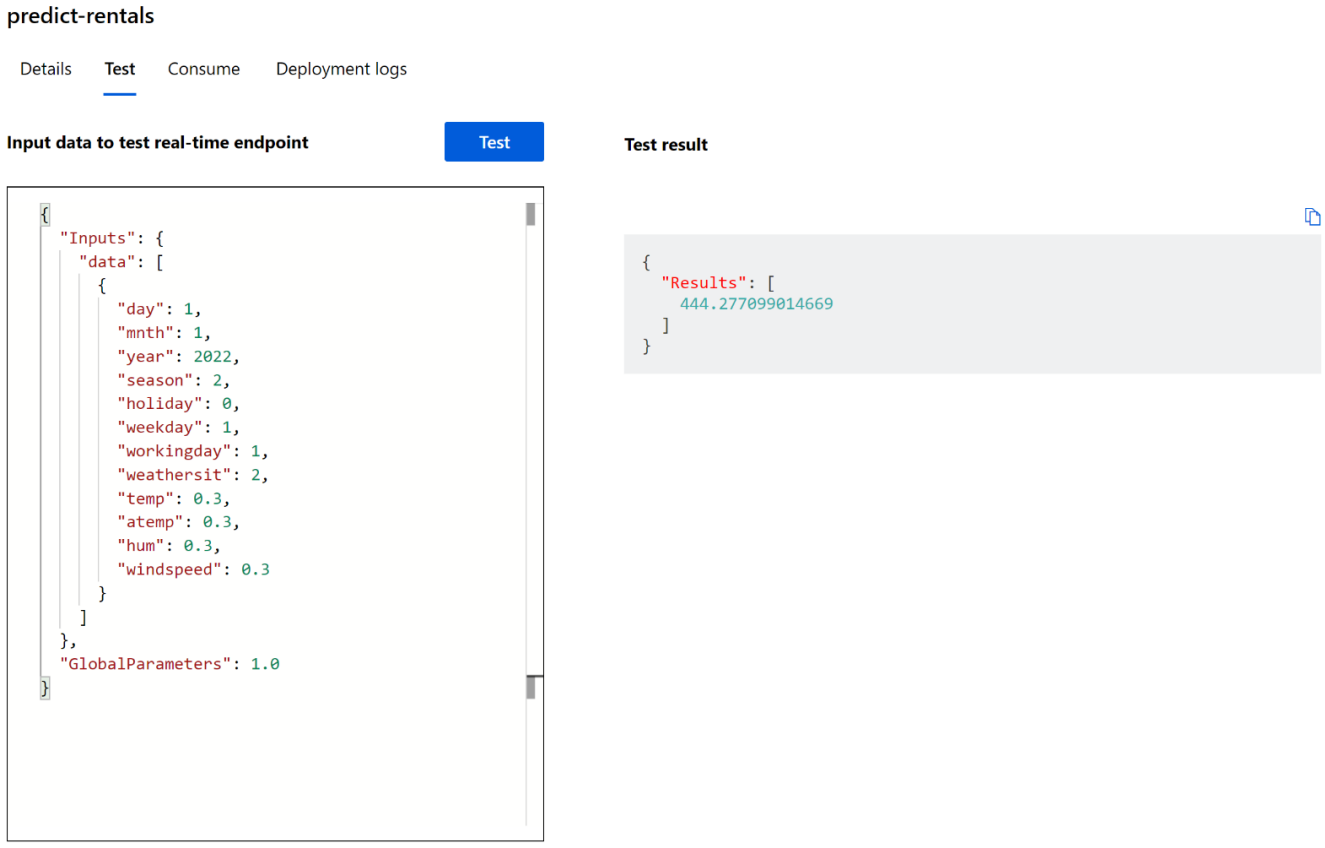
Now you can test your deployed service.

1. On the **Endpoints** page, open the **predict-rentals** real-time endpoint.
2. When the **predict-rentals** endpoint opens, view the **Test** tab.
3. In the input data pane, replace the template JSON with the following input data:

```
JSONCopy
{
  "Inputs": {
    "data": [
      {
        "day": 1,
        "mnth": 1,
        "year": 2022,
```

```
        "season": 2,
        "holiday": 0,
        "weekday": 1,
        "workingday": 1,
        "weathersit": 2,
        "temp": 0.3,
        "atemp": 0.3,
        "hum": 0.3,
        "windspeed": 0.3
      }
    ]
  },
  "GlobalParameters": 1.0
}
```

4. Click on the **Test** button.
5. Review the test results, which include a predicted number of rentals based on the input features. The test pane took the input data and used the model you trained to return the predicted number of rentals.



Let's review what you have done. You used a dataset of historical bicycle rental data to train a model. The model predicts the number of bicycle rentals expected on a given day, based on seasonal and meteorological *features*. In this case, the *labels* are number of bicycle rentals.

You have just tested a service that is ready to be connected to a client application using the credentials in the **Consume** tab. We will end the lab here. You are welcome to continue to experiment with the service you just deployed.

Knowledge check

1. An automobile dealership wants to use historic car sales data to train a machine learning model. The model should predict the price of a pre-owned car based on its make, model, engine size, and mileage. What kind of machine learning model should the dealership use automated machine learning to create?

- ○ Classification
- ◉ Regression **Correct. To predict a numeric value, use a regression model.**
- ○ Time series forecasting

2. A bank wants to use historic loan repayment records to categorize loan applications as low-risk or high-risk based on characteristics like the loan amount, the income of the borrower, and the loan period. What kind of machine learning model should the bank use automated machine learning to create?

- ○ Classification

Correct. To predict a category, or class, use a classification model.

- ○ Regression
- ○ Time series forecasting

3. You want to use automated machine learning to train a regression model with the best possible R2 score. How should you configure the automated machine learning experiment?

- ○ Set the Primary metric to R2 score

Correct. The primary metric determines the metric used to evaluate the best performing model.

- ○ Block all algorithms other than GradientBoosting
- ○ Enable featurization

Summary

In this module, you explored machine learning and learned how to use the automated machine learning capability of Azure Machine Learning to train and deploy a predictive model.

Clean-up

The web service you created is hosted in an *Azure Container Instance*. If you don't intend to experiment with it further, you should delete the endpoint to avoid accruing unnecessary Azure usage.

1. In Azure Machine Learning studio, on the **Endpoints** tab, select the **predict-rentals** endpoint. Then select **Delete** (🗑) and confirm that you want to delete the endpoint.

Note

Deleting the endpoint ensures your subscription won't be charged for the container instance in which it is hosted. You will however be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription. If you have finished exploring Azure Machine Learning, you can delete the Azure Machine Learning workspace and associated resources. However, if you plan to complete any other labs in this series, you will need to recreate it.

To delete your workspace:

1. In the **Azure portal**, in the **Resource groups** page, open the resource group you specified when creating your Azure Machine Learning workspace.
2. Click **Delete resource group**, type the resource group name to confirm you want to delete it, and select **Delete**.

Module complete:

Part 2/6_2/4

Create a Regression Model with Azure Machine Learning designer

Introduction

Regression is a form of machine learning that is used to predict a numeric *label* based on an item's *features*. For example, an automobile sales company might use the characteristics of a car (such as engine size, number of seats, mileage, and so on) to predict its likely selling price. In this case, the characteristics of the car are the features, and the selling price is the label.

Regression is an example of a *supervised* machine learning technique in which you train a model using data that includes both the features and known values for the label, so that the model learns to *fit* the feature combinations to the label. Then, after training has been completed, you can use the trained model to predict labels for new items for which the label is unknown.

You can use Microsoft Azure Machine Learning designer to create regression models by using a drag and drop visual interface, without needing to write any code.

In this module, you'll learn how to:

- Use Azure Machine Learning designer to train a regression model.
- Use a regression model for inferencing.
- Deploy a regression model as a service.

To complete this module, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.microsoft.com.

Create an Azure Machine Learning workspace

Azure Machine Learning is a cloud-based platform for building and operating machine learning solutions in Azure. It includes a wide range of features and capabilities that help data scientists prepare data, train models, publish predictive services, and monitor their usage. One of these features is a visual interface called *designer*, that you can use to train, test, and deploy machine learning models without writing any code.

Create an Azure Machine Learning workspace

To use Azure Machine Learning, you create a *workspace* in your Azure subscription. You can then use this workspace to manage data, compute resources, code, models, and other artifacts related to your machine learning workloads.

Note

This module is one of many that make use of an Azure Machine Learning workspace, including the other modules in the **Microsoft Azure AI Fundamentals: Explore visual tools for machine learning** learning path. If you are using your own Azure subscription, you may consider creating the workspace once and reusing it in other modules. Your Azure subscription will be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription, so we recommend you delete the Azure Machine Learning workspace when it is no longer required.

If you don't already have one, follow these steps to create a workspace:

1. Sign into the Azure portal using your Microsoft credentials.
2. Select **Create a resource**, search for *Machine Learning*, and create a new **Azure Machine Learning** resource with an *Azure Machine Learning* plan. Use the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*

- **Workspace name**: *Enter a unique name for your workspace*
- **Region**: *Select the closest geographical region*
- **Storage account**: *Note the default new storage account that will be created for your workspace*
- **Key vault**: *Note the default new key vault that will be created for your workspace*
- **Application insights**: *Note the default new application insights resource that will be created for your workspace*
- **Container registry**: None (*one will be created automatically the first time you deploy a model to a container*)

3. Select **Review + create**. Wait for your workspace to be created (it can take a few minutes). Then go to it in the portal.
4. On the **Overview** page for your workspace, launch Azure Machine Learning Studio (or open a new browser tab and navigate to https://ml.azure.com), and sign into Azure Machine Learning Studio using your Microsoft account.
5. In Azure Machine Learning Studio, select the three lines icon at the top left to view the various pages in the interface. You can use these pages to manage the resources in your workspace.

You can manage your workspace using the Azure portal, but for data scientists and Machine Learning operations engineers, Azure Machine Learning Studio provides a more focused user interface for managing workspace resources.

Create compute resources

To train and deploy models using Azure Machine Learning designer, you need compute targets to run the training process. You will also use these compute targets to test the trained model after its deployment.

Create compute targets

Compute targets are cloud-based resources on which you can run model training and data exploration processes.

In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Compute** page (under **Manage**). You manage the compute targets for your data science activities in the studio. There are four kinds of compute resource that you can create:

- **Compute Instances**: Development workstations that data scientists can use to work with data and models.
- **Compute Clusters**: Scalable clusters of virtual machines for on-demand processing of experiment code.
- **Inference Clusters**: Deployment targets for predictive services that use your trained models.
- **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

Note

Compute clusters are based on standard Azure virtual machine images. For this module, the *Standard_DS11_v2* image is recommended to achieve the optimal balance of cost and performance. If your subscription has a quota that does not include this image, choose an alternative image; but bear in mind that a larger image may incur higher cost and a smaller image may not be sufficient to complete the tasks. Alternatively, ask your Azure administrator to extend your quota.

1. On the **Compute Clusters** tab, add a new compute cluster with the following settings:
   - **Location**: *Select the same as your workspace. If that location is not listed, choose the one closest to you*
   - **Virtual Machine tier**: Dedicated
   - **Virtual Machine type**: CPU
   - **Virtual Machine size**:
     - Choose **Select from all options**
     - Search for and select **Standard_DS11_v2**
   - Select **Next**

- **Compute name**: *enter a unique name*
- **Minimum number of nodes**: 0
- **Maximum number of nodes**: 2
- **Idle seconds before scale down**: 120
- **Enable SSH access**: Unselected
- Select **Create**

Tip

After you finish the entire module, be sure to follow the **Clean Up** instructions at the end of the module to stop your compute resources. Stop your compute resources to ensure your subscription won't be charged.

The compute target takes some time to be created. You can move onto the next unit while you wait.

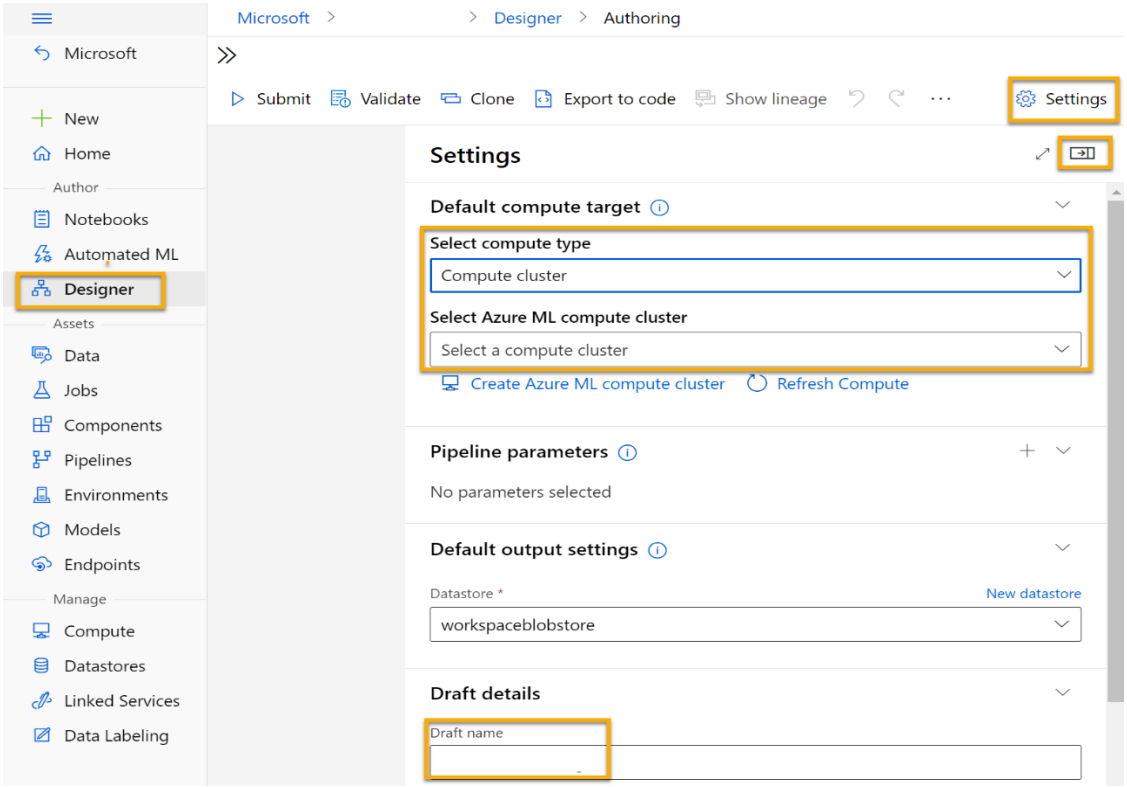
Explore data

To train a regression model, you need a dataset that includes historical *features*, characteristics of the entity for which you want to make a prediction. You also need known *label* values, the numeric value that you want to train a model to predict.

Create a pipeline

To use the Azure Machine Learning designer, you create a *pipeline* that you use to train a machine learning model. This pipeline starts with the dataset from which you want to train the model.

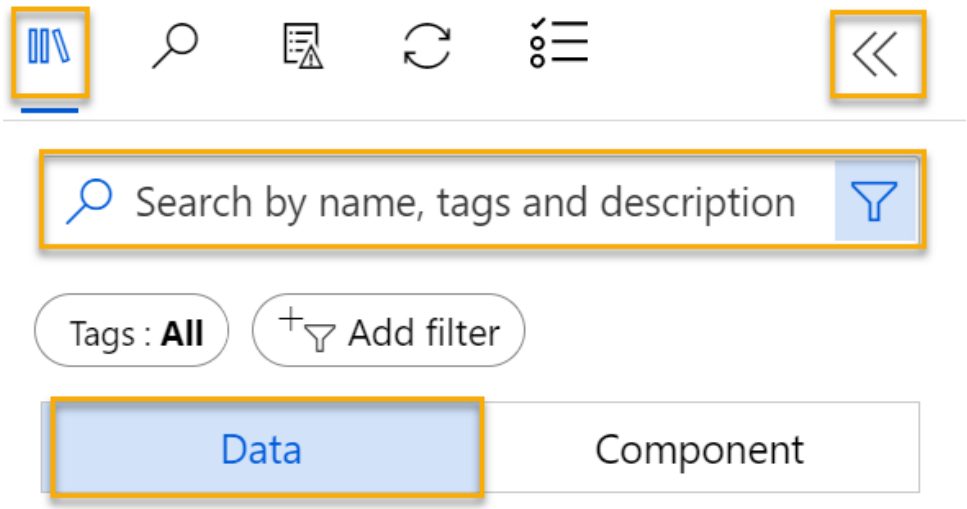
1. In Azure Machine Learning studio, expand the left pane by selecting the three lines icon at the top left of the screen. View the **Designer** page (under **Author**), and select **+** to create a new pipeline.
2. At the top right-hand side of the screen, select **Settings**. If the **Settings** pane is not visible, select the wheel icon next to the pipeline name at the top.
3. In **Settings**, you must specify a compute target on which to run the pipeline. Under **Select compute type**, select **Compute cluster**. Then under **Select Azure ML *compute-type***, select the compute cluster you created previously.
4. In **Settings**, under **Draft Details**, change the draft name (**Pipeline-Created-on-*date***) to **Auto Price Training**.
5. Select the *close icon* on the top right of the **Settings** pane to close the pane.



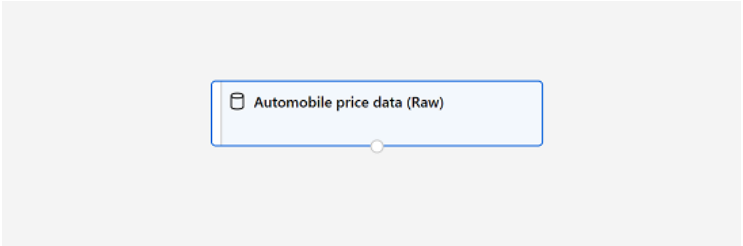
Add and explore a dataset

In this module, you train a regression model that predicts the price of an automobile based on its characteristics. Azure Machine Learning includes a sample dataset that you can use for this model.

1. Next to the pipeline name on the left, select the arrows icon to expand the panel if it is not already expanded. The panel should open by default to the **Asset Library** pane, indicated by the books icon at the top of the panel. Note that there is a search bar to locate assets. Notice two buttons, **Data** and **Components**.



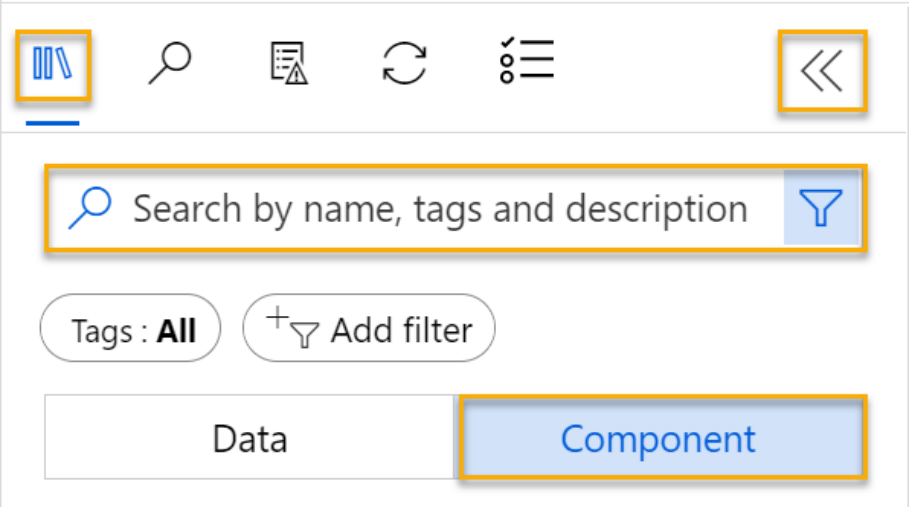
2. Click on **Data**. Search for and place the **Automobile price data (Raw)** dataset onto the canvas.
3. Right-click (Ctrl+click on a Mac) the **Automobile price data (Raw)** dataset on the canvas, and click on **Preview data**.
4. Review the *Dataset output* schema of the data, noting that you can see the distributions of the various columns as histograms.
5. Scroll to the right of the dataset until you see the **Price** column, which is the label that your model predicts.
6. Scroll back to the left and select the **normalized-losses** column header. Then review the statistics for this column. Note there are quite a few missing values in this column. Missing values limit the column's usefulness for predicting the **price** label so you might want to exclude it from training.
7. Close the **Automobile price data (Raw) result visualization** window so that you can see the dataset on the canvas like this:



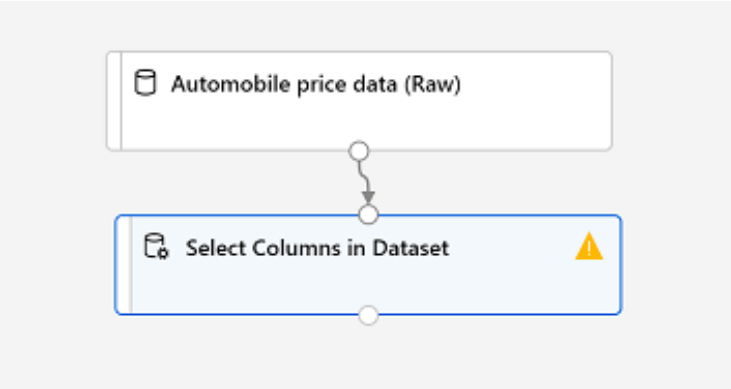
Add data transformations

You typically apply data transformations to prepare the data for modeling. In the case of the automobile price data, you add transformations to address the issues you identified when you explored the data.

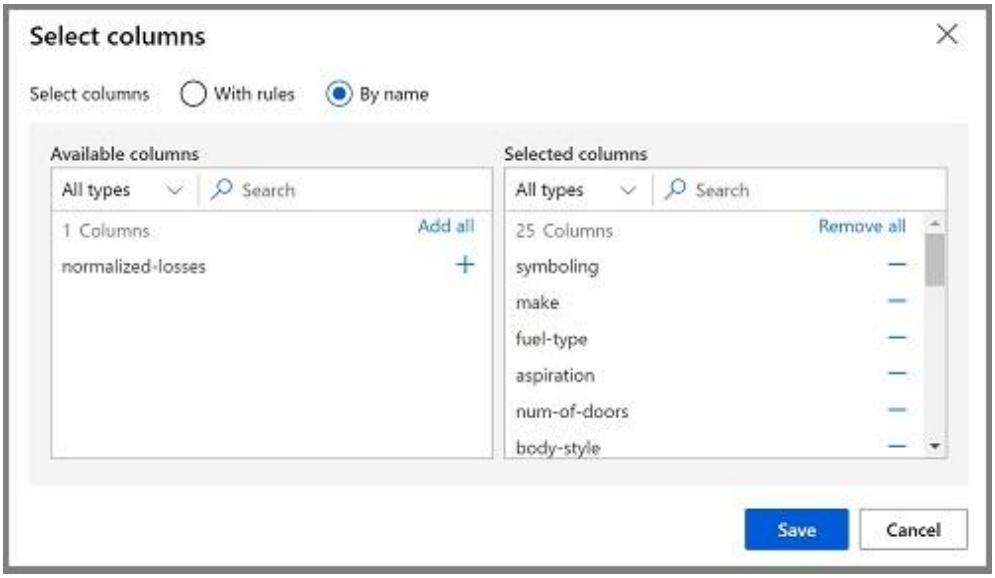
1. In the **Asset Library** pane on the left, click on **Components**, which contain a wide range of modules you can use for data transformation and model training. You can also use the search bar to quickly locate modules.



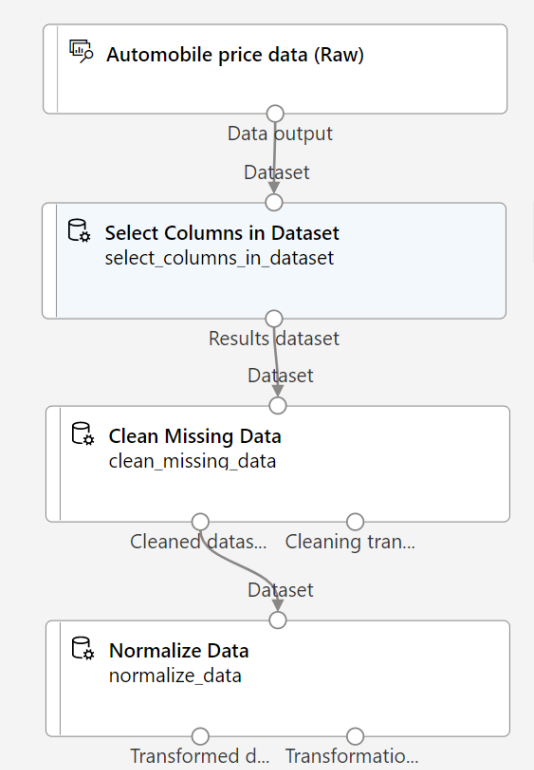
2. Search for a **Select Columns in Dataset** module and place it to the canvas, below the **Automobile price data (Raw)** module. Then connect the output at the bottom of the **Automobile price data (Raw)** module to the input at the top of the **Select Columns in Dataset** module, like this:



3. Double click on the **Select Columns in Dataset** module to access a settings pane on the right. Select **Edit column**. Then in the **Select columns** window, select **By name** and **Add all** to add all the columns. Then remove **normalized-losses**, so your final column selection looks like this:



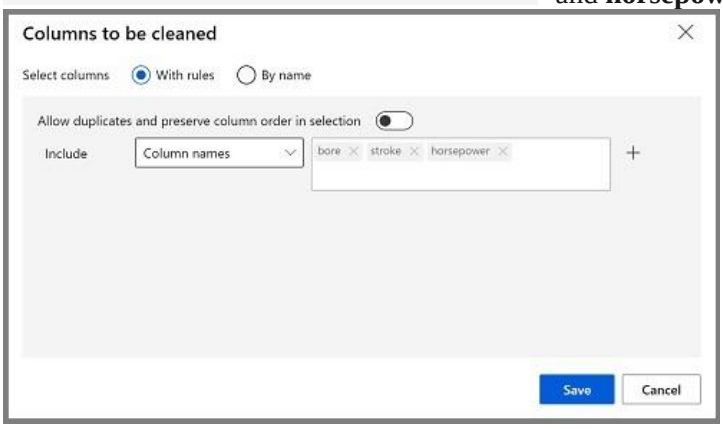
In the rest of this exercise, you go through steps to create a pipeline that looks like this:



Follow the remaining steps, use the image for reference as you add and configure the required modules.

4. In the **Asset Library**, search for a **Clean Missing Data** module and place it under the **Select Columns in Dataset** module on the canvas. Then connect the output from the **Select Columns in Dataset** module to the input of the **Clean Missing Data** module.

5. Double click the **Clean Missing Data** module, and in the pane on the right, click **Edit column**. Then in the **Select columns** window, select **With rules**, in the **Include** list select **Column names**, in the box of column names enter **bore**, **stroke**, and **horsepower** like this:



6. With the **Clean Missing Data** module still selected, in the pane on the right, set the following configuration settings:

- **Minimum missing value ratio**: 0.0
- **Maximum missing value ratio**: 1.0
- **Cleaning mode**: Remove entire row

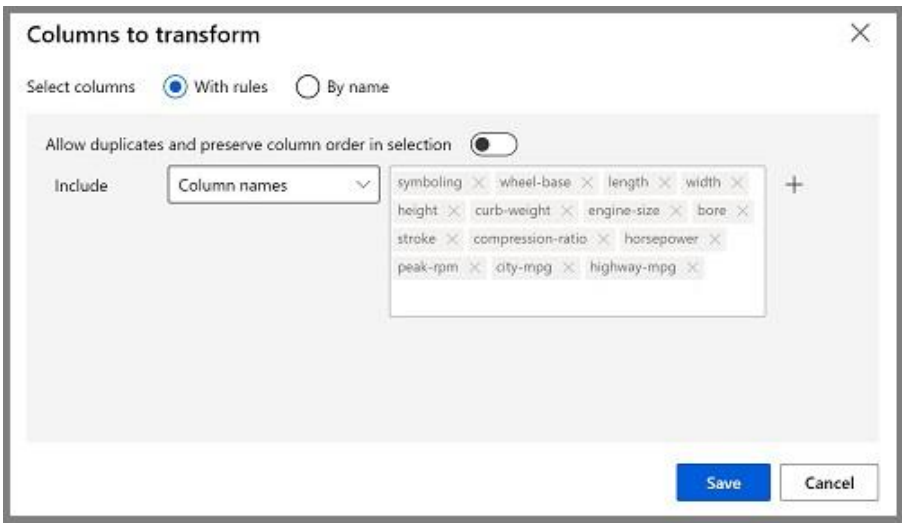
Tip

If you view the statistics for the **bore**, **stroke**, and **horsepower** columns, you'll see a number of missing values. These columns have fewer missing values than **normalized-losses**, so they might still be useful in predicting **price** once you exclude the rows where the values are missing from training.

7. In the **Asset library**, search for a **Normalize Data** module and place it on the canvas, below the **Clean Missing Data** module. Then connect the left-most output from the **Clean Missing Data** module to the input of the **Normalize Data** module.

8. Double click on the **Normalize Data** module to view its parameters pane. Note that it requires you to specify the transformation method and the columns to be transformed. Then, set the transformation to **MinMax**. Apply a rule by selecting **Edit column** to include the following **Column names**:

- **symboling**
- **wheel-base**
- **length**
- **width**
- **height**
- **curb-weight**
- **engine-size**
- **bore**
- **stroke**
- **compression-ratio**
- **horsepower**
- **peak-rpm**
- **city-mpg**
- **highway-mpg**



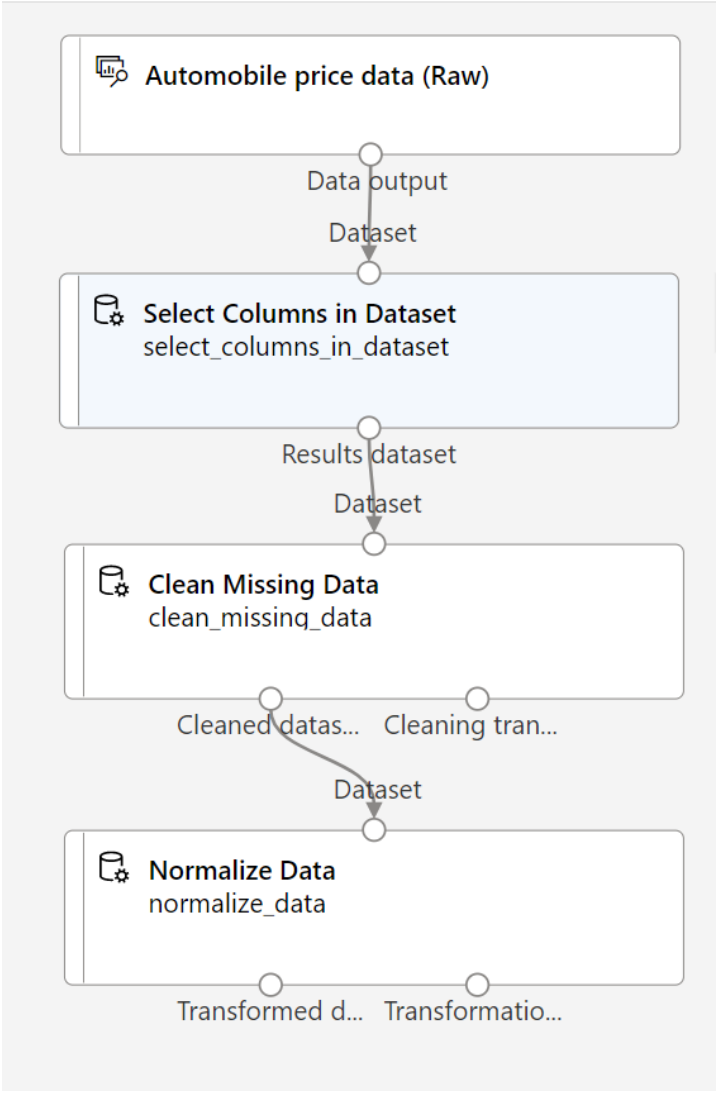
Tip

If you compare the values in the **stroke**, **peak-rpm**, and **city-mpg** columns, they are all measured in different scales, and it is possible that the larger values for **peak-rpm** might bias the training algorithm and create an over-dependency on this column compared to columns with lower values, such as **stroke**. Typically, data scientists mitigate this possible bias by *normalizing* the numeric columns so they're on the similar scales.

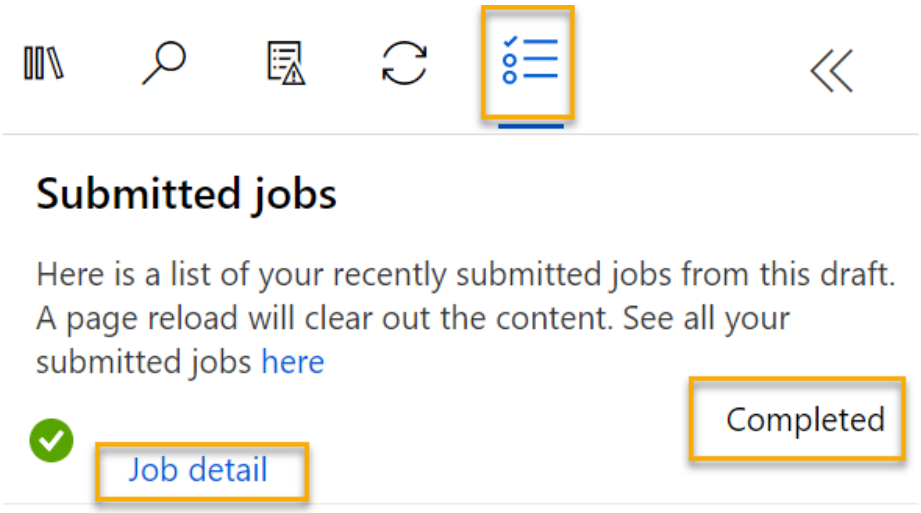
Run the pipeline

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

1. Ensure that your pipeline looks similar to this image:

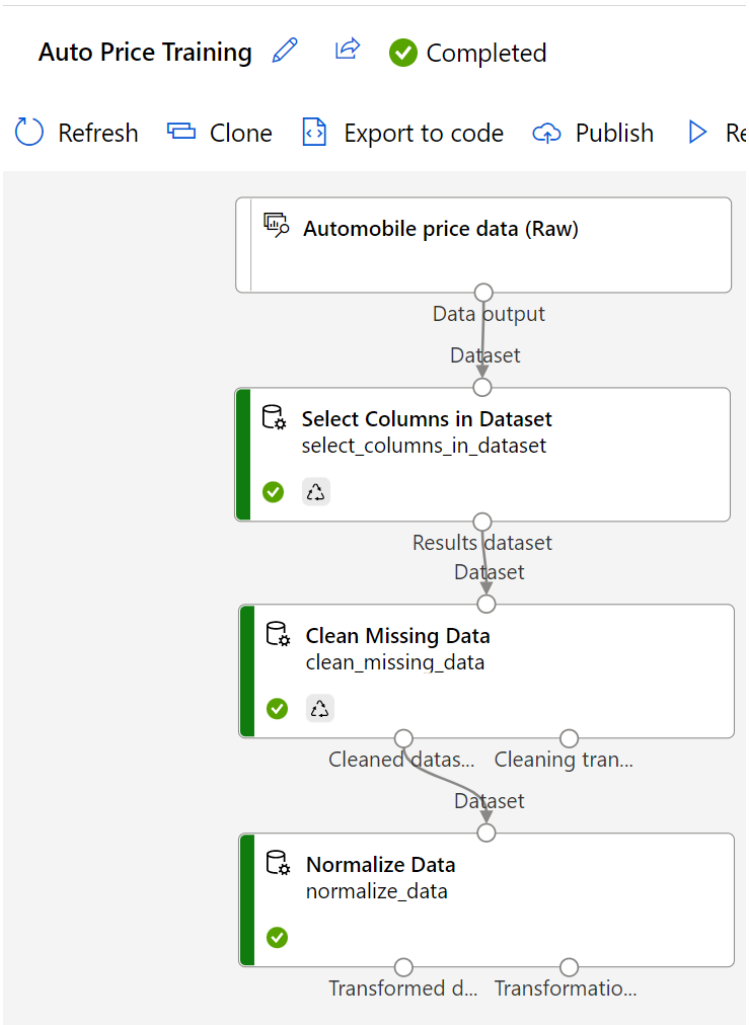


2. Select **Submit**, and select **new experiment** named **mslearn-auto-training** on your compute cluster.
3. Wait for the run to finish, which might take 5 minutes or more.



Notice that the left hand panel is now on the **Submitted Jobs** pane. You will know when the run is complete because the status of the job will change to **Completed**.

4. When the run has completed, click on **Job detail**. You will be taken to another window. Note that the modules have completed check marks like this:



The dataset is now prepared for model training. Close the Job detail window to return to the pipeline.

Create and run a training pipeline

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

Add training modules

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

In this exercise, you're going to work through steps to extend the **Auto Price Training** pipeline.

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

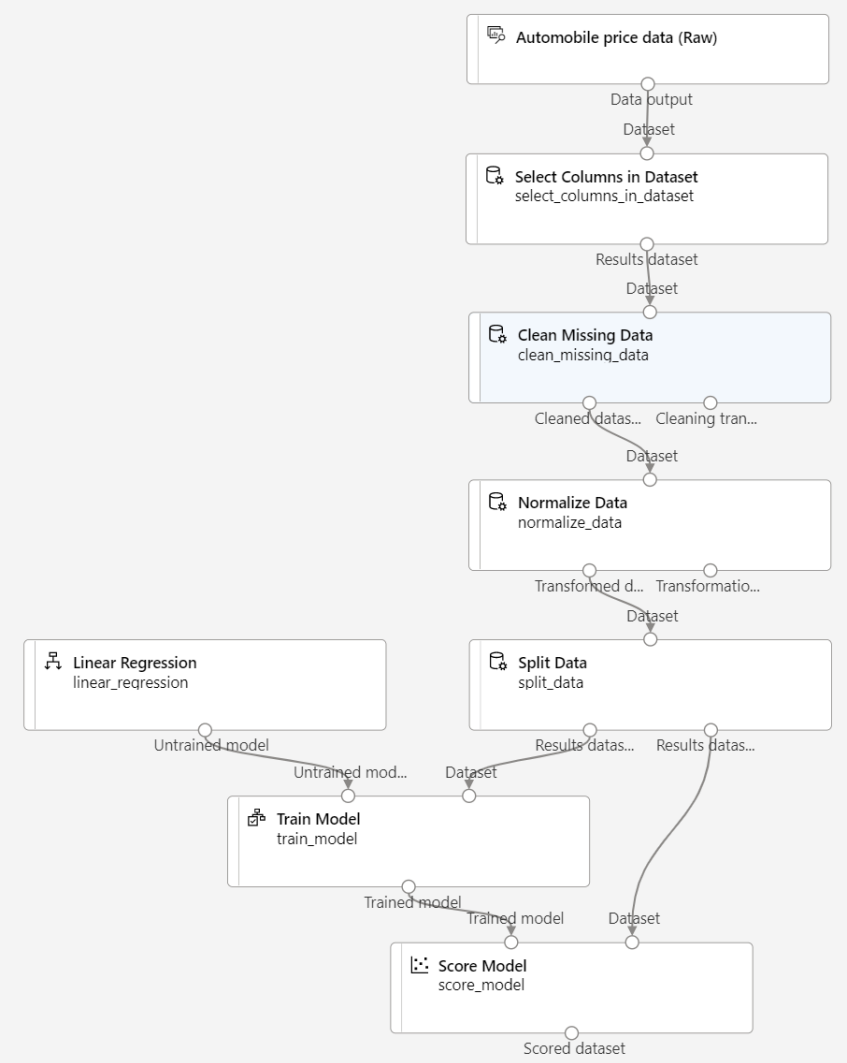
1. Return to the **Auto Price Training** pipeline you created in the previous unit if it's not already open.
2. In the **Asset Library** pane on the left, search for and place a **Split Data** module onto the canvas under the **Normalize Data** module. Then connect the *Transformed Dataset* (left) output of the **Normalize Data** module to the input of the **Split Data** module.

   Tip

   Use the search bar to quickly locate modules.
3. Select the **Split Data** module, and configure its settings as follows:
   - **Splitting mode**: Split Rows
   - **Fraction of rows in the first output dataset**: 0.7
   - **Randomized split**: True
   - **Random seed**: 123
   - **Stratified split**: False
4. In the **Asset Library**, search for and place a **Train Model** module to the canvas, under the **Split Data** module. Then connect the *Result dataset1* (left) output of the **Split Data** module to the *Dataset* (right) input of the **Train Model** module.
5. The model we're training will predict the **price** value, so select the **Train Model** module and modify its settings to set the **Label column** to **price** (matching the case and spelling exactly!)
6. The **price** label the model will predict is a numeric value, so we need to train the model using a *regression* algorithm. In the **Asset Library**, search for and place a **Linear Regression** module to the canvas, to the left of the **Split Data** module and above the **Train Model** module. Then connect its output to the **Untrained model** (left) input of the **Train Model** module.

   Note

   There are multiple algorithms you can use to train a regression model. For help choosing one, take a look at the **Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer**.
7. To test the trained model, we need to use it to *score* the validation dataset we held back when we split the original data - in other words, predict labels for the features in the validation dataset. In the **Asset Library**, search for and place a **Score Model** module to the canvas, below the **Train Model** module. Then connect the output of the **Train Model** module to the **Trained model** (left) input of the **Score Model** module; and drag the **Results dataset2** (right) output of the **Split Data** module to the **Dataset** (right) input of the **Score Model** module.
8. Ensure your pipeline looks like this image:



Run the training pipeline

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

1. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-auto-training**.
2. Wait for the experiment run to complete. This may take 5 minutes or more. When the experiment run has completed, click on **Job details**. You will be taken to a new window.
3. In the new window, right click on the **Score Model** module and select **Preview data** and then **Scored dataset** to view the results.
4. Scroll to the right, and note that next to the **price** column (which contains the known true values of the label) there is a new column named **Scored labels**, which contains the predicted label values.
5. Close the **Score Model result visualization** window.

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

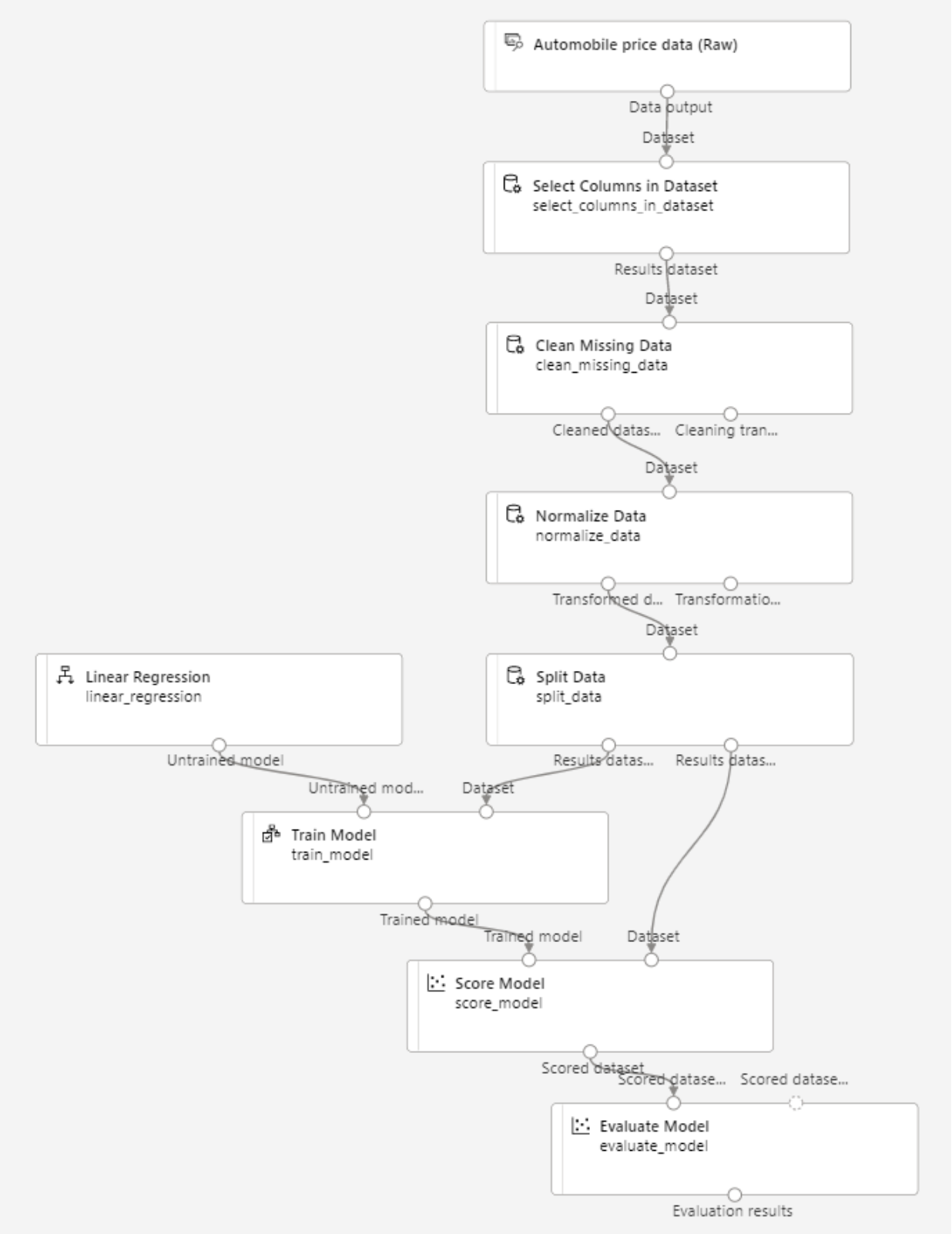
Evaluate a regression model

To evaluate a regression model, you could simply compare the predicted labels to the actual labels in the validation dataset to held back during training, but this is an imprecise process and doesn't provide a simple metric that you can use to compare the performance of multiple models.

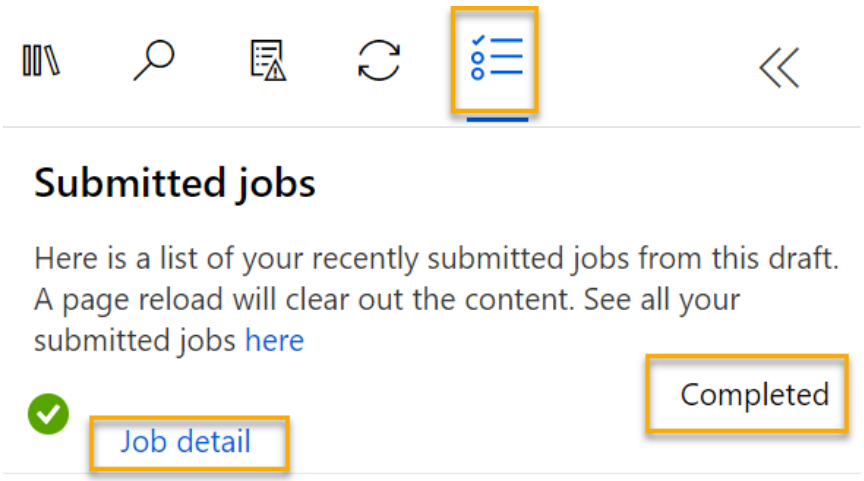
Add an Evaluate Model module

1. Open the **Auto Price Training** pipeline you created in the previous unit if it's not already open.

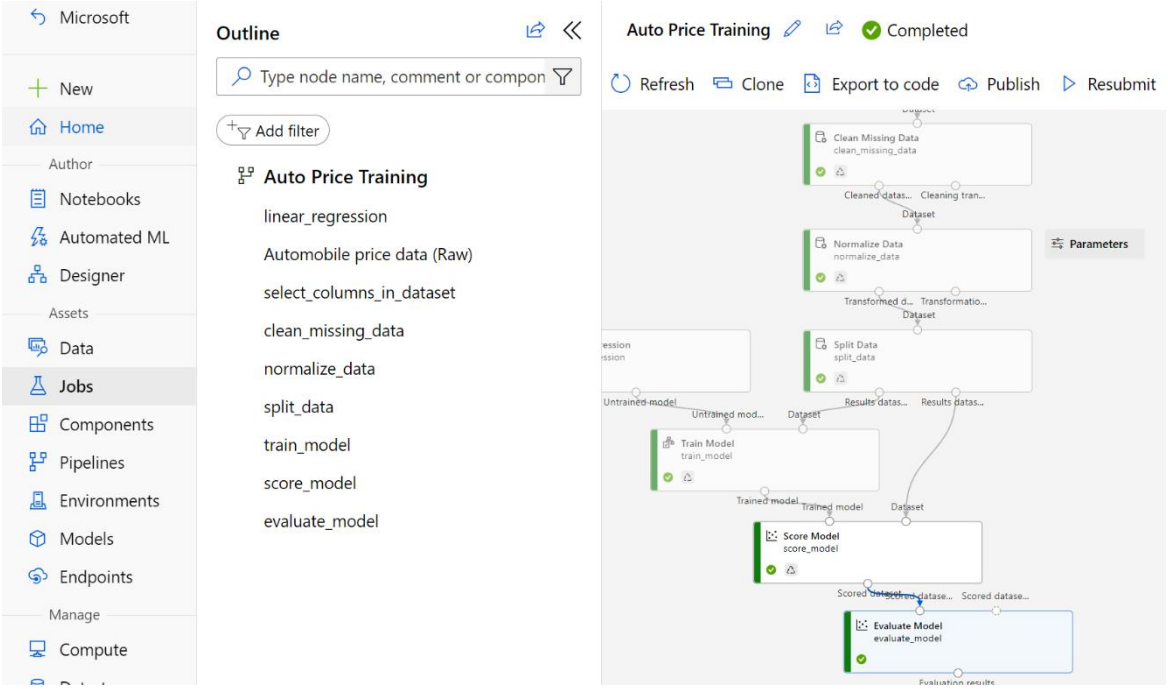
2. In the **Asset Library**, search for and place an **Evaluate Model** module to the canvas, under the **Score Model** module, and connect the output of the **Score Model** module to the **Scored dataset** (left) input of the **Evaluate Model** module.
3. Ensure your pipeline looks like this:



4. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-auto-training**.
5. Wait for the experiment run to complete.



6. When the experiment run has completed, select **Job details**. This will take you to another window. Find and right click on the **Evaluate Model** module. Select **Preview data** and then **Evaluation results**.



7. In the *Evaluation_results* pane, review the regression performance metrics. These include the following metrics:

   - **Mean Absolute Error (MAE)**: The average difference between predicted values and true values. This value is based on the same units as the label, in this case dollars. The lower this value is, the better the model is predicting.
   - **Root Mean Squared Error (RMSE)**: The square root of the mean squared difference between predicted and true values. The result is a metric based on the same unit as the label (dollars). When compared to the MAE (above), a larger difference indicates greater variance in the individual errors (for example, with some errors being very small, while others are large).
   - **Relative Squared Error (RSE)**: A relative metric between 0 and 1 based on the square of the differences between predicted and true values. The closer to 0 this metric is, the better the model is performing. Because this metric is relative, it can be used to compare models where the labels are in different units.
   - **Relative Absolute Error (RAE)**: A relative metric between 0 and 1 based on the absolute differences between predicted and true values. The closer to 0 this metric is, the better the model is performing. Like RSE, this metric can be used to compare models where the labels are in different units.

- **Coefficient of Determination (R^2)**: This metric is more commonly referred to as *R-Squared*, and summarizes how much of the variance between predicted and true values is explained by the model. The closer to 1 this value is, the better the model is performing.

8. Close the *Evaluation_results* pane.

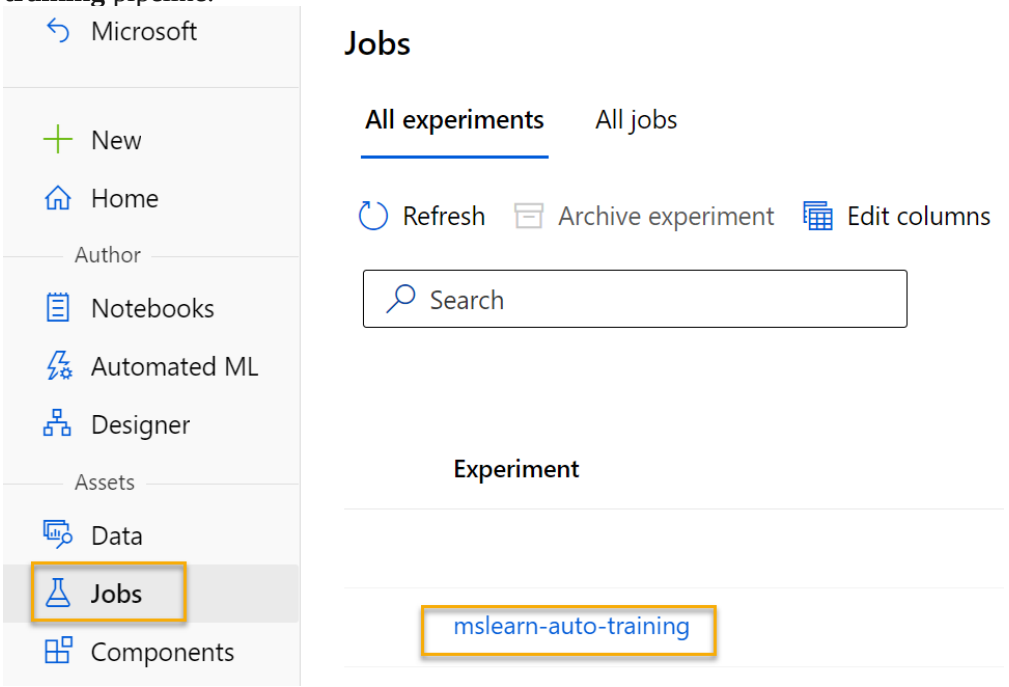
When you've identified a model with evaluation metrics that meet your needs, you can prepare to use that model with new data.

Create an inference pipeline

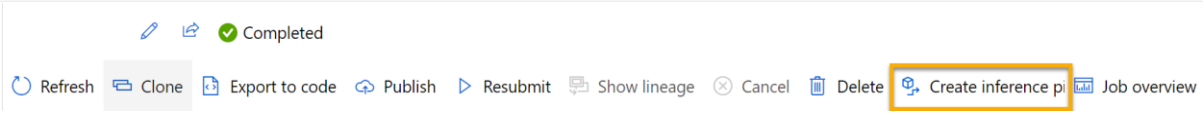
After creating and running a pipeline to train the model, you need a second pipeline known as an inference pipeline. The inference pipeline performs the same data transformations as the first pipeline for *new* data. Then it uses the trained model to *infer*, or predict, label values based on its features. This model will form the basis for a predictive service that you can publish for applications to use.

Create and run an inference pipeline

1. In Azure Machine Learning studio, expand the left-hand pane by selecting the three lines at the top left of the screen. Click on **Jobs** (under **Assets**) to view all of the jobs you have run. Select the experiment **mslearn-auto-training**, then select the **mslearn-auto-training** pipeline.



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

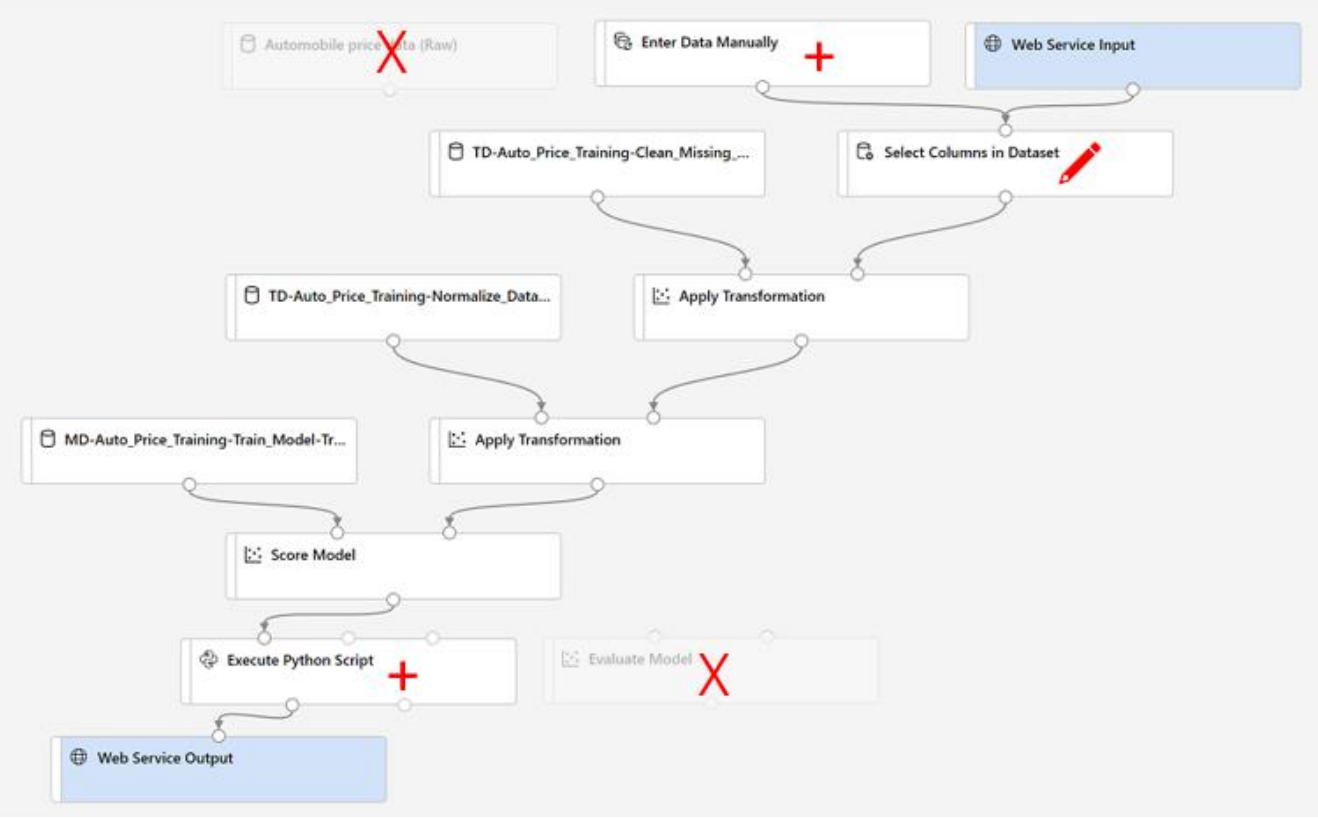


3. In the **Create inference pipeline** drop-down list, click **Real-time inference pipeline**. After a few seconds, a new version of your pipeline named **Auto Price Training-real time inference** will be opened.

   *If the pipeline doesn't include **Web Service Input** and **Web Service Output** modules, go back to the **Designer** page and then reopen the **Auto Price Training-real time inference** pipeline.*

4. Rename the new pipeline to **Predict Auto Price**, and then review the new pipeline. It contains a web service input for new data to be submitted, and a web service output to return results. Some of the transformations and training steps are a part of this pipeline. The trained model will be used to score the new data.

   You're going to make the following changes to the inference pipeline in the next steps #5-9:



Use the image for reference as you modify the pipeline in the next steps.

5. The inference pipeline assumes that new data will match the schema of the original training data, so the **Automobile price data (Raw)** dataset from the training pipeline is included. However, this input data includes the **price** label that the model predicts, which is unintuitive to include in new car data for which a price prediction hasn't yet been made. Delete this module and replace it with an **Enter Data Manually** module from the **Data Input and Output** section, containing the following CSV data, which includes feature values without labels for three cars (copy and paste the entire block of text):

   CSVCopy

   ```
   symboling,normalized-losses,make,fuel-type,aspiration,num-of-doors,body-style,drive-wheels,engine-location,wheel-base,length,width,height,curb-weight,engine-type,num-of-cylinders,engine-size,fuel-system,bore,stroke,compression-ratio,horsepower,peak-rpm,city-mpg,highway-mpg

   3,NaN,alfa-romero,gas,std,two,convertible,rwd,front,88.6,168.8,64.1,48.8,2548,dohc,four,130,mpfi,3.47,2.68,9,111,5000,21,27

   3,NaN,alfa-romero,gas,std,two,convertible,rwd,front,88.6,168.8,64.1,48.8,2548,dohc,four,130,mpfi,3.47,2.68,9,111,5000,21,27

   1,NaN,alfa-romero,gas,std,two,hatchback,rwd,front,94.5,171.2,65.5,52.4,2823,ohcv,six,152,mpfi,2.68,3.47,9,154,5000,19,26
   ```

6. Connect the new **Enter Data Manually** module to the same **dataset** input of the **Select Columns in Dataset** module as the **Web Service Input**.

7. Now that you've changed the schema of the incoming data to exclude the **price** field, you need to remove any explicit uses of this field in the remaining modules. Select the **Select Columns in Dataset** module and then in the settings pane, edit the columns to remove the **price** field.

8. The inference pipeline includes the **Evaluate Model** module, which isn't useful when predicting from new data, so delete this module.

9. The output from the **Score Model** module includes all of the input features and the predicted label. To modify the output to include only the prediction:

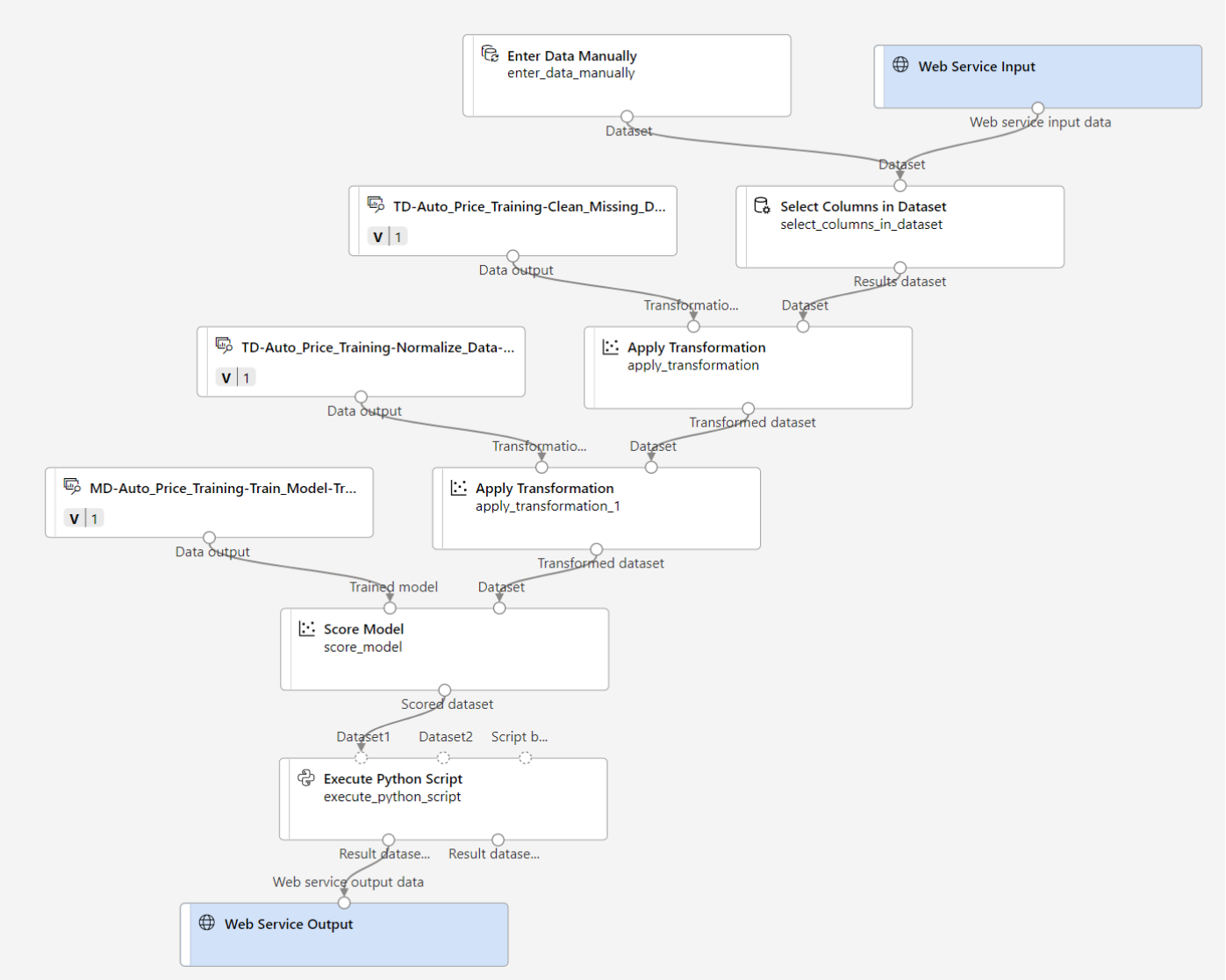
   - Delete the connection between the **Score Model** module and the **Web Service Output**.

- Add an **Execute Python Script** module from the **Python Language** section, replacing all of the default python script with the following code (which selects only the **Scored Labels** column and renames it to **predicted_price**):

```
PythonCopy

import pandas as pd


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


    scored_results = dataframe1[['Scored Labels']]
    scored_results.rename(columns={'Scored Labels':'predicted_price'},
                          inplace=True)
    return scored_results
```

- Connect the output from the **Score Model** module to the **Dataset1** (left-most) input of the **Execute Python Script**, and connect the output of the **Execute Python Script** module to the **Web Service Output**.

10. Verify that your pipeline looks similar to the following image:



11. Submit the pipeline as a new experiment named **mslearn-auto-inference** on your compute cluster. The experiment may take a while to run.

12. When the pipeline has completed, select **Job details**. In the new window, right click on the **Execute Python Script** module. Select **Preview data** and then **Result dataset** to see the predicted prices for the three cars in the input data.

13. Close the visualization window.

Your inference pipeline predicts prices for cars based on their features. Now you're ready to publish the pipeline so that client applications can use it.

Deploy a predictive service

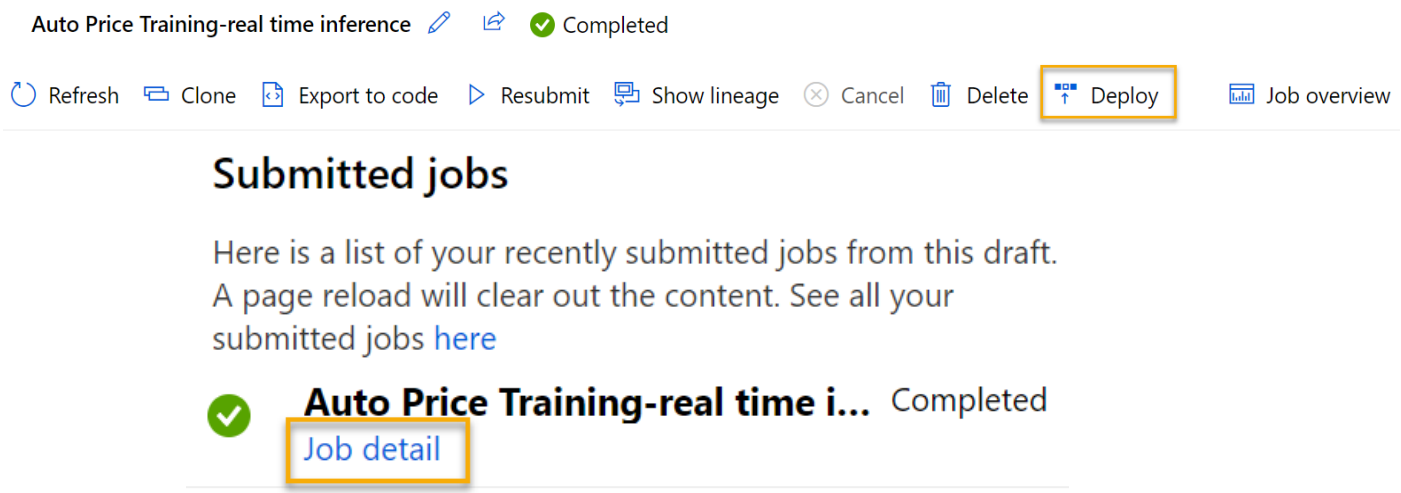
After you've created and tested an inference pipeline for real-time inferencing, you can publish it as a service for client applications to use.

Note

In this exercise, you'll deploy the web service to an Azure Container Instance (ACI). This type of compute is created dynamically, and is useful for development and testing. For production, you should create an *inference cluster*, which provides an Azure Kubernetes Service (AKS) cluster that provides better scalability and security.

Deploy a service

1. View the **Predict Auto Price** inference pipeline you created in the previous unit.
2. Select **Job detail** on the left hand pane. This will open up another window.



3. In the new window, select **Deploy**.

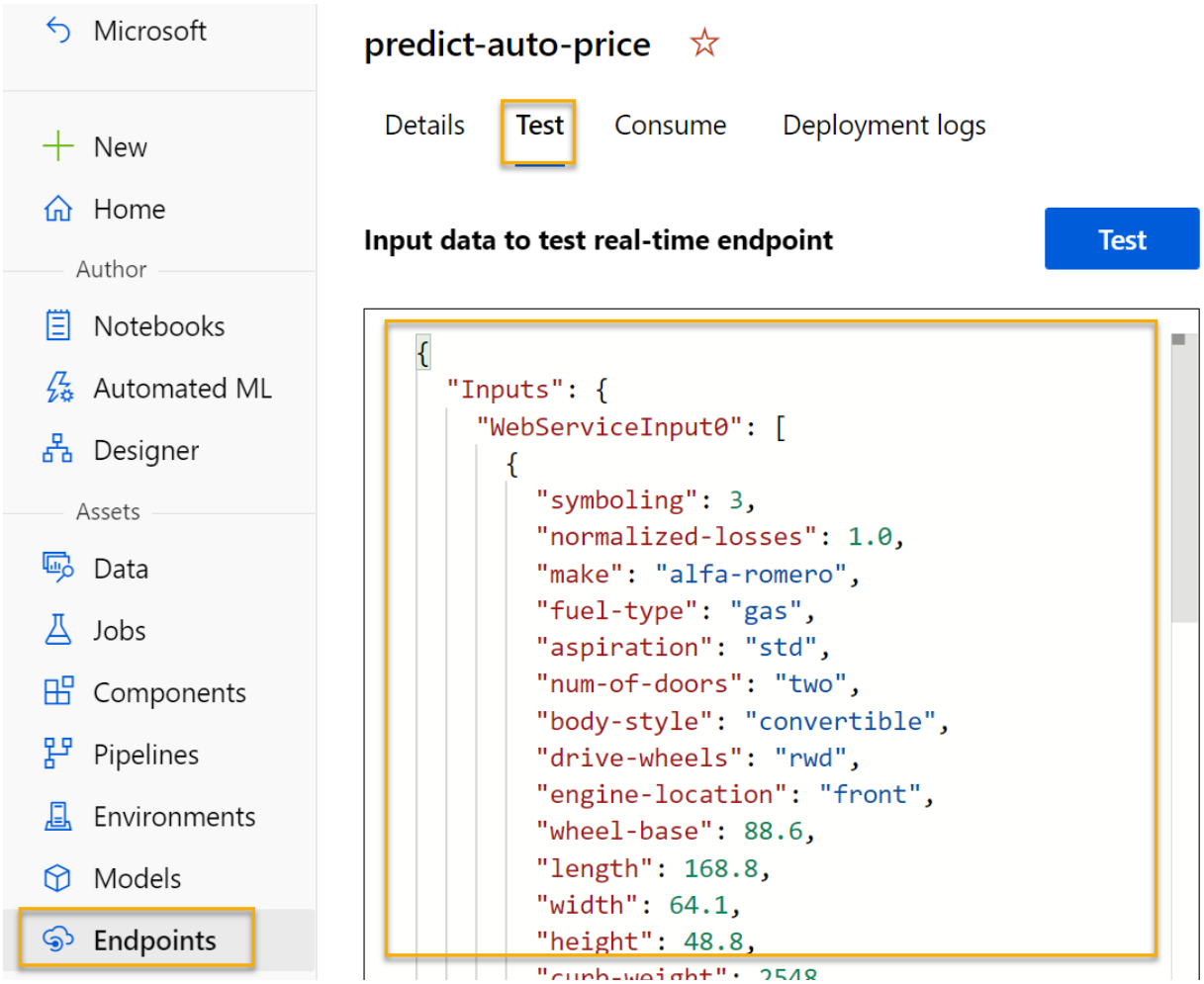
4. In the configuration screen, select **Deploy a new real-time endpoint**, using the following settings:
    - **Name**: predict-auto-price
    - **Description**: Auto price regression
    - **Compute type**: Azure Container Instance
5. Wait for the web service to be deployed - this can take several minutes. The deployment status is shown at the top left of the designer interface.

Test the service

1. On the **Endpoints** page, open the **predict-auto-price** real-time endpoint.

2. When the **predict-auto-price** endpoint opens, select the **Test** tab. We will use it to test our model with new data. Delete the current data under **Input data to test real-time endpoint**. Copy and paste the below data into the data section:

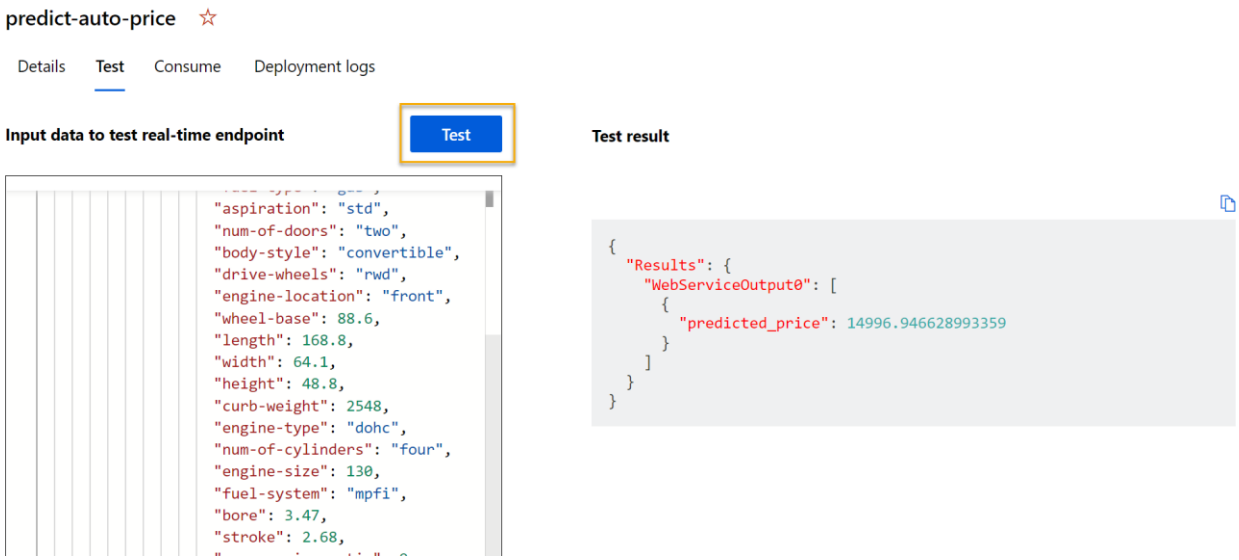
    JSONCopy




```
{
"Inputs": {
"WebServiceInput0":
[
{
"symboling": 3,
"normalized-losses": 1.0,
"make": "alfa-romero",
"fuel-type": "gas",
"aspiration": "std",
"num-of-doors": "two",
"body-style": "convertible",
"drive-wheels": "rwd",
"engine-location": "front",
"wheel-base": 88.6,
"length": 168.8,
"width": 64.1,
"height": 48.8,
"curb-weight": 2548,
"engine-type": "dohc",
"num-of-cylinders": "four",
"engine-size": 130,
"fuel-system": "mpfi",
```

```
            "bore": 3.47,
            "stroke": 2.68,
            "compression-ratio": 9,
            "horsepower": 111,
            "peak-rpm": 5000,
            "city-mpg": 21,
            "highway-mpg": 27
          }
        ]
      },
    "GlobalParameters": {}
    }
```


3. Select **Test**. On the right hand of the screen, you should see the output **'predicted_price'**. The output is the predicted price for a vehicle with the particular input features specified in the data.



Knowledge check

1. You are creating a training pipeline for a regression model. You use a dataset that has multiple numeric columns in which the values are on different scales. You want to transform the numeric columns so that the values are all on a similar scale. You also want the transformation to scale relative to the minimum and maximum values in each column. Which module should you add to the pipeline?

- Select Columns in a Dataset
- Normalize Data

Correct. When you transform numeric data to be on a similar scale, use a Normalize Data module.

- Clean Missing Data

2. Why do you split data into training and validation sets?

- Data is split into two sets in order to create two models, one model with the training set and a different model with the validation set.

Splitting data into two sets enables you to compare the labels that the model predicts with the actual known labels in the original dataset.

Correct. You want to test the model created with training data on validation data to see how well the model performs with data it was not trained on.

Only split data when you use the Azure Machine Learning Designer, not in other machine learning scenarios.

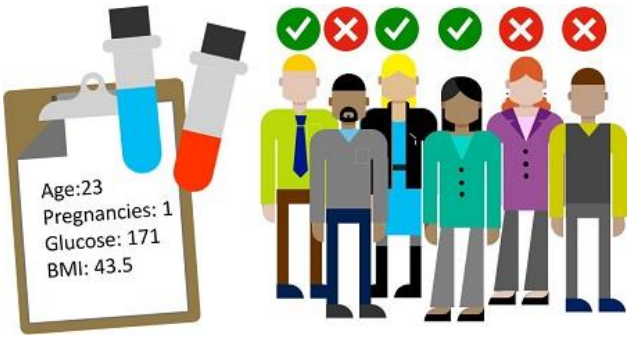
Module complete:

Part 2/6_3/4

Create a classification model with Azure Machine Learning designer

Introduction

Classification is a form of machine learning that is used to predict which category, or *class*, an item belongs to. For example, a health clinic might use the characteristics of a patient (such as age, weight, blood pressure, and so on) to predict whether the patient is at risk of diabetes. In this case, the characteristics of the patient are the features, and the label is a classification of either **0** or **1**, representing non-diabetic or diabetic.



Classification is an example of a *supervised* machine learning technique in which you train a model using data that includes both the features and known values for the label, so that the model learns to *fit* the feature combinations to the label. Then, after training has been completed, you can use the trained model to predict labels for new items for which the label is unknown.

You can use Microsoft Azure Machine Learning designer to create classification models by using a drag and drop visual interface, without needing to write any code.

In this module, you'll learn how to:

- Use Azure Machine Learning designer to train a classification model.
- Use a classification model for inferencing.
- Deploy and test a classification model.

To complete this module, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.microsoft.com.

Create an Azure Machine Learning workspace

Azure Machine Learning is a cloud-based platform for building and operating machine learning solutions in Azure. It includes a wide range of features and capabilities that help data scientists prepare data, train models, publish predictive services, and monitor their usage. One of these features is a visual interface called *designer*, that you can use to train, test, and deploy machine learning models without writing any code.

Create an Azure Machine Learning workspace

To use Azure Machine Learning, you create a *workspace* in your Azure subscription. You can then use this workspace to manage data, compute resources, code, models, and other artifacts related to your machine learning workloads.

Note

This module is one of many that make use of an Azure Machine Learning workspace, including the other modules in the **Microsoft Azure AI Fundamentals: Explore visual tools for machine learning** learning path. If you are using your own Azure subscription, you may consider creating the workspace once and reusing it in other modules. Your Azure subscription will be charged a small amount for data storage as long as the Azure Machine Learning

workspace exists in your subscription, so we recommend you delete the Azure Machine Learning workspace when it is no longer required.

If you don't already have one, follow these steps to create a workspace:

1. Sign into the Azure portal using your Microsoft credentials.
2. Select **+Create a resource**, search for *Machine Learning*, and create a new **Azure Machine Learning** resource with an *Azure Machine Learning* plan. Use the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Workspace name**: *Enter a unique name for your workspace*
    - **Region**: *Select the closest geographical region*
    - **Storage account**: *Note the default new storage account that will be created for your workspace*
    - **Key vault**: *Note the default new key vault that will be created for your workspace*
    - **Application insights**: *Note the default new application insights resource that will be created for your workspace*
    - **Container registry**: None (*one will be created automatically the first time you deploy a model to a container*)
3. Select **Review + create**. Wait for your workspace to be created (it can take a few minutes). Then go to it in the portal.
4. On the **Overview** page for your workspace, launch Azure Machine Learning studio (or open a new browser tab and navigate to https://ml.azure.com), and sign into Azure Machine Learning studio using your Microsoft account.
5. In Azure Machine Learning studio, select the three lines at the top left to view the various pages in the interface. You can use these pages to manage the resources in your workspace.

You can manage your workspace using the Azure portal, but for data scientists and Machine Learning operations engineers, Azure Machine Learning studio provides a more focused user interface for managing workspace resources.

Create compute resources

To train and deploy models using Azure Machine Learning designer, you need compute targets to run the training process. You will also use these compute targets to test the trained model after its deployment.

Create compute targets

Compute targets are cloud-based resources on which you can run model training and data exploration processes.

In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Compute** page (under **Manage**). You manage the compute targets for your data science activities in the studio. There are four kinds of compute resource that you can create:

- **Compute Instances**: Development workstations that data scientists can use to work with data and models.
- **Compute Clusters**: Scalable clusters of virtual machines for on-demand processing of experiment code.
- **Inference Clusters**: Deployment targets for predictive services that use your trained models.
- **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

Note

Compute instances and clusters are based on standard Azure virtual machine images. For this module, the *Standard_DS11_v2* image is recommended to achieve the optimal balance of cost and performance. If your subscription has a quota that does not include this image, choose an alternative image; but bear in mind that a larger image may incur higher cost and a smaller image may not be sufficient to complete the tasks. Alternatively, ask your Azure administrator to extend your quota.

1. Navigate to the **Compute Clusters** tab and add a new compute cluster with the following settings:

- **Location**: *Select the same as your workspace. If that location is not listed, choose the one closest to you*
- **Virtual Machine tier**: Dedicated
- **Virtual Machine type**: CPU
- **Virtual Machine size**:
  - Choose **Select from all options**
  - Search for and select **Standard_DS11_v2**
- Select **Next**
- **Compute name**: *enter a unique name*
- **Minimum number of nodes**: 0
- **Maximum number of nodes**: 2
- **Idle seconds before scale down**: 120
- **Enable SSH access**: Unselected
- Select **Create**

Tip

After you finish the entire module, be sure to follow the **Clean Up** instructions at the end of the module to stop your compute resources. Stop your compute resources to ensure your subscription won't be charged.

The compute targets take some time to be created. You can move onto the next unit while you wait.

Explore data

To train a classification model, you need a dataset that includes historical *features* (characteristics of the entity for which you want to make a prediction) and known *label* values (the class indicator that you want to train a model to predict).

Create a dataset

In Azure Machine Learning, data for model training and other operations is usually encapsulated in an object called a *dataset*.

1. In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Data** page (under **Assets**). The Data page contains specific data files or tables that you plan to work with in Azure ML. You can create datasets from this page as well.
2. Create a dataset **from web files**, using the following settings:
   - **Basic Info**:
     - **Web URL**: https://aka.ms/diabetes-data
     - **Name**: diabetes-data
     - **Dataset type**: Tabular
     - **Description**: Diabetes data
     - **Skip data validation**: *Do not select*
   - **Settings and preview**:
     - **File format**: Delimited
     - **Delimiter**: Comma
     - **Encoding**: UTF-8
     - **Column headers**: Only first file has headers
     - **Skip rows**: None
     - **Dataset contains multi-line data**: *do not select*
   - **Schema**:

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

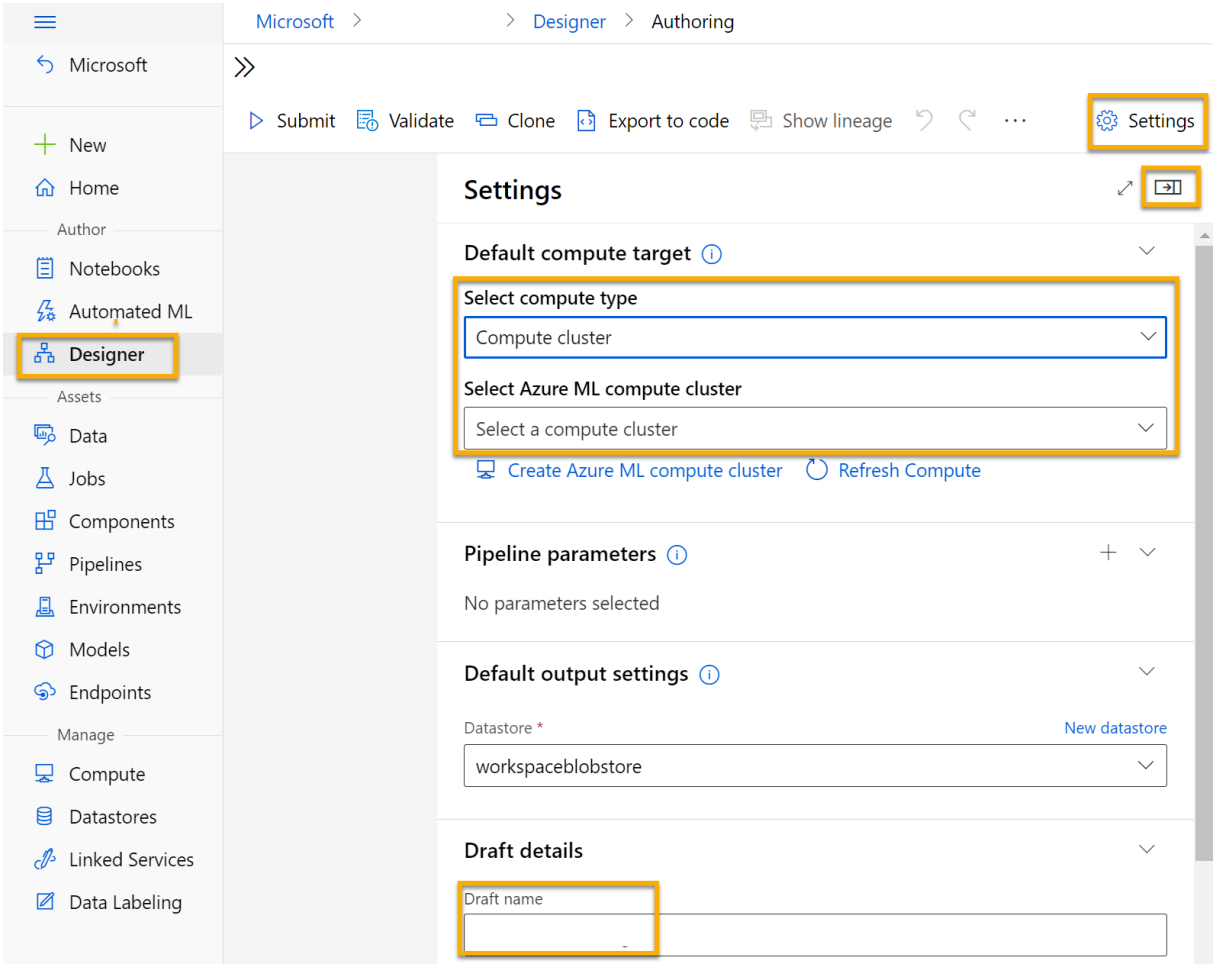
- **Confirm details**:
  - Do not profile the dataset after creation

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

Create a pipeline

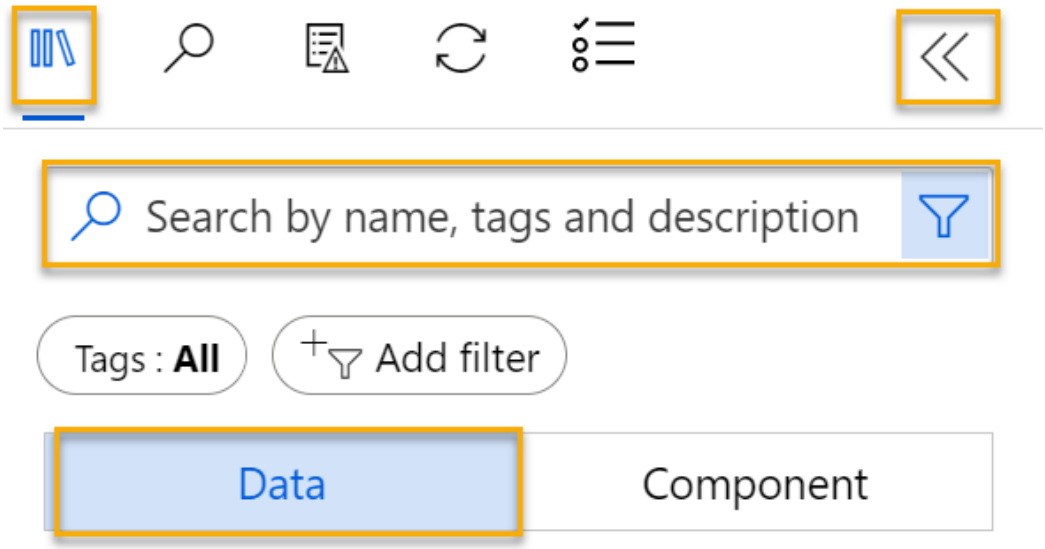
To get started with Azure Machine Learning designer, first you must create a pipeline and add the dataset you want to work with.

1. In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Designer** page (under **Author**), and select + to create a new pipeline.
2. At the top right-hand side of the screen, select **Settings**. If the **Settings** pane is not visible, select the wheel icon next to the pipeline name at the top.
3. In **Settings**, you must specify a compute target on which to run the pipeline. Under **Select compute type**, select **Compute cluster**. Then under **Select Azure ML *compute-type***, select the compute cluster you created previously.
4. In **Settings**, under **Draft Details**, change the draft name (**Pipeline-Created-on-*date***) to **Diabetes Training**.
5. Select the close icon on the top right of the **Settings** pane to close the pane.

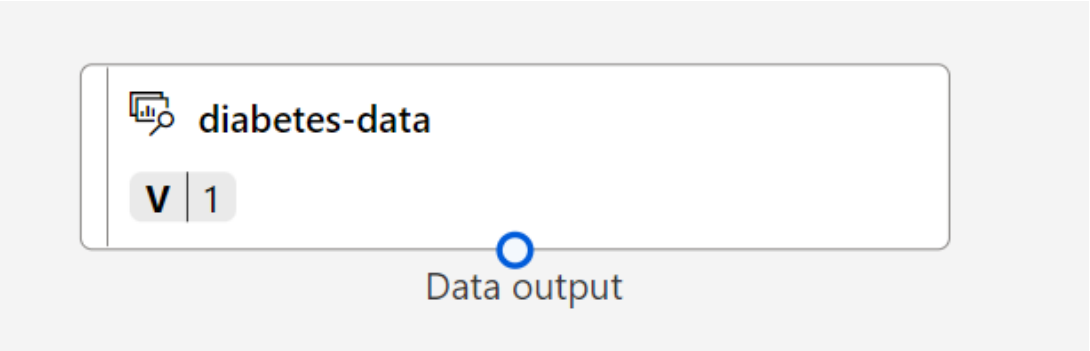


Load data to canvas

1. Next to the pipeline name on the left, select the arrows icon to expand the panel if it is not already expanded. The panel should open by default to the **Asset Library** pane, indicated by the books icon at the top of the panel. Note that there is a search bar to locate assets. Notice two buttons, **Data** and **Components**.



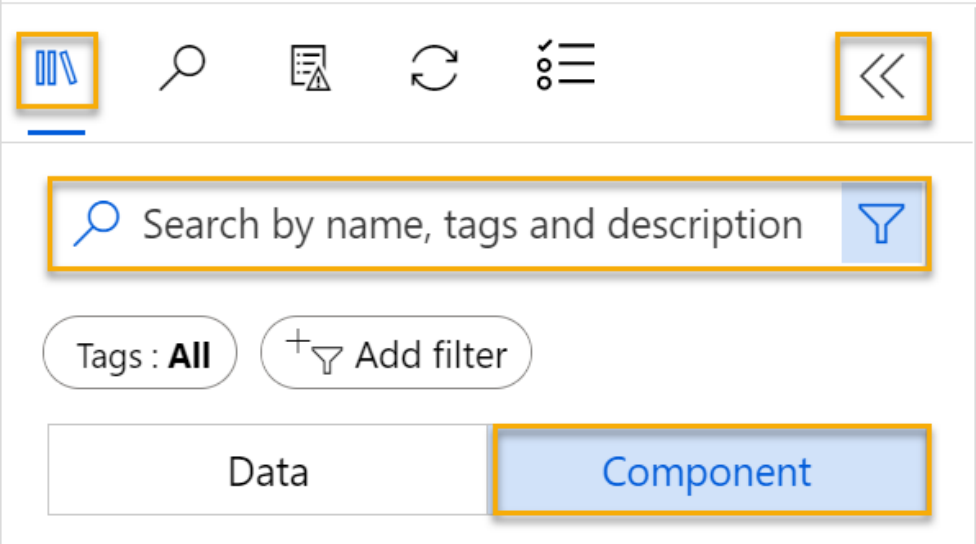
2. Click on **Data**. Search for and place the **diabetes-data** dataset onto the canvas.
3. Right-click (Ctrl+click on a Mac) the **diabetes-data** dataset on the canvas, and click on **Preview data**.
4. Review the schema of the data in the *Profile* tab, noting that you can see the distributions of the various columns as histograms.
5. Scroll down and select the column heading for the **Diabetic** column, and note that it contains two values **0** and **1**. These values represent the two possible classes for the *label* that your model will predict, with a value of **0** meaning that the patient does not have diabetes, and a value of **1** meaning that the patient is diabetic.
6. Scroll back up and review the other columns, which represent the *features* that will be used to predict the label. Note that most of these columns are numeric, but each feature is on its own scale. For example, **Age** values range from 21 to 77, while **DiabetesPedigree** values range from 0.078 to 2.3016. When training a machine learning model, it is sometimes possible for larger values to dominate the resulting predictive function, reducing the influence of features that on a smaller scale. Typically, data scientists mitigate this possible bias by *normalizing* the numeric columns so they're on the similar scales.
7. Close the **diabetes-data result visualization** window so that you can see the dataset on the canvas like this:



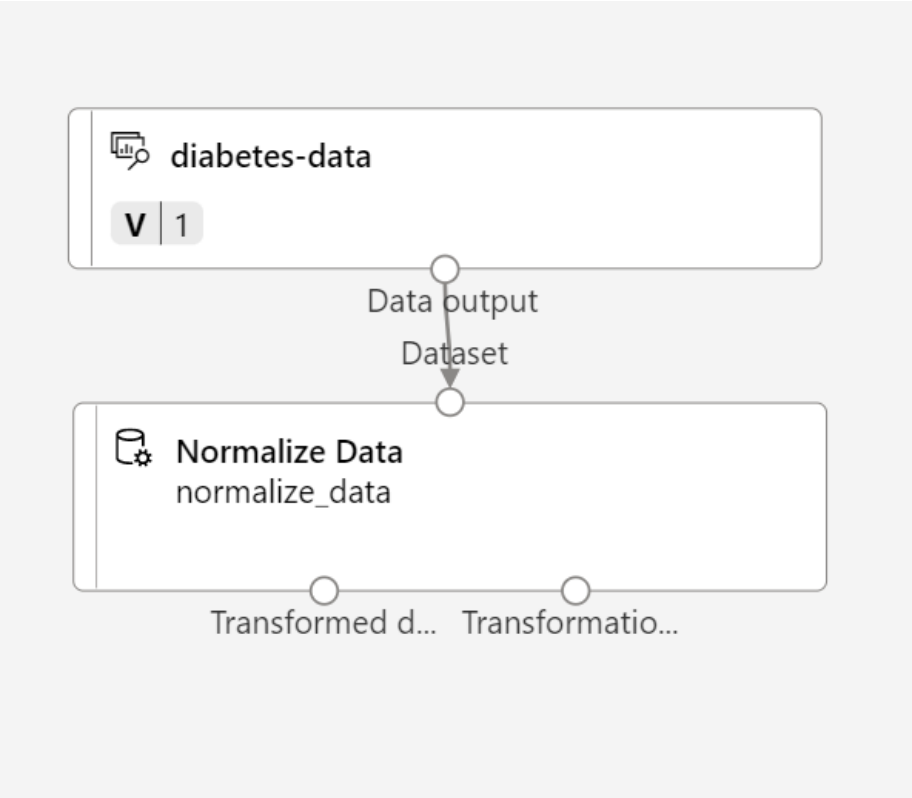
Add transformations

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

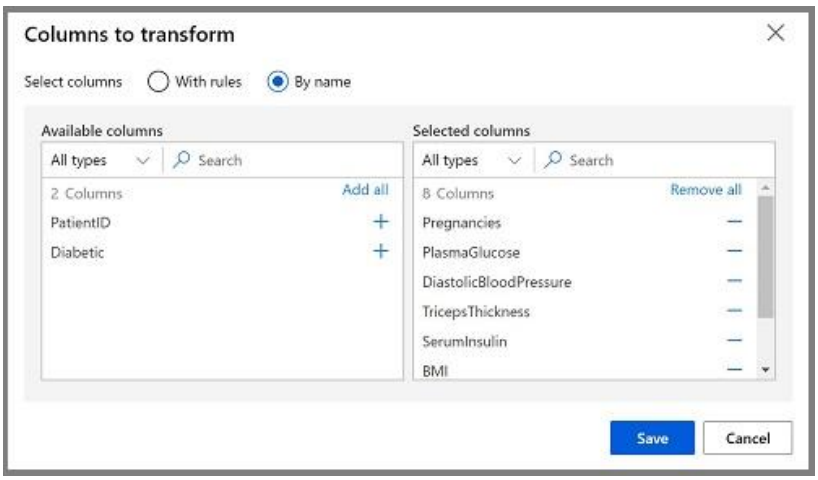
1. In the **Asset Library** pane on the left, click on **Components**, which contain a wide range of modules you can use for data transformation and model training. You can also use the search bar to quickly locate modules.



2. Find the **Normalize Data** module and place it on the canvas, below the **diabetes-data** dataset. Then connect the output from the bottom of the **diabetes-data** dataset to the input at the top of the **Normalize Data** module, like this:



3. Double-click the **Normalize Data** module to view its settings, noting that it requires you to specify the transformation method and the columns to be transformed.
4. Set the transformation to **MinMax** and edit the columns to include the following columns by name, as shown in the image:
    - **Pregnancies**
    - **PlasmaGlucose**
    - **DiastolicBloodPressure**
    - **TricepsThickness**
    - **SerumInsulin**
    - **BMI**
    - **DiabetesPedigree**
    - **Age**

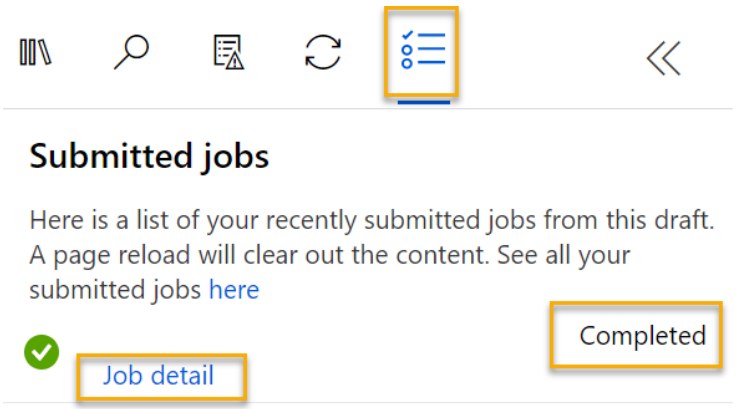


The data transformation is normalizing the numeric columns to put them on the same scale, which should help prevent columns with large values from dominating model training. You'd usually apply a whole bunch of pre-processing transformations like this to prepare your data for training, but we'll keep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

Run the pipeline

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

1. Select **Submit**, and run the pipeline as a new experiment named **mslearn-diabetes-training** on your compute cluster.
2. Wait for the run to finish - this may take a few minutes.



Notice that the left hand panel is now on the **Submitted Jobs** pane. You will know when the run is complete because the status of the job will change to **Complete**.

View the transformed data

1. When the run has completed, the dataset is now prepared for model training. Click on **Job Details**. You will be taken to another window.
2. Right-click (Ctrl+click on a Mac) the **Normalize data** module on the canvas, and click on **Preview data**. Select **Transformed dataset**.
3. View the data, noting that the numeric columns you selected have been normalized to a common scale.
4. Close the normalized data result visualization. Return to the previous window.

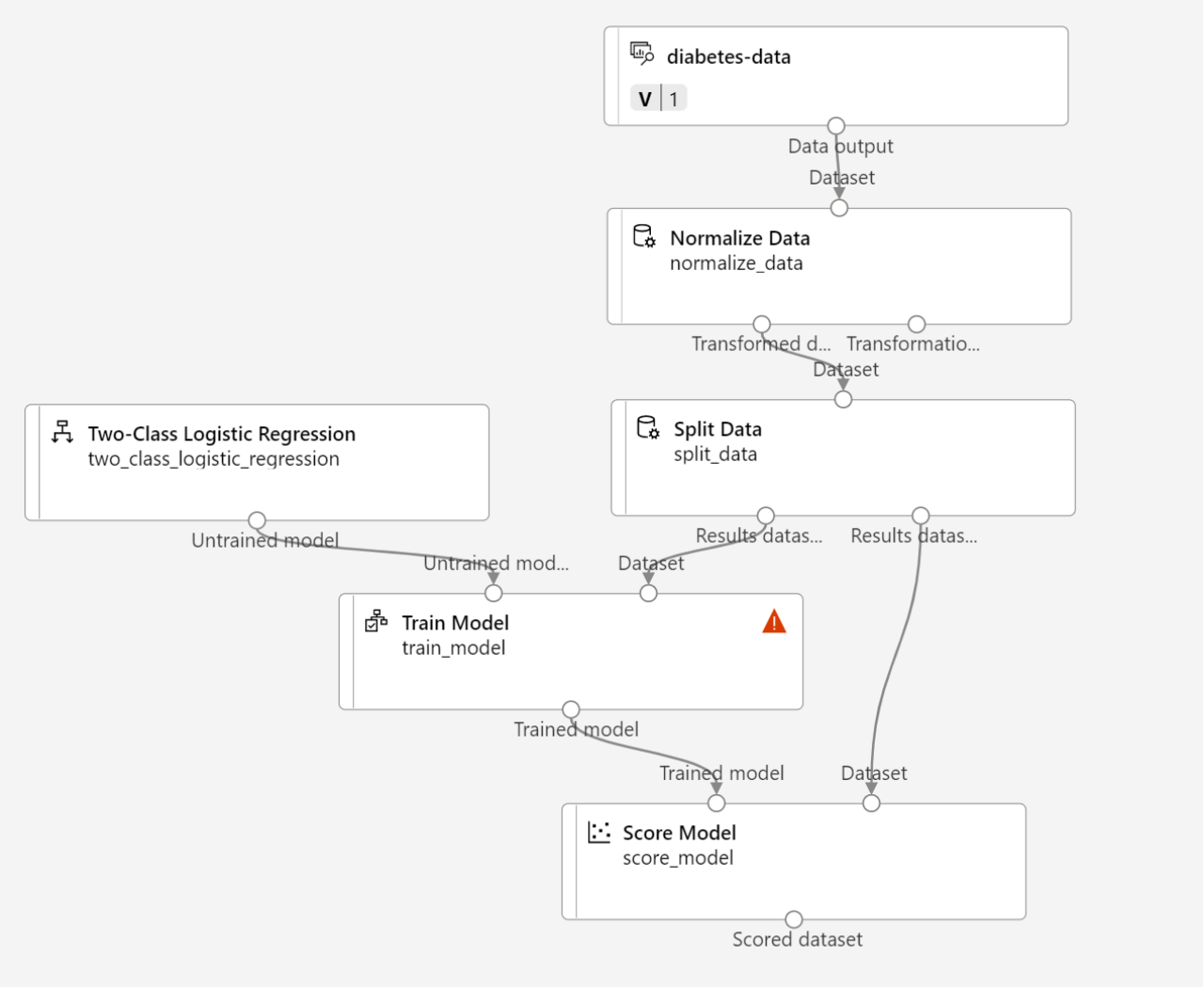
Create and run a training pipeline

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

Add training modules

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

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



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

1. Open the **Diabetes Training** pipeline you created in the previous unit if it's not already open.
2. In the **Asset Library** pane on the left, in **Components**, search for and place a **Split Data** module onto the canvas under the **Normalize Data** module. Then connect the *Transformed Dataset* (left) output of the **Normalize Data** module to the input of the **Split Data** module.

   Tip

   Use the search bar to quickly locate modules.
3. Select the **Split Data** module, and configure its settings as follows:
   - **Splitting mode** Split Rows
   - **Fraction of rows in the first output dataset**: 0.7
   - **Randomized split**: True
   - **Random seed**: 123
   - **Stratified split**: False
4. In the **Asset Library**, search for and place a **Train Model** module to the canvas, under the **Split Data** module. Then connect the *Result dataset1* (left) output of the **Split Data** module to the *Dataset* (right) input of the **Train Model** module.
5. The model we're training will predict the **Diabetic** value, so select the **Train Model** module and modify its settings to set the **Label column** to **Diabetic**.
6. The **Diabetic** label the model will predict is a class (0 or 1), so we need to train the model using a *classification* algorithm. Specifically, there are two possible classes, so we need a *binary classification* algorithm. In the **Asset Library**, search for and place a **Two-Class Logistic**

Regression module to the canvas, to the left of the **Split Data** module and above the **Train Model** module. Then connect its output to the **Untrained model** (left) input of the **Train Model** module.

Note

There are multiple algorithms you can use to train a classification model. For help choosing one, take a look at the **Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer**.

7. To test the trained model, we need to use it to *score* the validation dataset we held back when we split the original data - in other words, predict labels for the features in the validation dataset. In the **Asset Library**, search for and place a **Score Model** module to the canvas, below the **Train Model** module. Then connect the output of the **Train Model** module to the **Trained model** (left) input of the **Score Model** module; and connect the **Results dataset2** (right) output of the **Split Data** module to the **Dataset** (right) input of the **Score Model** module.

Run the training pipeline

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

1. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-diabetes-training**.
2. Wait for the experiment run to finish. This may take 5 minutes or more.
3. When the experiment run has finished, select **Job details**. You'll be directed to another window.
4. On the new window, right-click (Ctrl+click on a Mac) the **Score Model** module on the canvas, and click on **Preview data**. Select **Scored dataset** to view the results.
5. Scroll to the right, and note that next to the **Diabetic** column (which contains the known true values of the label) there is a new column named **Scored Labels**, which contains the predicted label values, and a **Scored Probabilities** columns containing a probability value between 0 and 1. This indicates the probability of a *positive* prediction, so probabilities greater than 0.5 result in a predicted label of ***1*** (diabetic), while probabilities between 0 and 0.5 result in a predicted label of ***0*** (not diabetic).
6. Close the **Score Model result visualization** window.

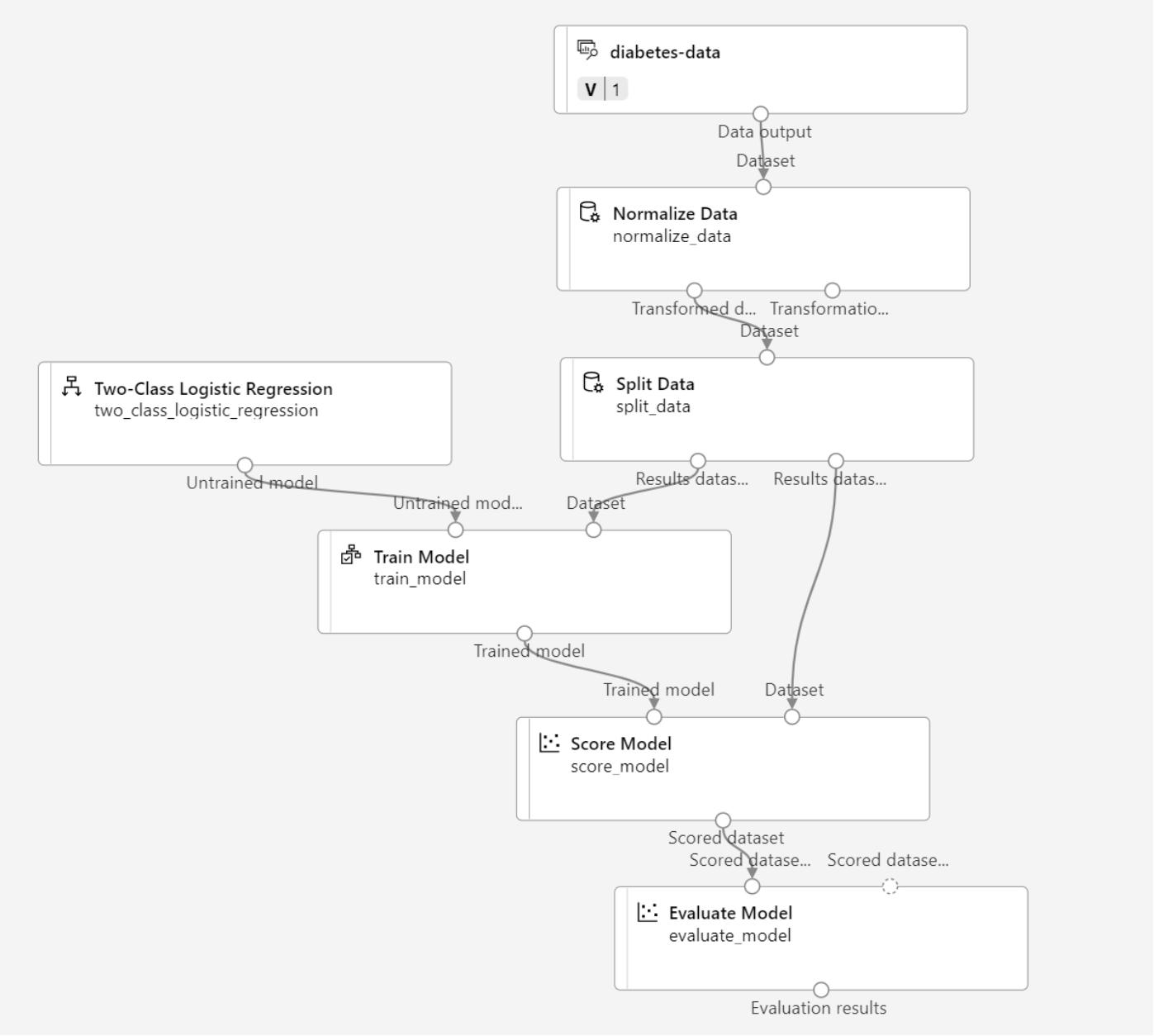
The model is predicting values for the **Diabetic** label, but how reliable are its predictions? To assess that, you need to evaluate the model.

Evaluate a classification model

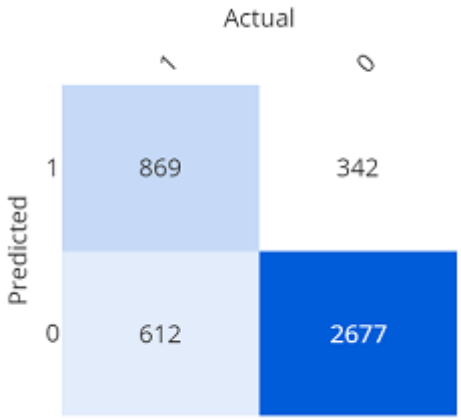
The validation data you held back and used to score the model includes the known values for the label. So to validate the model, you can compare the true values for the label to the label values that were predicted when you scored the validation dataset. Based on this comparison, you can calculate various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

Add an Evaluate Model module

1. Open the **Diabetes Training** pipeline you created in the previous unit if it's not already open.
2. In the **Asset Library**, search for and place an **Evaluate Model** module to the canvas, under the **Score Model** module, and connect the output of the **Score Model** module to the **Scored dataset** (left) input of the **Evaluate Model** module.
3. Ensure your pipeline looks like this:



4. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-diabetes-training**.
5. Wait for the experiment run to finish.
6. When the experiment run has finished, select **Job details**. You'll be directed to another window.
7. On the new window, right-click (Ctrl+click on a Mac) the **Evaluate Model** module on the canvas, and click on **Preview data**. Select **Evaluation results** to view the performance metrics. These metrics can help data scientists assess how well the model predicts based on the validation data.
8. Scroll down to view the *confusion matrix* for the model, which is a tabulation of the predicted and actual value counts for each possible class. For a binary classification model like this one, where you're predicting one of two possible values, the confusion matrix is a 2x2 grid showing the predicted and actual value counts for classes **0** and **1**, similar to this:



The confusion matrix shows cases where both the predicted and actual values were 1 (known as *true positives*) at the top left, and cases where both the predicted and the actual values were 0 (*true negatives*) at the bottom right. The other cells show cases where the predicted and actual values differ (*false positives* and *false negatives*). The cells in the matrix are colored so that the more cases represented in the cell, the more intense the color - with the result that you can identify a model that predicts accurately for all classes by looking for a diagonal line of intensely colored cells from the top left to the bottom right (in other words, the cells where the predicted values match the actual values). For a multi-class classification model (where there are more than two possible classes), the same approach is used to tabulate each possible combination of actual and predicted value counts - so a model with three possible classes would result in a 3x3 matrix with a diagonal line of cells where the predicted and actual labels match.

8. Review the metrics to the left of the confusion matrix, which include:

   - **Accuracy**: The ratio of correct predictions (true positives + true negatives) to the total number of predictions. In other words, what proportion of diabetes predictions did the model get right?
   - **Precision**: The fraction of positive cases correctly identified (the number of true positives divided by the number of true positives plus false positives). In other words, out of all the patients that the model predicted as having diabetes, how many are actually diabetic?
   - **Recall**: The fraction of the cases classified as positive that are actually positive (the number of true positives divided by the number of true positives plus false negatives). In other words, out of all the patients who actually have diabetes, how many did the model identify?
   - **F1 Score**: An overall metric that essentially combines precision and recall.
   - *We'll return to **AUC** later.*

   Of these metric, *accuracy* is the most intuitive. However, you need to be careful about using simple accuracy as a measurement of how well a model works. Suppose that only 3% of the population is diabetic. You could create a model that always predicts **0** and it would be 97% accurate - just not very useful! For this reason, most data scientists use other metrics like precision and recall to assess classification model performance.

9. Above the list of metrics, note that there's a **Threshold** slider. Remember that what a classification model predicts is the probability for each possible class. In the case of this binary classification model, the predicted probability for a *positive* (that is, diabetic) prediction is a value between 0 and 1. By default, a predicted probability for diabetes *including or above* 0.5 results in a class prediction of 1, while a prediction *below* this threshold means that there's a greater probability of the patient **not** having diabetes (remember that the probabilities for all classes add up to 1), so the predicted class would be 0. Try moving the threshold slider and observe the effect on the confusion matrix. If you move it all the way to the left (0), the Recall metric becomes 1, and if you move it all the way to the right (1), the Recall metric becomes 0.

10. Look above the Threshold slider at the **ROC curve** (ROC stands for *receiver operating characteristic*, but most data scientists just call it a ROC curve). Another term for *recall* is **True positive rate**, and it has a corresponding metric named **False positive rate**, which measures the number of negative cases incorrectly identified as positive compared the number of actual negative cases. Plotting these metrics against each other for every possible threshold value between 0 and 1 results in a curve. In an ideal model, the curve would go all the way up the left side and across the top, so that it covers the full area of the chart. The larger the *area under the curve* (which can be any value from 0 to 1), the better the model is performing - this is the **AUC** metric listed with the other metrics below. To get an idea of how this area represents the performance of the model, imagine a straight diagonal line from the bottom left to the top right of the ROC chart. This represents the expected performance if you just guessed or flipped a coin for each patient - you could expect to get around half of them right, and half of them wrong, so the area under the diagonal line represents an AUC of 0.5. If the AUC for your model is higher than this for a binary classification model, then the model performs better than a random guess.

11. Close the **Evaluate Model result visualization** window.

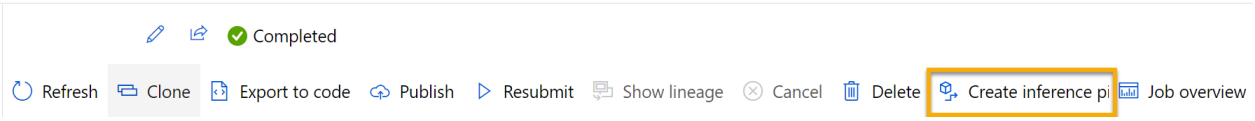
The performance of this model isn't all that great, partly because we performed only minimal feature engineering and pre-processing. You could try a different classification algorithm, such as **Two-Class Decision Forest**, and compare the results. You can connect the outputs of the **Split Data** module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

Create an inference pipeline

After creating and running a pipeline to train the model, you need a second pipeline known as an inference pipeline. The inference pipeline performs the same data transformations as the first pipeline for *new* data. Then it uses the trained model to *infer*, or predict, label values based on its features. This model will form the basis for a predictive service that you can publish for applications to use.

Create an inference pipeline

1. In Azure Machine Learning studio, expand the left-hand pane by selecting the three lines at the top left of the screen. Click on **Jobs** (under **Assets**) to view all of the jobs you have run. Select the experiment **mslearn-diabetes-training**, then select the **mslearn-diabetes-training** pipeline.
2. Locate the menu above the canvas and click on **Create inference pipeline**. You may need to expand your screen to full and click on the three dots icon **...** on the top right hand corner of the screen in order to find **Create inference pipeline** in the menu.

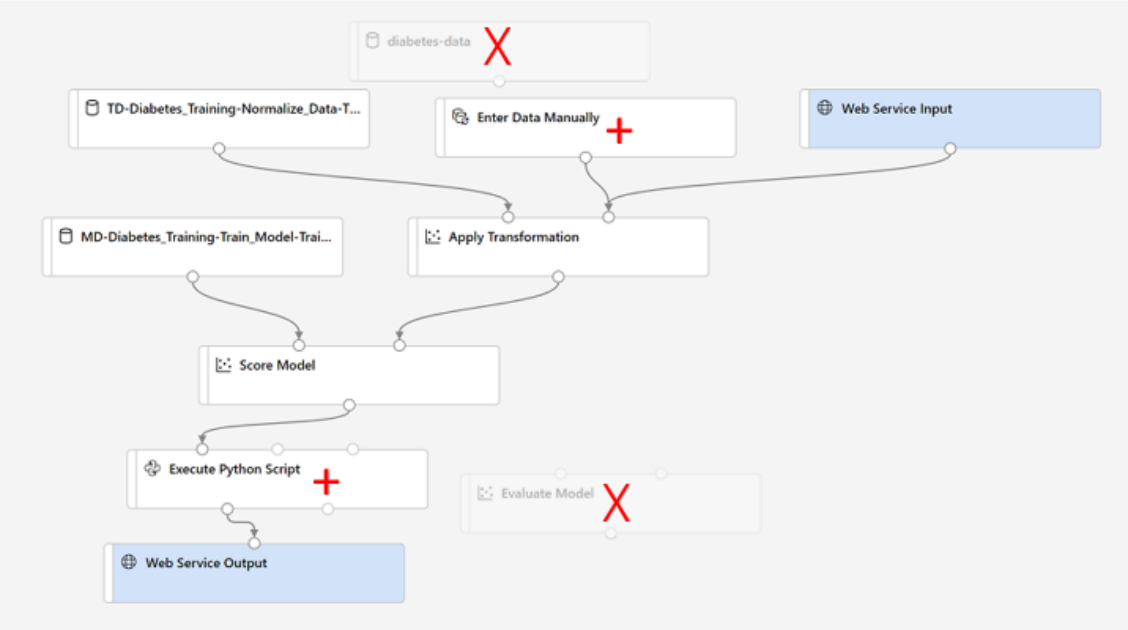


3. In the **Create inference pipeline** drop-down list, click **Real-time inference pipeline**. After a few seconds, a new version of your pipeline named **Diabetes Training-real time inference** will be opened.

   *If the pipeline doesn't include **Web Service Input** and **Web Service Output** modules, go back to the **Designer** page and then reopen the **Diabetes Training-real time inference** pipeline.*

4. Navigate to **Settings** on the upper right hand menu. Under **Draft details**, rename the new pipeline to **Predict Diabetes**, and then review the new pipeline. It contains a web service input for new data to be submitted, and a web service output to return results. Some of the transformations and training steps are a part of this pipeline. The trained model will be used to score the new data.

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



- Replace the **diabetes-data** dataset with an **Enter Data Manually** module that doesn't include the label column (**Diabetic**).
- Remove the **Evaluate Model** module.
- Insert an **Execute Python Script** module before the web service output to return only the patient ID, predicted label value, and probability.

  Follow the remaining steps below, using the image and information above for reference as you modify the pipeline.

5. The inference pipeline assumes that new data will match the schema of the original training data, so the **diabetes-data** dataset from the training pipeline is included. However, this input data includes the **Diabetic** label that the model predicts, which is not included in new patient data for which a diabetes prediction hasn't yet been made. Delete this module and replace it with an **Enter Data Manually** module, containing the following CSV data, which includes feature values without labels for three new patient observations:

CSVCopy

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

6. Connect the new **Enter Data Manually** module to the same **Dataset** input of the **Apply Transformation** module as the **Web Service Input**.

7. The inference pipeline includes the **Evaluate Model** module, which isn't useful when predicting from new data, so delete this module.

8. The output from the **Score Model** module includes all of the input features and the predicted label and probability score. To limit the output to only the prediction and probability:

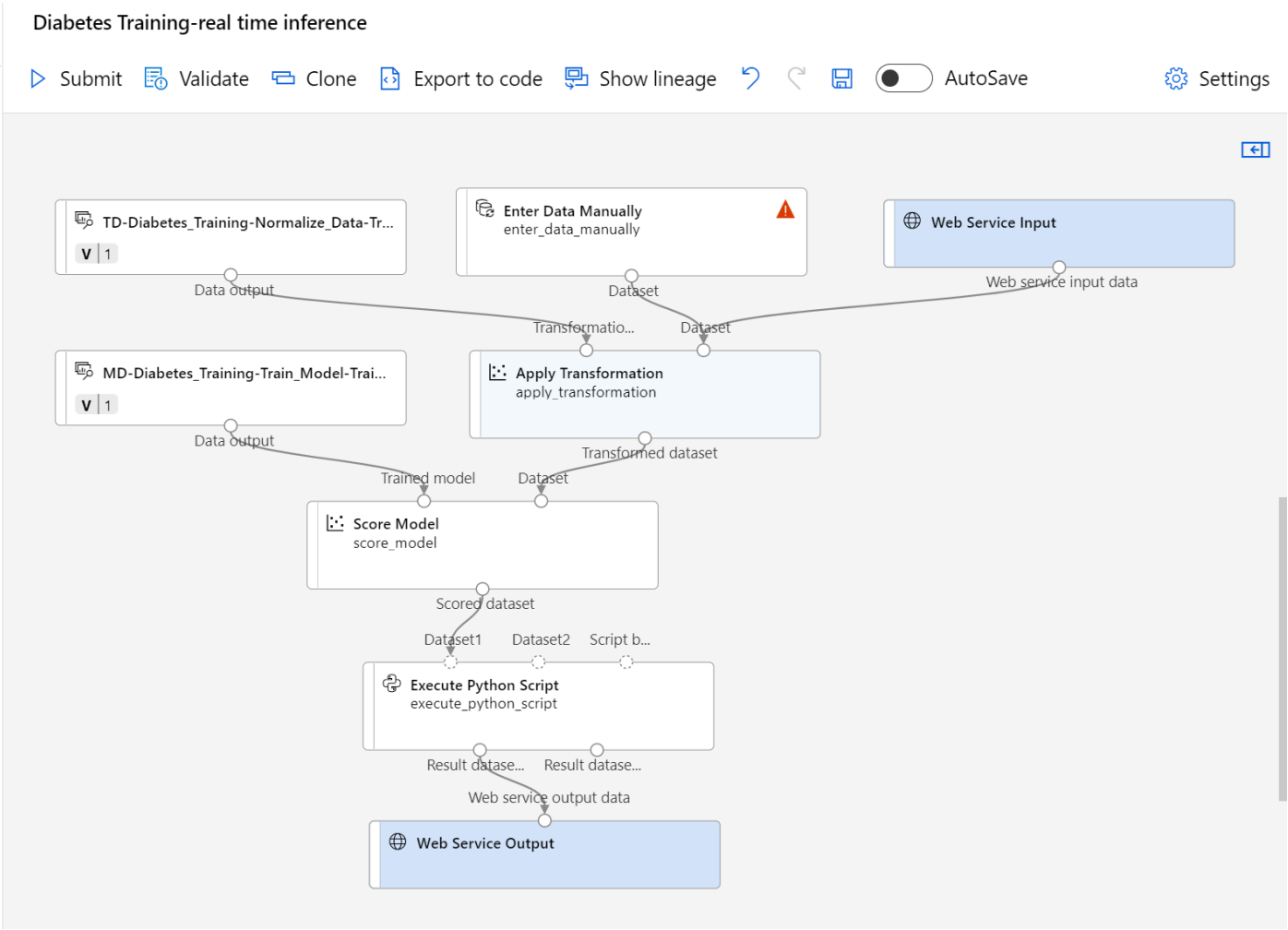
- Delete the connection between the **Score Model** module and the **Web Service Output**.
- Add an **Execute Python Script** module, replacing all of the default python script with the following code (which selects only the **PatientID**, **Scored Labels** and **Scored Probabilities** columns and renames them appropriately):

PythonCopy

```
import pandas as pd

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

    scored_results = dataframe1[['PatientID', 'Scored Labels', 'Scored Probabilities']]
    scored_results.rename(columns={'Scored Labels':'DiabetesPrediction',
                                   'Scored Probabilities':'Probability'},
                          inplace=True)
    return scored_results
```

9. Connect the output from the **Score Model** module to the **Dataset1** (left-most) input of the **Execute Python Script**, and connect the output of the **Execute Python Script** module to the **Web Service Output**.

10. Verify that your pipeline looks similar to the following image:



11. Run the pipeline as a new experiment named **mslearn-diabetes-inference** on your compute cluster. The experiment may take a while to run.

12. When the pipeline has completed, select **Job details**. In the new window, right click the **Execute Python Script** module. Select the **Preview Data** and select **Result dataset** to see the predicted labels and probabilities for the three patient observations in the input data.

Your inference pipeline predicts whether or not patients are at risk for diabetes based on their features. Now you're ready to publish the pipeline so that client applications can use it.

Deploy a predictive service

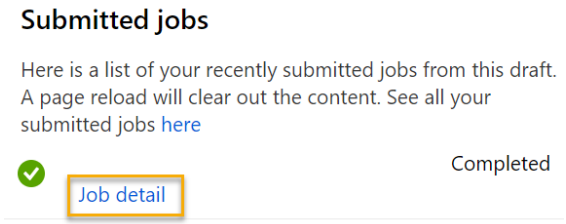
After you've created and tested an inference pipeline for real-time inferencing, you can publish it as a service for client applications to use.

Note

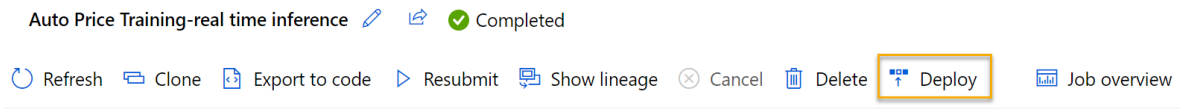
In this exercise, you'll deploy the web service to an Azure Container Instance (ACI). This type of compute is created dynamically, and is useful for development and testing. For production, you should create an *inference cluster*, which provide an Azure Kubernetes Service (AKS) cluster that provides better scalability and security.

Deploy a service

1. View the **Predict Diabetes** inference pipeline you created in the previous unit.
2. Select **Job detail** on the left hand pane. This will open up another window.



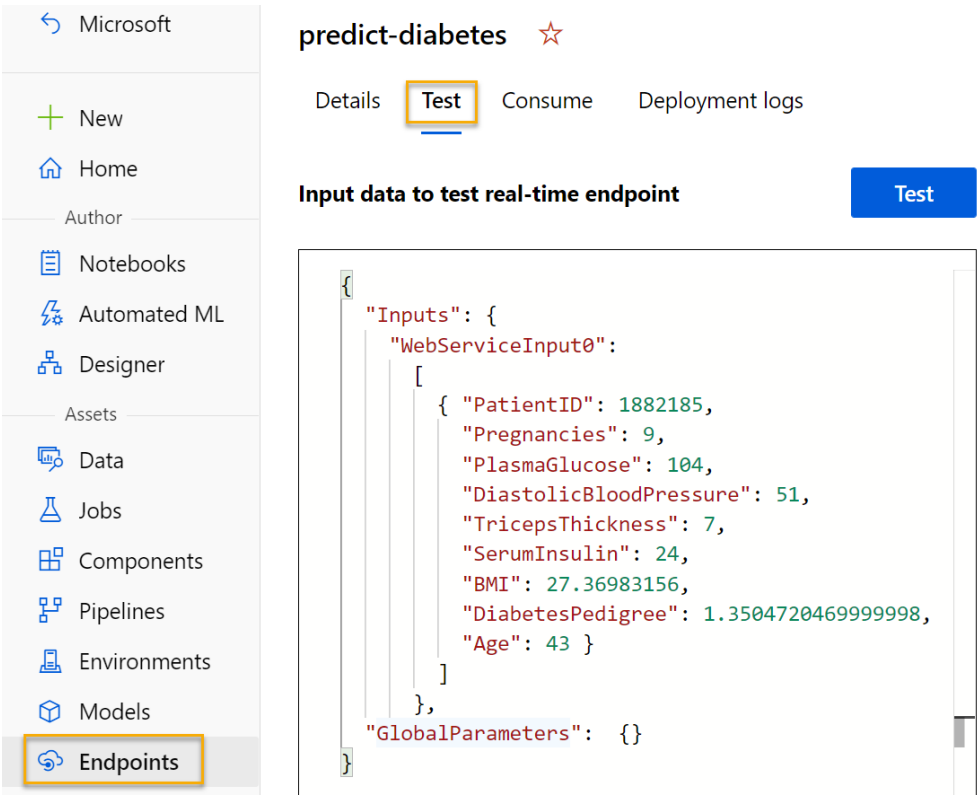
3. In the new window, select **Deploy**.



4. At the top right, select **Deploy**, and deploy a **new real-time endpoint**, using the following settings:
    - **Name**: predict-diabetes
    - **Description**: Classify diabetes
    - **Compute type**: Azure Container Instance
5. Wait for the web service to be deployed - this can take several minutes. The deployment status is shown at the top left of the designer interface.

Test the service

1. On the **Endpoints** page, open the **predict-diabetes** real-time endpoint.



2. When the **predict-diabetes** endpoint opens, select the **Test** tab. We will use it to test our model with new data. Delete the current data under **Input data to test real-time endpoint**. Copy and paste the below data into the data section:

JSONCopy

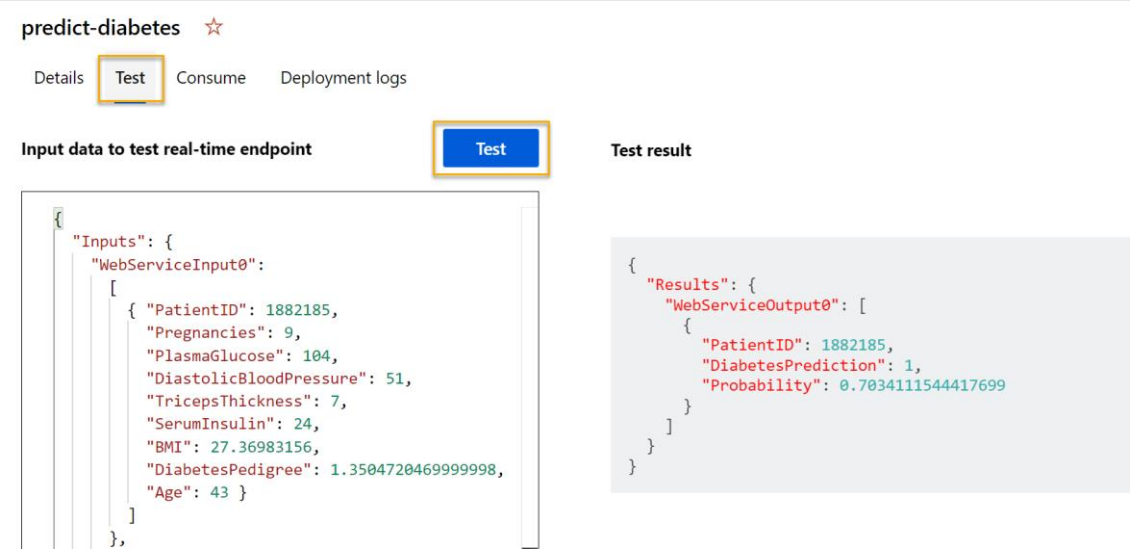
```
{
 "Inputs": {
  "WebServiceInput0":
   [
    { "PatientID": 1882185,
     "Pregnancies": 9,
```

```
        "PlasmaGlucose": 104,
        "DiastolicBloodPressure": 51,
        "TricepsThickness": 7,
        "SerumInsulin": 24,
        "BMI": 27.36983156,
        "DiabetesPedigree": 1.3504720469999998,
        "Age": 43 }
      ]
    },
  "GlobalParameters":  {}
}
```


Note

The JSON above defines features for a patient, and uses the **predict-diabetes** service you created to predict a diabetes diagnosis.

3. Select **Test**. On the right hand of the screen, you should see the output '**DiabetesPrediction**'. The output is 1 if the patient is predicted to have diabetes, and 0 if the patient is predicted not to have diabetes.



You have just tested a service that is ready to be connected to a client application using the credentials in the **Consume** tab. We will end the lab here. You are welcome to continue to experiment with the service you just deployed.

Knowledge check

1. You're using Azure Machine Learning designer to create a training pipeline for a binary classification model. You've added a dataset containing features and labels, a Two-Class Decision Forest module, and a Train Model module. You plan to use Score Model and Evaluate Model modules to test the trained model with a subset of the dataset that wasn't used for training. What's another module should you add?

- Join Data
- Split Data
- Select Columns in Dataset

2. You use an Azure Machine Learning designer pipeline to train and test a binary classification model. You review the model's performance metrics in an Evaluate Model module, and note that it has an AUC score of 0.3. What can you conclude about the model?

- The model can explain 30% of the variance between true and predicted labels.
- The model predicts accurately for 70% of test cases.
- The model performs worse than random guessing.

3. You use Azure Machine Learning designer to create a training pipeline for a classification model. What must you do before deploying the model as a service?

- Create an inference pipeline from the training pipeline
- Add an Evaluate Model module to the training pipeline
- Clone the training pipeline with a different name

Summary

In this module, you learned how to use Azure Machine Learning designer to train and publish a classification model.

Clean-up

The web service you created is hosted in an *Azure Container Instance*. If you don't intend to experiment with it further, you should delete the endpoint to avoid accruing unnecessary Azure usage. You should also stop the compute instance until you need it again.

1. In Azure Machine Learning studio, on the **Endpoints** tab, select the **predict-diabetes** endpoint. Then select **Delete** (🗑) and confirm that you want to delete the endpoint.
2. On the **Compute** page, on the **Compute Instances** tab, select your compute instance and then select **Stop**.

Note

Stopping your compute ensures your subscription won't be charged for compute resources. You will however be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription. If you have finished exploring Azure Machine Learning, you can delete the Azure Machine Learning workspace and associated resources. However, if you plan to complete any other labs in this series, you will need to recreate it.

To delete your workspace:

1. In the **Azure portal**, in the **Resource groups** page, open the resource group you specified when creating your Azure Machine Learning workspace.
2. Click **Delete resource group**, type the resource group name to confirm you want to delete it, and select **Delete**.

Module complete

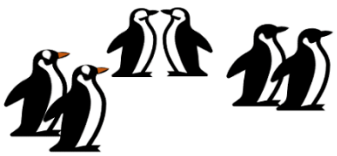
Part 2/6_4/4

Create a Clustering Model with Azure Machine Learning designer

Clustering is an unsupervised machine learning technique used to group similar entities based on their features. Learn how to create clustering models using Azure Machine Learning designer.

Introduction

Clustering is a form of machine learning that is used to group similar items into clusters based on their features. For example, a researcher might take measurements of penguins, and group them based on similarities in their proportions.



Clustering is an example of *unsupervised* machine learning, in which you train a model to separate items into clusters based purely on their characteristics, or *features*. There is no previously known cluster value (or *label*) from which to train the model.

You can use Microsoft Azure Machine Learning designer to create clustering models by using a drag and drop visual interface, without needing to write any code.

In this module, you'll learn how to:

- Use Azure Machine Learning designer to train a clustering model.
- Use a clustering model for inferencing.
- Deploy a clustering model as a service.

To complete this module, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.microsoft.com.

Create an Azure Machine Learning workspace

Azure Machine Learning is a cloud-based platform for building and operating machine learning solutions in Azure. It includes a wide range of features and capabilities that help data scientists prepare data, train models, publish predictive services, and monitor their usage. One of these features is a visual interface called *designer*, that you can use to train, test, and deploy machine learning models without writing any code.

Create an Azure Machine Learning workspace

To use Azure Machine Learning, you create a *workspace* in your Azure subscription. You can then use this workspace to manage data, compute resources, code, models, and other artifacts related to your machine learning workloads.

Note

This module is one of many that make use of an Azure Machine Learning workspace, including the other modules in the **Microsoft Azure AI Fundamentals: Explore visual tools for machine learning** learning path. If you are using your own Azure subscription, you may consider creating the workspace once and reusing it in other modules. Your Azure subscription will be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription, so we recommend you delete the Azure Machine Learning workspace when it is no longer required.

If you don't already have one, follow these steps to create a workspace:

1. Sign into the Azure portal using your Microsoft credentials.
2. Select **Create a resource**, search for *Machine Learning*, and create a new **Azure Machine Learning** resource with an *Azure Machine Learning* plan. Use the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Workspace name**: *Enter a unique name for your workspace*
    - **Region**: *Select the closest geographical region*
    - **Storage account**: *Note the default new storage account that will be created for your workspace*
    - **Key vault**: *Note the default new key vault that will be created for your workspace*
    - **Application insights**: *Note the default new application insights resource that will be created for your workspace*
    - **Container registry**: None (*one will be created automatically the first time you deploy a model to a container*)
3. Select **Review + create**. Wait for your workspace to be created (it can take a few minutes). Then go to it in the portal.
4. On the **Overview** page for your workspace, launch Azure Machine Learning studio (or open a new browser tab and navigate to https://ml.azure.com), and sign into Azure Machine Learning studio using your Microsoft account.

5. In Azure Machine Learning studio, select the three lines icon at the top left to view the various pages in the interface. You can use these pages to manage the resources in your workspace.

You can manage your workspace using the Azure portal, but for data scientists and Machine Learning operations engineers, Azure Machine Learning studio provides a more focused user interface for managing workspace resources.

Create compute resources

To train and deploy models using Azure Machine Learning designer, you need compute targets to run the training process. You will also use these compute targets to test the trained model after its deployment.

Create compute targets

Compute targets are cloud-based resources on which you can run model training and data exploration processes.

In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Compute** page (under **Manage**). You manage the compute targets for your data science activities in the studio. There are four kinds of compute resource that you can create:

- **Compute Instances**: Development workstations that data scientists can use to work with data and models.
- **Compute Clusters**: Scalable clusters of virtual machines for on-demand processing of experiment code.
- **Inference Clusters**: Deployment targets for predictive services that use your trained models.
- **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

Note

Compute instances and clusters are based on standard Azure virtual machine images. For this module, the *Standard_DS11_v2* image is recommended to achieve the optimal balance of cost and performance. If your subscription has a quota that does not include this image, choose an alternative image; but bear in mind that a larger image may incur higher cost and a smaller image may not be sufficient to complete the tasks. Alternatively, ask your Azure administrator to extend your quota.

1. Navigate to the **Compute Clusters** tab and add a new compute cluster with the following settings:
    - **Location**: *Select the same as your workspace. If that location is not listed, choose the one closest to you*
    - **Virtual Machine tier**: Dedicated
    - **Virtual Machine type**: CPU
    - **Virtual Machine size**:
        - Choose **Select from all options**
        - Search for and select **Standard_DS11_v2**

    Select **Next**

    - **Compute name**: *enter a unique name*
    - **Minimum number of nodes**: 0
    - **Maximum number of nodes**: 2
    - **Idle seconds before scale down**: 120
    - **Enable SSH access**: Unselected

    Select **Create**

Tip

After you finish the entire module, be sure to follow the **Clean Up** instructions at the end of the module to stop your compute resources. Stop your compute resources to ensure your subscription won't be charged.

The compute targets take some time to be created. You can move onto the next unit while you wait.

Explore data

To train a clustering model, you need a dataset that includes multiple observations of the items you want to cluster, including numeric features that can be used to determine similarities between individual cases that will help separate them into clusters.

Create a dataset

In Azure Machine Learning, data for model training and other operations is usually encapsulated in an object called a *dataset*. In this module, you'll use a dataset that includes observations of three species of penguin.

1. In Azure Machine Learning studio, expand the left pane by selecting the three lines at the top left of the screen. View the **Data** page (under **Assets**). The Data page contains specific data files or tables that you plan to work with in Azure ML. You can create datasets from this page as well.
2. Create a dataset from web files, using the following settings:
    - **Basic Info**:
        - **Web URL**: https://aka.ms/penguin-data
        - **Name**: penguin-data
        - **Dataset type**: Tabular
        - **Description**: Penguin data
        - **Skip data validation**: *do not select*
    - **Settings and preview**:
        - **File format**: Delimited
        - **Delimiter**: Comma
        - **Encoding**: UTF-8
        - **Column headers**: Only first file has headers
        - **Skip rows**: None
        - **Dataset contains multi-line data**: *do not select*
    - **Schema**:
        - Include all columns other than **Path**
        - Review the automatically detected types
    - **Confirm details**:
        - Do not profile the dataset after creation

    After the dataset has been created, open it and view the **Explore** page to see a sample of the data. This data represents measurements of the culmen (bill) length and depth, flipper length, and body mass for multiple observations of penguins. There are three species of penguin represented in the dataset: *Adelie*, *Gentoo*, and *Chinstrap*.

Note

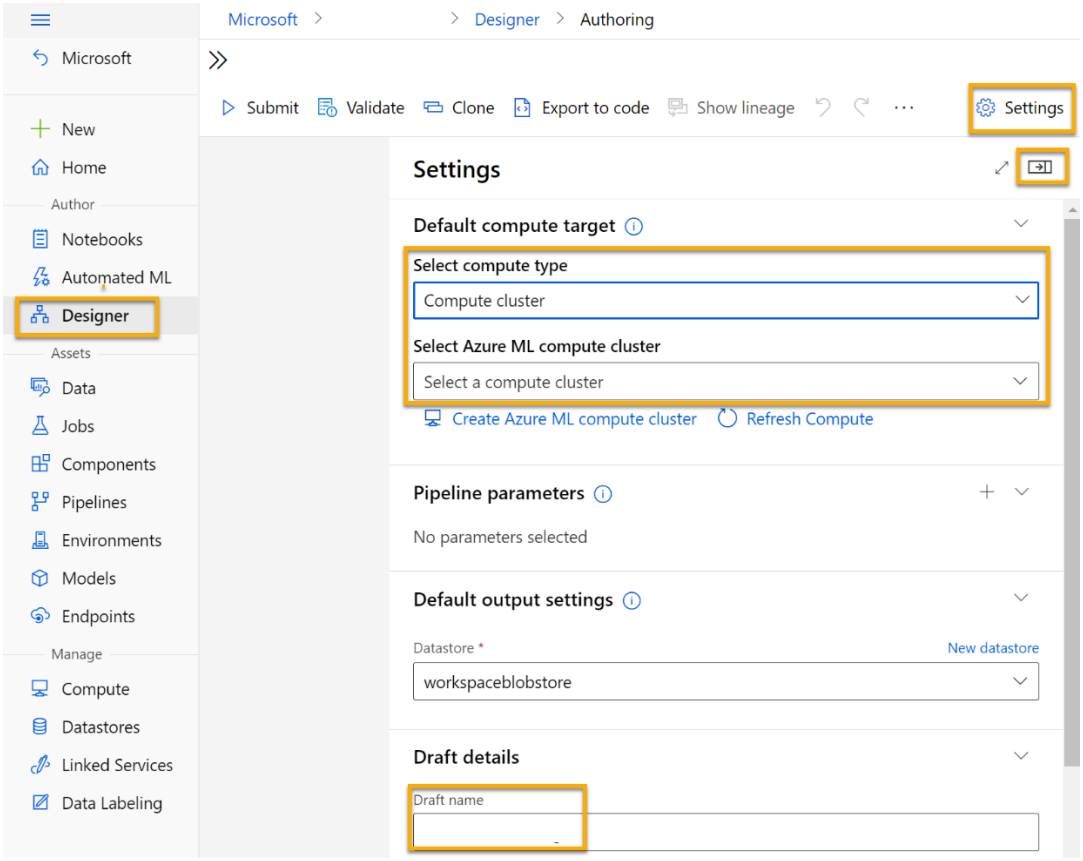
The penguins dataset used in the this exercise is a subset of data collected and made available by **Dr. Kristen Gorman** and the **Palmer Station, Antarctica LTER**, a member of the **Long Term Ecological Research Network**.

Create a pipeline

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

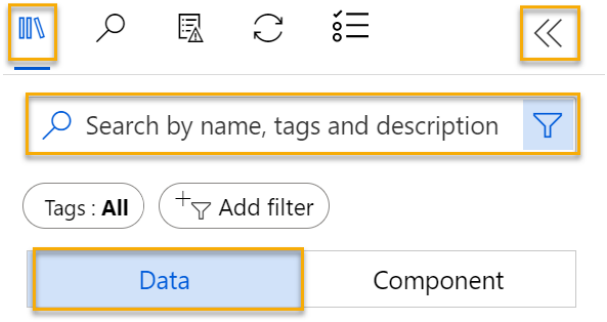
1. In Azure Machine Learning studio, expand the left pane by selecting the three lines icon at the top left of the screen. View the **Designer** page (under **Author**), and select the plus sign to create a new pipeline.
2. At the top right-hand side of the screen, select **Settings**. If the **Settings** pane is not visible, select the wheel icon next to the pipeline name at the top.

3. In **Settings**, you must specify a compute target on which to run the pipeline. Under **Select compute type**, select **Compute cluster**. Then under **Select Azure ML *compute-type***, select the compute cluster you created previously.
4. In **Settings**, under **Draft Details**, change the draft name (**Pipeline-Created-on-*date***) to **Train Penguin Clustering**
5. Select the *close icon* on the top right of the **Settings** pane to close the pane.

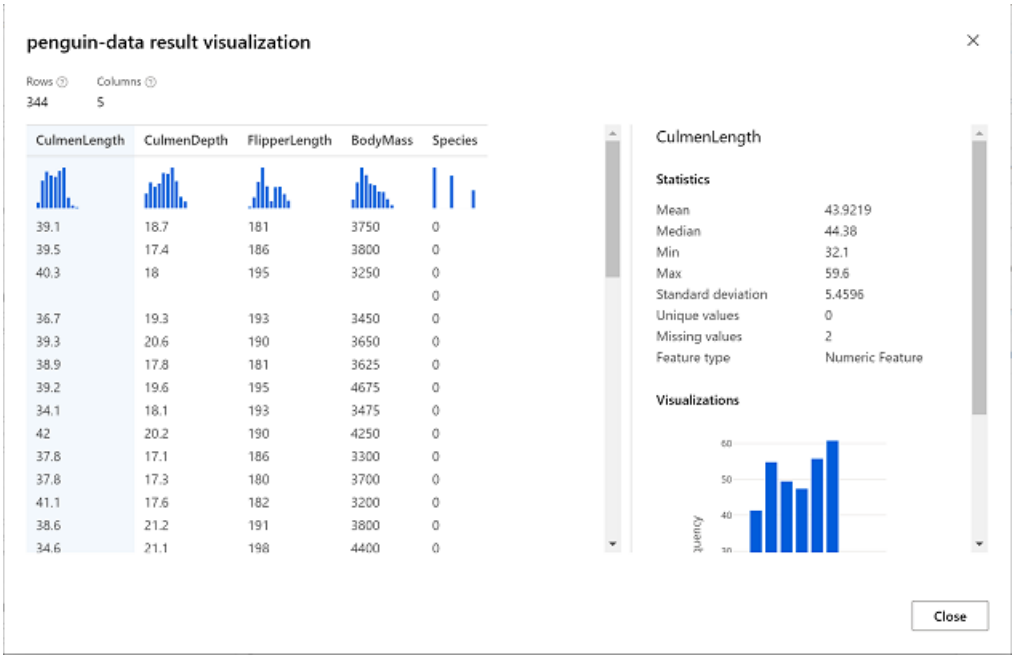


Load data to canvas

1. Next to the pipeline name on the left, select the arrows icon to expand the panel if it is not already expanded. The panel should open by default to the **Asset Library** pane, indicated by the books icon at the top of the panel. Note that there is a search bar to locate assets. Notice two buttons, **Data** and **Components**.



2. Click on **Data**. Search for and place the **penguin-data** dataset onto the canvas.
3. Right-click (Ctrl+click on a Mac) the **penguin-data** dataset on the canvas, and click on **Preview data**.
4. Review the *Profile* schema of the data, noting that you can see the distributions of the various columns as histograms. Then select the **CulmenLength** column. The dataset should look similar to this:



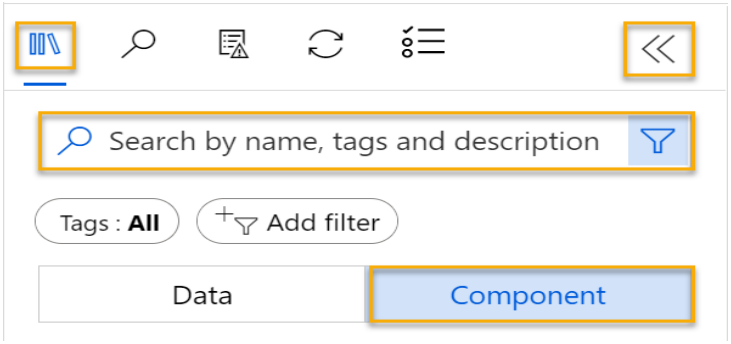
5. Note the following characteristics of the dataset:
    - The dataset includes the following columns:
        - **CulmenLength**: Length of the penguin's bill in millimeters.
        - **CulmenDepth**: Depth of the penguin's bill in millimeters.
        - **FlipperLength**: Length of the penguin's flipper in millimeters.
        - **BodyMass**: Weight of the penguin in grams.
        - **Species**: Species indicator (0:"Adelie", 1:"Gentoo", 2:"Chinstrap")
    - There are two missing values in the **CulmenLength** column (the **CulmenDepth**, **FlipperLength**, and **BodyMass** columns also have two missing values).
    - The measurement values are in different scales (from tens of millimeters to thousands of grams).

    Close the dataset visualization so you can see the dataset on the pipeline canvas.

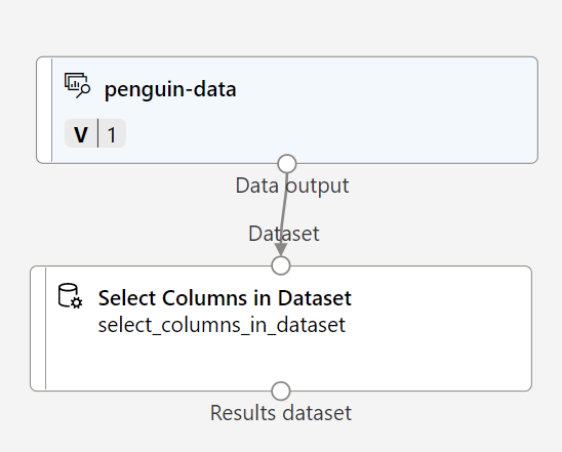
Apply transformations

To cluster the penguin observations, we're going to use only the measurements; so we'll discard the species column. We also need to remove rows where values are missing, and normalize the numeric measurement values so they're on a similar scale.

1. In the **Asset Library** pane on the left, click on **Components**, which contain a wide range of modules you can use for data transformation and model training. You can also use the search bar to quickly locate modules.

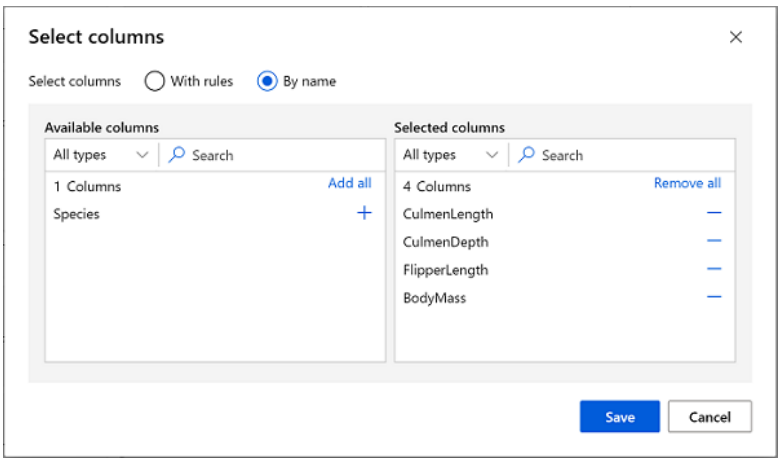


2. To cluster the penguin observations, we're going to use only the measurements - we'll ignore the species column. So, search for a **Select Columns in Dataset** module and place it on the canvas, below the **penguin-data** module and connect the output at the bottom of the **penguin-data** module to the input at the top of the **Select Columns in Dataset** module, like this:



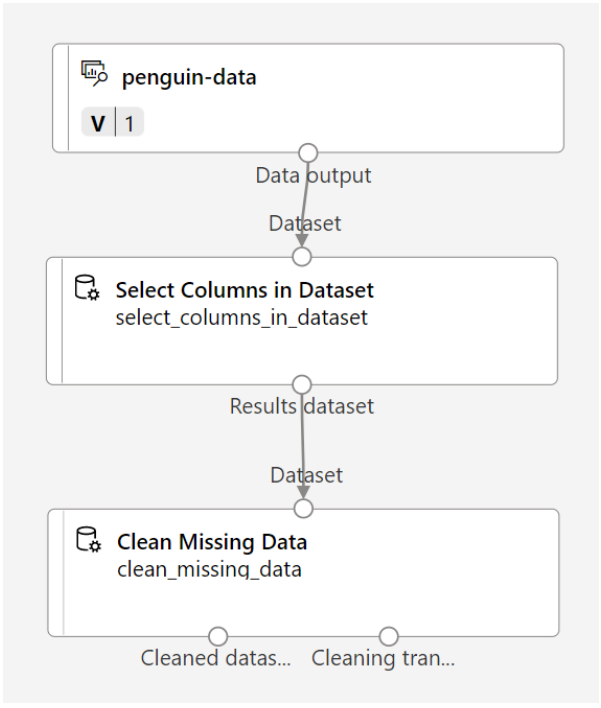
3. Double click on the **Select Columns in Dataset** module, and in the pane on the right, select **Edit column**. Then in the **Select columns** window, select **By name** and use the + links to select the column names **CulmenLength**, **CulmenDepth**, **FlipperLength**, and **BodyMass**; like this:

4. Close the **Select Columns in a Dataset** module settings to return to the designer canvas.



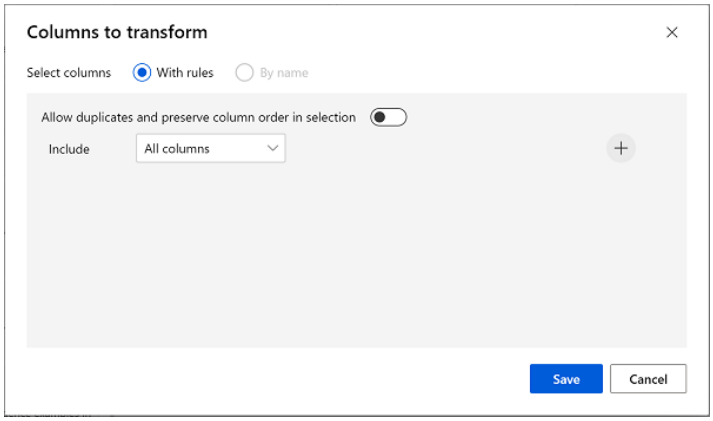
5. In the **Asset library**, search for a **Clean Missing Data** module and place it onto the canvas, below the **Select columns in a dataset** module and connect them like this:

6. Double click the **Clean Missing Data** module, and in the settings pane on the right, click **Edit column**. Then in the **Select columns** window, select **With rules** and include **All columns**; like this:

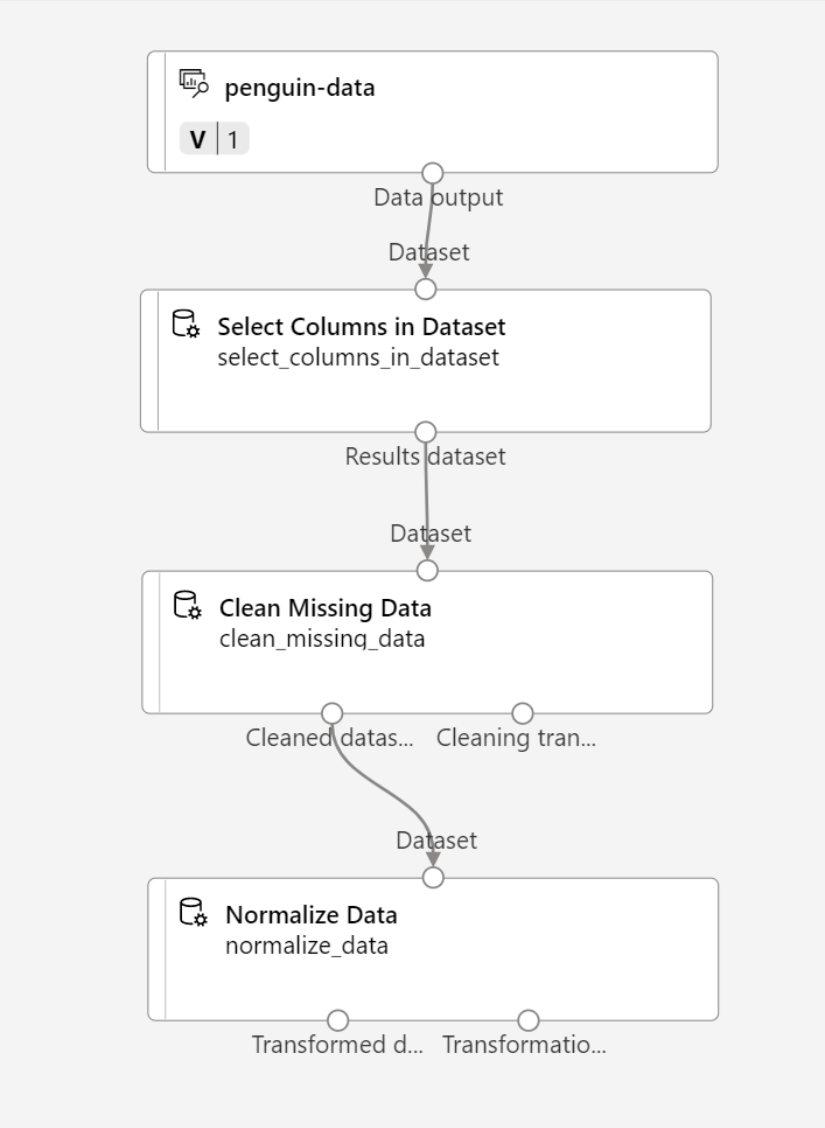


7. With the **Clean Missing Data** module still selected, in the settings pane, set the following configuration settings:

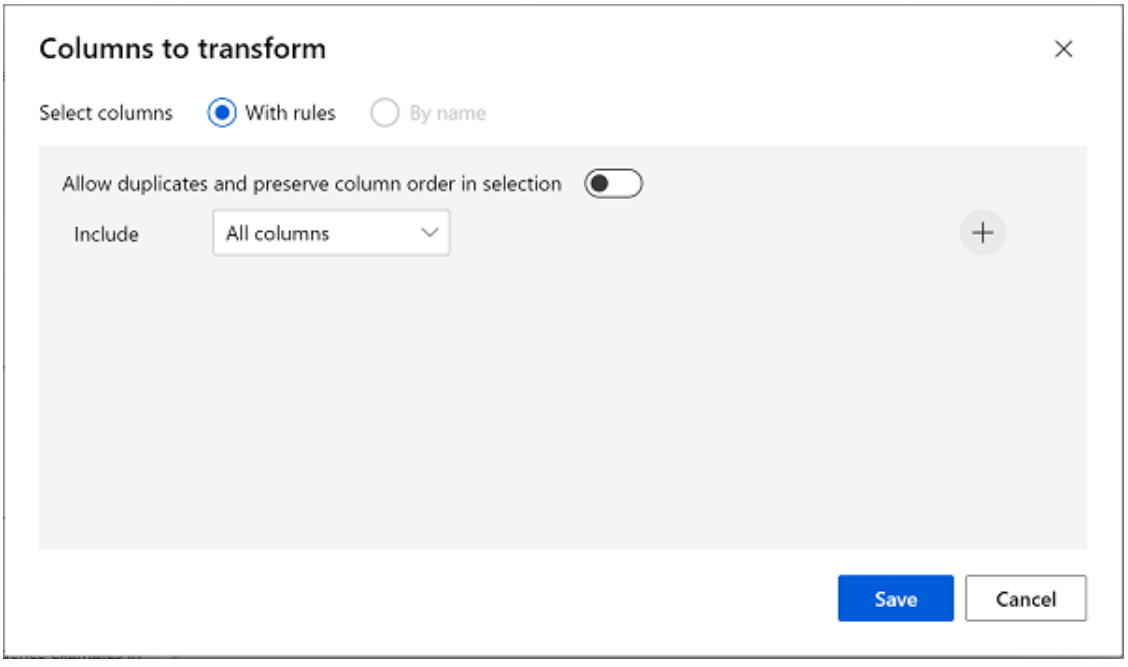
- **Minimum missing value ratio**: 0.0
- **Maximum missing value ratio**: 1.0
- **Cleaning mode**: Remove entire row



8. In the **Asset library**, search for a **Normalize Data** module and place it to the canvas, below the **Clean Missing Data** module. Then connect the left-most output from the **Clean Missing Data** module to the input of the **Normalize Data** module.



9. Double click the **Normalize Data** module, and in the pane on the right, set the **Transformation method** to **MinMax** and select **Edit column**. Then in the **Select columns** window, select **With rules** and include **All columns**; like this:

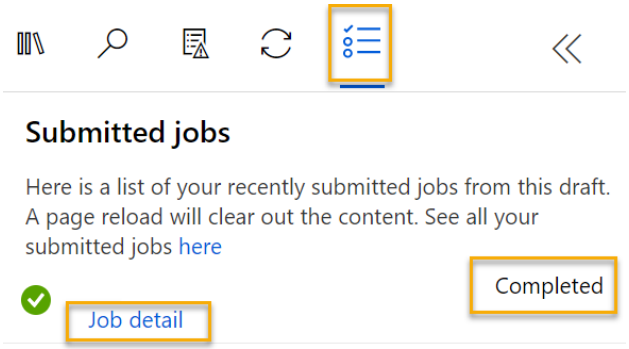


10. Close the **Normalize Data** module settings to return to the designer canvas.

Run the pipeline

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

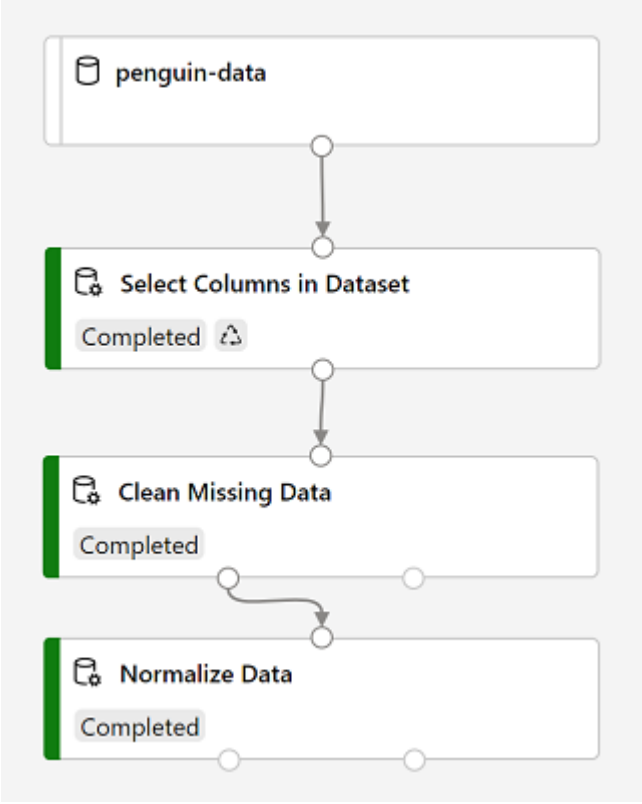
1. Select **Submit**, and run the pipeline as a **new experiment** named **mslearn-penguin-training** on your compute cluster.
2. Wait for the run to finish. This may take 5 minutes or more.



Notice that the left hand panel is now on the **Submitted Jobs** pane. You will know when the run is complete because the status of the job will change to **Complete**.

View the transformed data

1. When the run has completed, the dataset is now prepared for model training. Click on **Job Details**. You will be taken to another window which will show the modules like this:



2. In the new window, right click on the **Normalize Data** module, select **Preview data**, then select **Transformed dataset** to view the results.
3. View the data, noting that the **Species** column has been removed, there are no missing values, and the values for all four features have been normalized to a common scale.
4. Close the normalized data result visualization. Return to the previous pipeline window.

Now that you have selected and prepared the features you want to use from the dataset, you're ready to use them to train a clustering model.

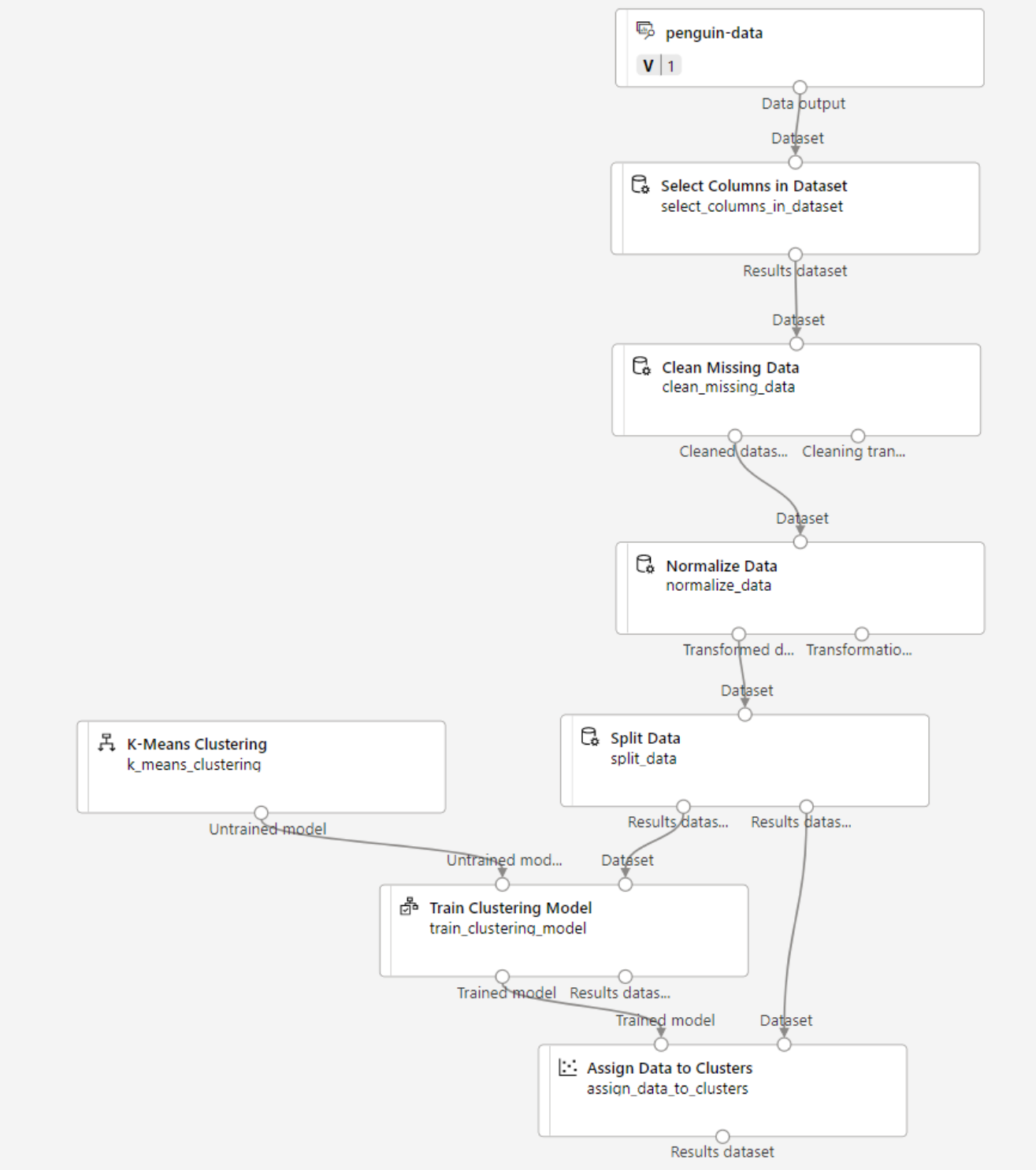
Create and run a training pipeline

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

Add training modules

To train a clustering model, you need to apply a clustering algorithm to the data, using only the features that you have selected for clustering. You'll train the model with a subset of the data, and use the rest to test the trained model.

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



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

1. Open the **Train Penguin Clustering** pipeline, if it's not already open.

2. In the **Asset Library** pane on the left, search for and place a **Split Data** module onto the canvas under the **Normalize Data** module. Then connect the left output of the **Normalize Data** module to the input of the **Split Data** module.

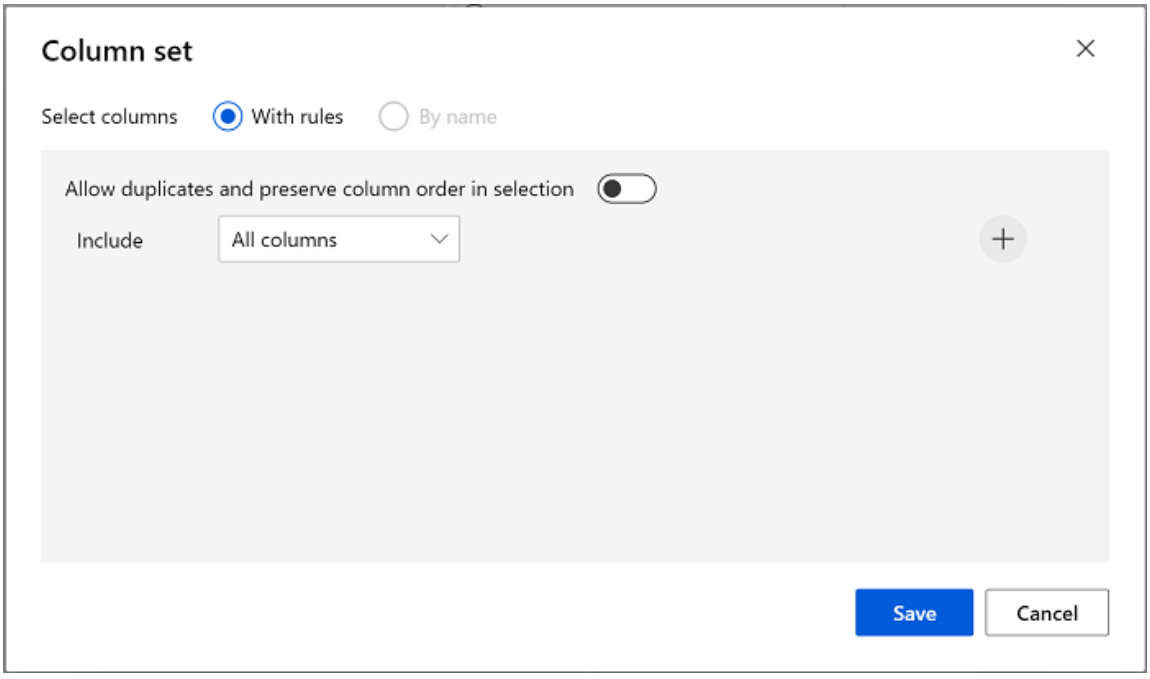
   Tip

   Use the search bar to quickly locate modules.

3. Select the **Split Data** module, and configure its settings as follows:
   - **Splitting mode**: Split Rows
   - **Fraction of rows in the first output dataset**: 0.7
   - **Randomized split**: True
   - **Random seed**: 123
   - **Stratified split**: False

4. In the **Asset library**, search for and place a **Train Clustering Model** module to the canvas, under the **Split Data** module. Then connect the *Result dataset1* (left) output of the **Split Data** module to the *Dataset* (right) input of the **Train Clustering Model** module.

5. The clustering model should assign clusters to the data items by using all of the features you selected from the original dataset. Double click the **Train Clustering Model** module and in the right hand pane, select **Edit Columns**. Use the **With rules** option to include all columns; like this:



6. The model we're training will use the features to group the data into clusters, so we need to train the model using a *clustering* algorithm. In the **Asset library**, search for and place a **K-Means Clustering** module to the canvas, to the left of the **penguin-data** dataset and above the **Train Clustering Model** module. Then connect its output to the **Untrained model** (left) input of the **Train Clustering Model** module.

7. The *K-Means* algorithm groups items into the number of clusters you specify - a value referred to as ***K***. Select the **K-Means Clustering** module and in the right hand pane, set the **Number of centroids** parameter to **3**.

   Note

   You can think of data observations, like the penguin measurements, as being multidimensional vectors. The K-Means algorithm works by:

a. initializing *K* coordinates as randomly selected points called *centroids* in *n*-dimensional space (where *n* is the number of dimensions in the feature vectors).

b. Plotting the feature vectors as points in the same space, and assigning each point to its closest centroid.

c. Moving the centroids to the middle of the points allocated to it (based on the *mean* distance).

d. Reassigning the points to their closest centroid after the move.

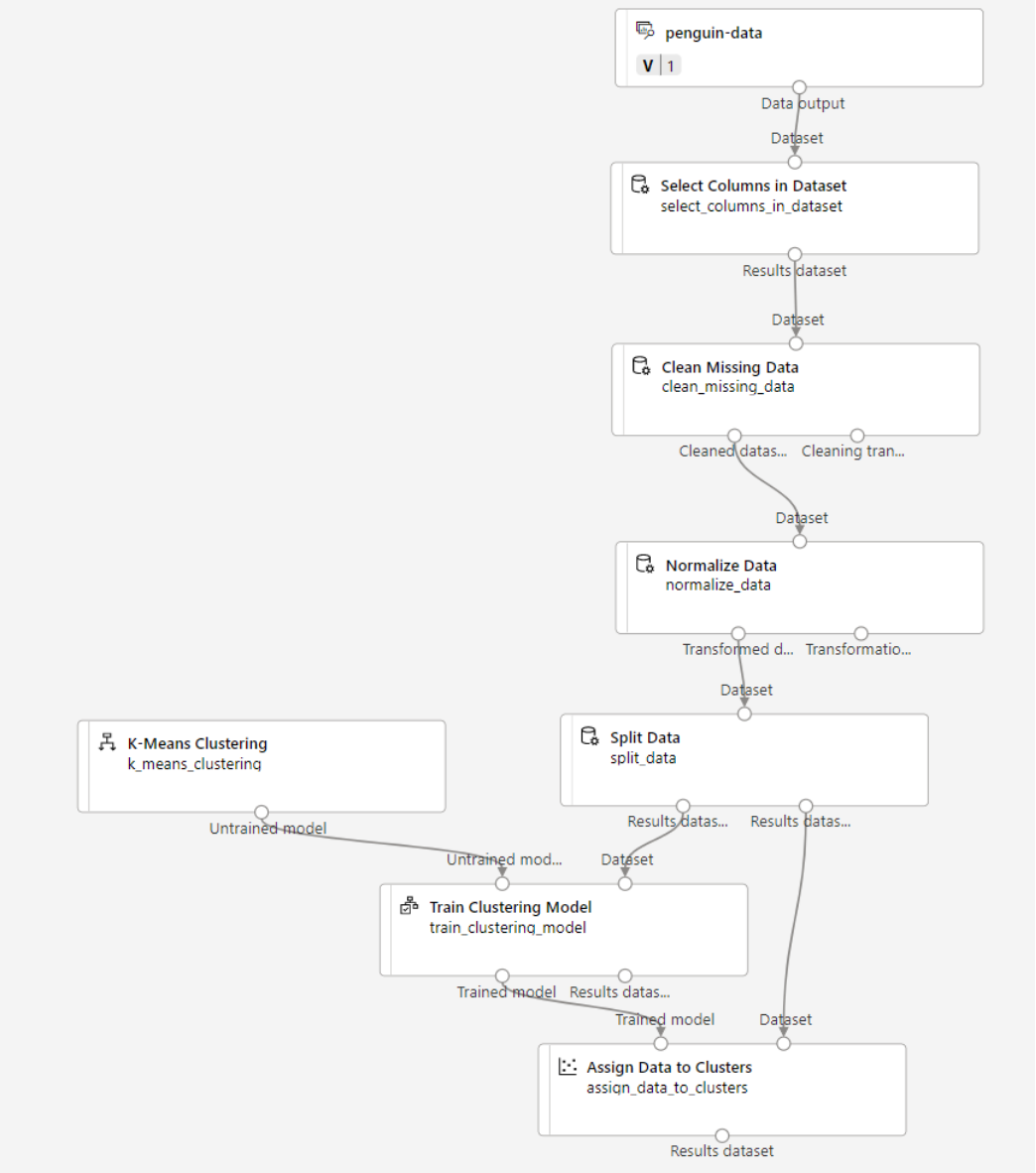
e. Repeating steps 3 and 4 until the cluster allocations stabilize or the specified number of iterations has completed.

8. After using 70% of the data to train the clustering model, you can use the remaining 30% to test it by using the model to assign the data to clusters. In the **Asset library**, search for and place an **Assign Data to Clusters** module to the canvas, below the **Train Clustering Model** module. Then connect the **Trained model** (left) output of the **Train Clustering Model** module to the **Trained model** (left) input of the **Assign Data to Clusters** module; and connect the **Results dataset2** (right) output of the **Split Data** module to the **Dataset** (right) input of the **Assign Data to Clusters** module.

Run the training pipeline

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

1. Ensure your pipeline looks like this:



2. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-penguin-training** on your compute cluster.

3. Wait for the experiment run to finish. This may take 5 minutes or more.

4. When the experiment run has finished, select **Job details**. In the new window, right click on the **Assign Data to Clusters** module, select **Preview data**, then select **Results dataset** to view the results.

5. Scroll down, and note the **Assignments** column, which contains the cluster (0, 1, or 2) to which each row is assigned. There are also new columns indicating the distance from the point representing this row to the centers of each of the clusters - the cluster to which the point is closest is the one to which it is assigned.
6. Close the **Assign Data to Clusters** visualization. Return to the pipeline window.

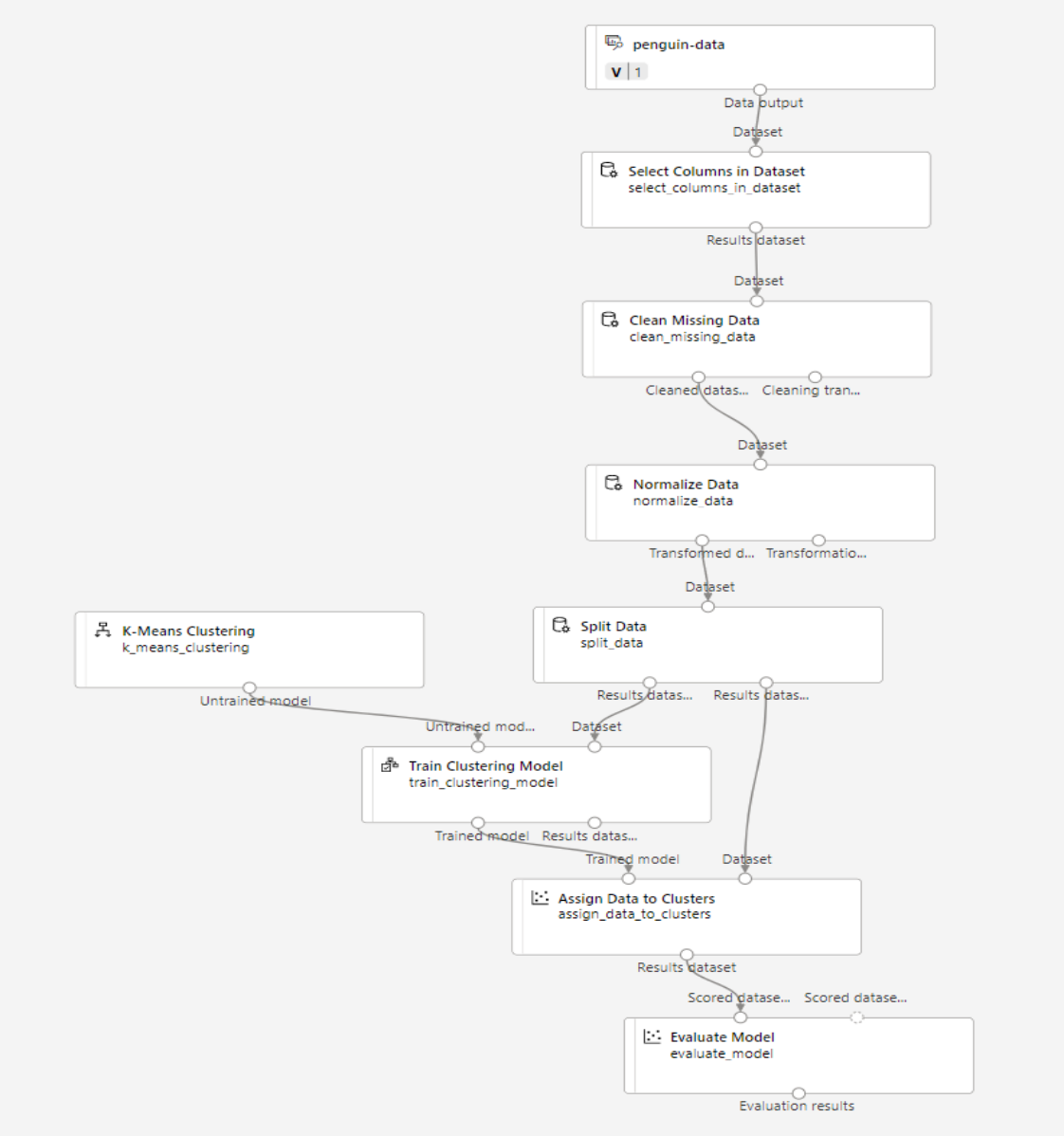
The model is predicting clusters for the penguin observations, but how reliable are its predictions? To assess that, you need to evaluate the model.

Evaluate a clustering model

Evaluating a clustering model is made difficult by the fact that there are no previously known *true* values for the cluster assignments. A successful clustering model is one that achieves a good level of separation between the items in each cluster, so we need metrics to help us measure that separation.

Add an Evaluate Model module

1. Open the **Train Penguin Clustering** pipeline you created in the previous unit if it's not already open.
2. In the **Asset library**, search for and place an **Evaluate Model** module on the canvas, under the **Assign Data to Clusters** module. Connect the output of the **Assign Data to Clusters** module to the **Scored dataset** (left) input of the **Evaluate Model** module.
3. Ensure your pipeline looks like this:



4. Select **Submit**, and run the pipeline using the existing **mslearn-penguin-training** experiment.
5. Wait for the experiment run to finish.
6. When the experiment run has finished, select **Job details**. Right click on the **Evaluate Model** module and select **Preview data**, then select **Evaluation results**. These metrics can help data scientists assess how well the model separates the clusters. They include a row of metrics for each cluster, and a summary row for a combined evaluation. The metrics in each row are:
    - **Average Distance to Other Center**: This indicates how close, on average, each point in the cluster is to the centroids of all other clusters.
    - **Average Distance to Cluster Center**: This indicates how close, on average, each point in the cluster is to the centroid of the cluster.
    - **Number of Points**: The number of points assigned to the cluster.
    - **Maximal Distance to Cluster Center**: The maximum of the distances between each point and the centroid of that point's cluster. If this number is high, the cluster may be widely dispersed. This statistic in combination with the **Average Distance to Cluster Center** helps you determine the cluster's *spread*.
7. Close the **Evaluate Model result visualization** window.

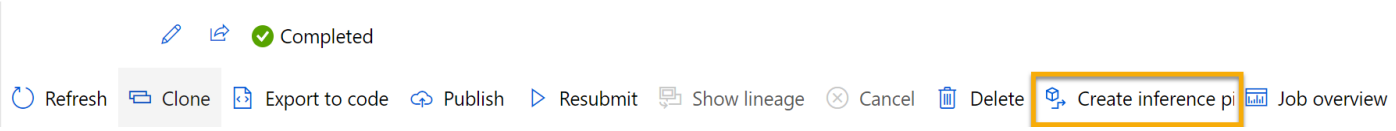
Now that you have a working clustering model, you can use it to assign new data to clusters in an *inference pipeline*.

Create an inference pipeline

After creating and running a pipeline to train the clustering model, you can create an *inference pipeline*. The inference pipeline uses the model to assign new data observations to clusters. This model will form the basis for a predictive service that you can publish for applications to use.

Create an inference pipeline

1. In Azure Machine Learning studio, expand the left-hand pane by selecting the three lines at the top left of the screen. Click on **Jobs** (under **Assets**) to view all of the jobs you have run. Select the experiment **mslearn-penguin-training**, then select the **mslearn-penguin-training** pipeline.
2. Locate the menu above the canvas and click on **Create inference pipeline**. You may need to expand your screen to full and click on the three dots icon **...** on the top right hand corner of the screen in order to find **Create inference pipeline** in the menu.

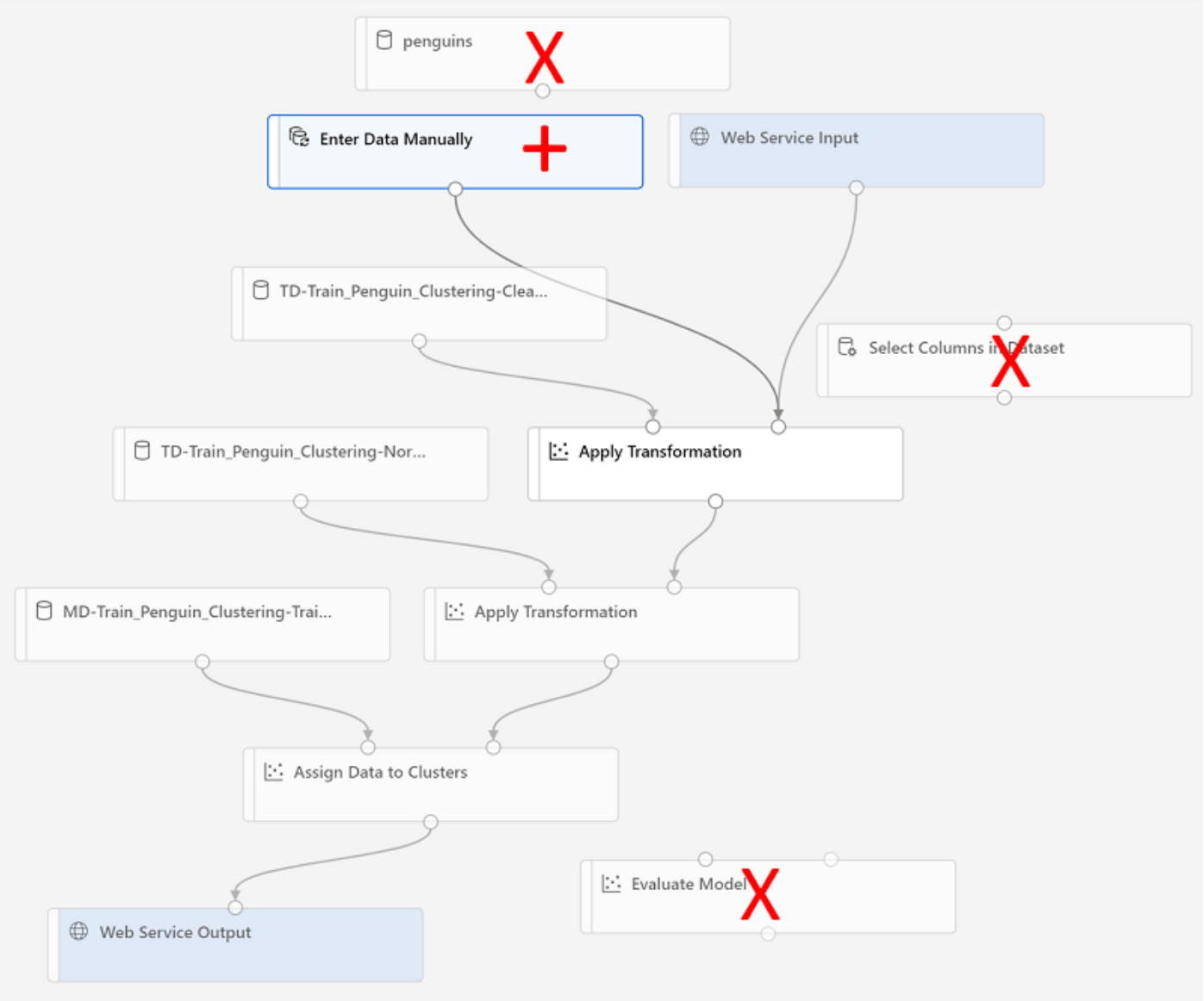


3. In the **Create inference pipeline** drop-down list, click **Real-time inference pipeline**. After a few seconds, a new version of your pipeline named **Train Penguin Clustering-real time inference** will be opened.

    *If the pipeline doesn't include **Web Service Input** and **Web Service Output** modules, go back to the **Designer** page and then reopen the **Train Penguin Clustering-real time inference** pipeline.*

4. Navigate to **Settings** on the upper right hand menu. Under **Draft details**, rename the new pipeline to **Predict Penguin Clusters**, and then review the new pipeline. It contains a web service input for new data to be submitted, and a web service output to return results. The transformations and clustering model in your training pipeline are a part of this pipeline. The trained model will be used to score the new data.

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



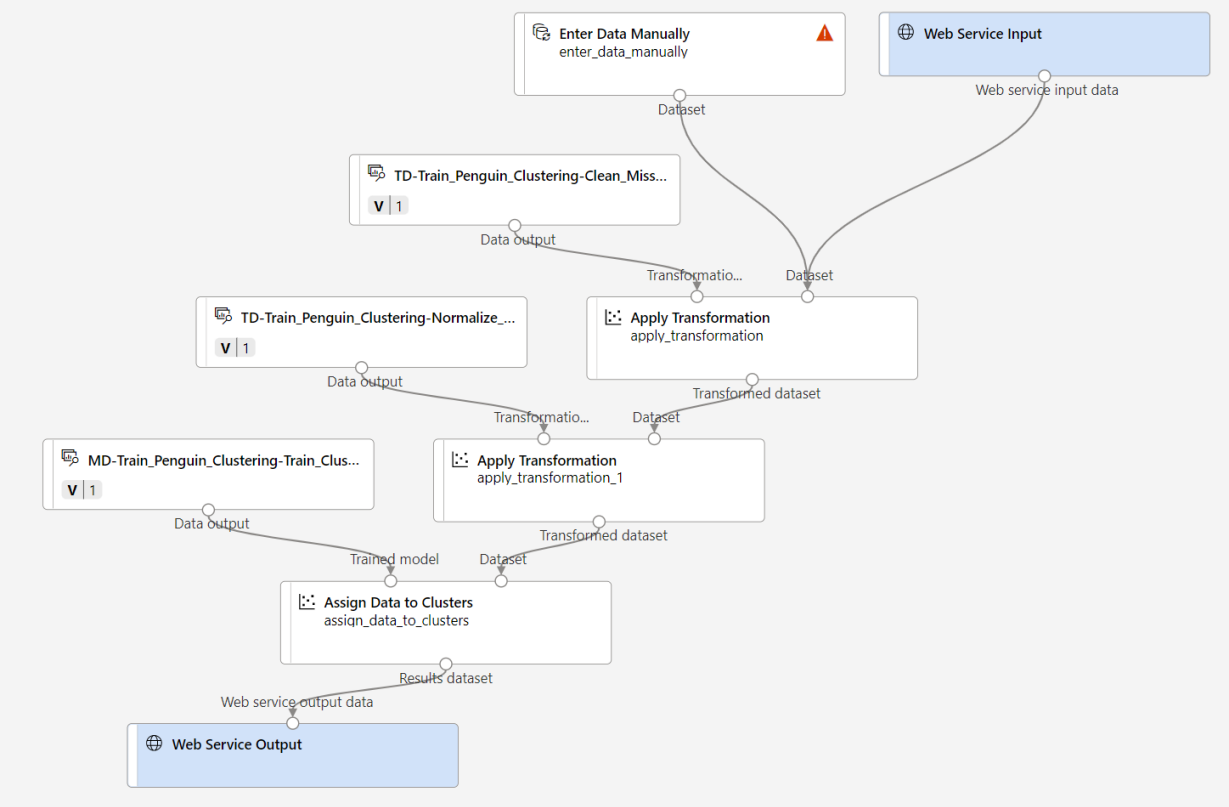
- Replace the **penguin-data** dataset with an **Enter Data Manually** module that doesn't include the **Species** column.
- Remove the **Select Columns in Dataset** module, which is now redundant.
- Connect the **Web Service Input** and **Enter Data Manually** modules (which represent inputs for data to be clustered) to the first **Apply Transformation** module.
- Remove the **Evaluate Model** module.

Follow the remaining steps below, using the image and information above for reference as you modify the pipeline.

5. The inference pipeline assumes that new data will match the schema of the original training data, so the **penguin-data** dataset from the training pipeline is included. However, this input data includes a column for the penguin species, which the model does not use. Delete both the **penguin-data** dataset and the **Select Columns in Dataset** modules, and replace them with an **Enter Data Manually** module from the **Asset library**. Then modify the settings of the **Enter Data Manually** module to use the following CSV input, which contains feature values for three new penguin observations (including headers):

```
CSVCopy
CulmenLength,CulmenDepth,FlipperLength,BodyMass
39.1,18.7,181,3750
49.1,14.8,220,5150
46.6,17.8,193,3800
```

6. Connect the outputs from both the **Web Service Input** and **Enter Data Manually** modules to the Dataset (right) input of the first **Apply Transformation** module.
7. Delete the **Evaluate Model** module.
8. Verify that your pipeline looks similar to the following image:



9. Submit the pipeline as a new experiment named **mslearn-penguin-inference** on your compute cluster. The experiment may take a while to run.
10. When the pipeline has finished, select **Job details**. In the new window, right click on **Assign Data to Clusters** module and select **Preview data** to see the predicted cluster assignments and metrics for the three penguin observations in the input data.

Your inference pipeline assigns penguin observations to clusters based on their features. Now you're ready to publish the pipeline so that client applications can use it.

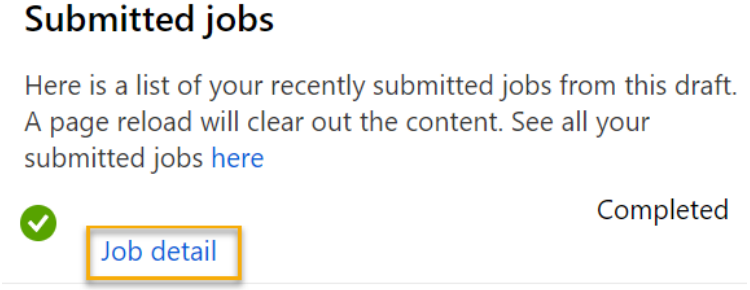
Deploy a predictive service

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

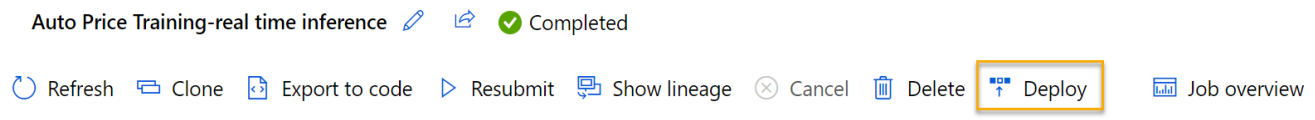
Note: In this exercise, you'll deploy the web service to to an Azure Container Instance (ACI). This type of compute is created dynamically, and is useful for development and testing. For production, you should create an *inference cluster*, which provide an Azure Kubernetes Service (AKS) cluster that provides better scalability and security.

Deploy a service

1. View the **Predict Penguin Clusters** inference pipeline you created in the previous unit.
2. Select **Job detail** on the left hand pane. This will open up another window.



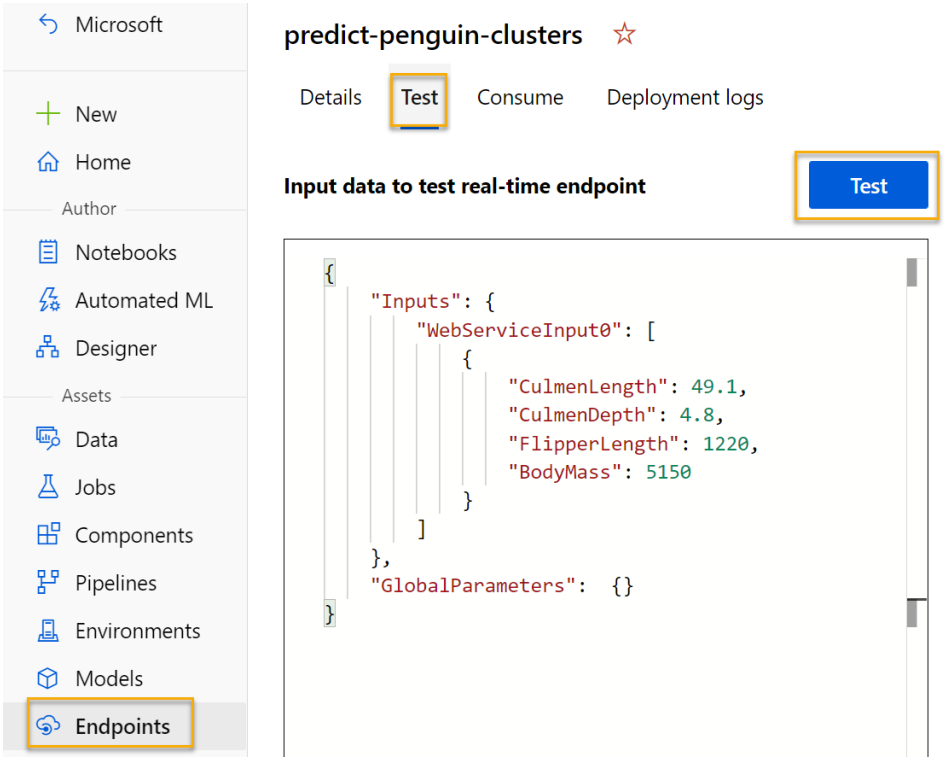
3. In the new window, select **Deploy**.



4. At the top right, select **Deploy**, and deploy a new real-time endpoint, using the following settings:
    - **Name**: predict-penguin-clusters
    - **Description**: Cluster penguins.
    - **Compute type**: Azure Container Instance
5. Wait for the web service to be deployed - this can take several minutes. The deployment status is shown at the top left of the designer interface.

Test the service

1. On the **Endpoints** page, open the **predict-penguin-clusters** real-time endpoint.



2. When the **predict-penguin-clusters** endpoint opens, select the **Test** tab. We will use it to test our model with new data. Delete the current data under **Input data to test real-time endpoint**. Copy and paste the below data into the data section:

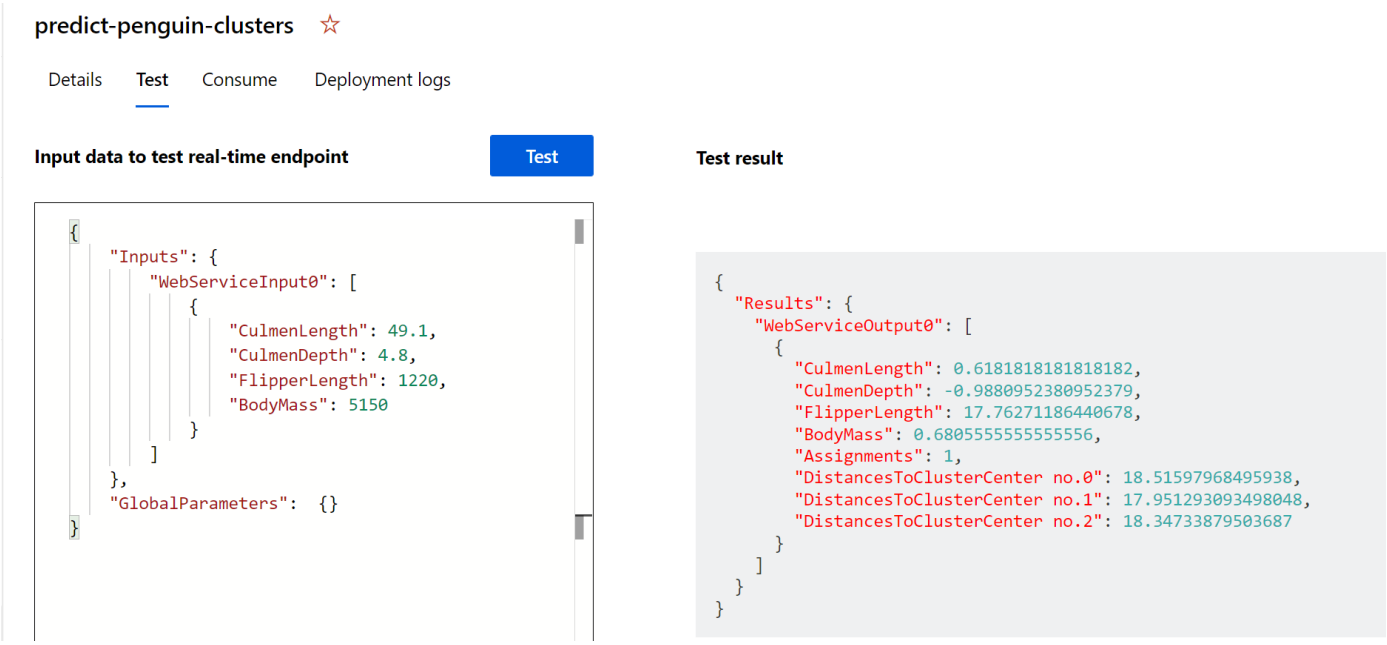
JSONCopy

```
{
  "Inputs": {
    "WebServiceInput0": [
      {
        "CulmenLength": 49.1,
        "CulmenDepth": 4.8,
        "FlipperLength": 1220,
        "BodyMass": 5150
      }
    ]
  },
  "GlobalParameters": {}
}
```

Note

The JSON above defines features for a penguin, and uses the **predict-penguin-clusters** service you created to predict a cluster assignment.

3. Select **Test**. On the right hand of the screen, you should see the output '**assignments**'. Notice how the assigned cluster is the one with the shortest distance to cluster center.



You have just tested a service that is ready to be connected to a client application using the credentials in the **Consume** tab. We will end the lab here. You are welcome to continue to experiment with the service you just deployed.

Knowledge check

1. You are using an Azure Machine Learning designer pipeline to train and test a K-Means clustering model. You want your model to assign items to one of three clusters. Which configuration property of the K-Means Clustering module should you set to accomplish this?

- Set Number of Centroids to 3
- Set Random number seed to 3
- Set Iterations to 3

2. You use Azure Machine Learning designer to create a training pipeline for a clustering model. Now you want to use the model in an inference pipeline. Which module should you use to infer cluster predictions from the model?

- Score Model
- Assign Data to Clusters
- Train Clustering Model

Part 3/6

Microsoft Azure AI Fundamentals: Explore computer vision

Computer vision is an area of artificial intelligence (AI) in which software systems are designed to perceive the world visually, though cameras, images, and video. There are multiple specific types of computer vision problem that AI engineers and data scientists can solve using a mix of custom machine learning models and platform-as-a-service (PaaS) solutions - including many cognitive services in Microsoft Azure.

Part 3/6_1/6

Analyze images with the Computer Vision service

The Computer Vision service enables software engineers to create intelligent solutions that extract information from images; a common task in many artificial intelligence (AI) scenarios.

Introduction

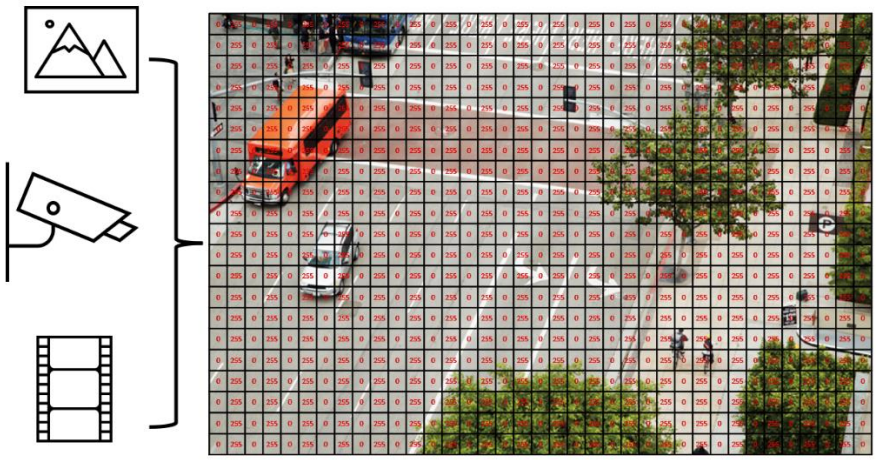
Computer vision is one of the core areas of artificial intelligence (AI), and focuses on creating solutions that enable AI applications to "see" the world and make sense of it.

Of course, computers don't have biological eyes that work the way ours do, but they are capable of processing images; either from a live camera feed or from digital photographs or videos. This ability to process images is the key to creating software that can emulate human visual perception.

Some potential uses for computer vision include:

- **Content Organization**: Identify people or objects in photos and organize them based on that identification. Photo recognition applications like this are commonly used in photo storage and social media applications.
- **Text Extraction**: Analyze images and PDF documents that contain text and extract the text into a structured format.
- **Spatial Analysis**: Identify people or objects, such as cars, in a space and map their movement within that space.

To an AI application, an image is just an array of pixel values. These numeric values can be used as *features* to train machine learning models that make predictions about the image and its contents.



Training machine learning models from scratch can be very time intensive and require a large amount of data. Microsoft's Computer Vision service gives you access to pre-trained computer vision capabilities.

Learning objectives

In this module you will:

- Identify image analysis tasks that can be performed with the Computer Vision service.
- Provision a Computer Vision resource.
- Use a Computer Vision resource to analyze an image.

Get started with image analysis on Azure

The Computer Vision service is a cognitive service in Microsoft Azure that provides pre-built computer vision capabilities. The service can analyze images, and return detailed information about an image and the objects it depicts.

Azure resources for Computer Vision

To use the Computer Vision service, you need to create a resource for it in your Azure subscription. You can use either of the following resource types:

- **Computer Vision**: A specific resource for the Computer Vision service. Use this resource type if you don't intend to use any other cognitive services, or if you want to track utilization and costs for your Computer Vision resource separately.
- **Cognitive Services**: A general cognitive services resource that includes Computer Vision along with many other cognitive services; such as Text Analytics, Translator Text, and others. Use this resource type if you plan to use multiple cognitive services and want to simplify administration and development.

Whichever type of resource you choose to create, it will provide two pieces of information that you will need to use it:

- A **key** that is used to authenticate client applications.
- An **endpoint** that provides the HTTP address at which your resource can be accessed.

Note

If you create a Cognitive Services resource, client applications use the same key and endpoint regardless of the specific service they are using.

Analyzing images with the Computer Vision service

After you've created a suitable resource in your subscription, you can submit images to the Computer Vision service to perform a wide range of analytical tasks.

Describing an image

Computer Vision has the ability to analyze an image, evaluate the objects that are detected, and generate a human-readable phrase or sentence that can describe what was detected in the image. Depending on the image contents, the service may return multiple results, or phrases. Each returned phrase will have an associated confidence score, indicating how confident the algorithm is in the supplied description. The highest confidence phrases will be listed first.

To help you understand this concept, consider the following image of the Empire State building in New York. The returned phrases are listed below the image in the order of confidence.

- A black and white photo of a city
- A black and white photo of a large city
- A large white building in a city

Tagging visual features

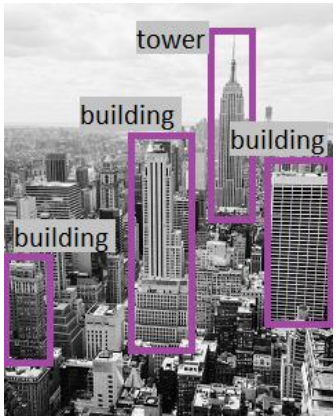
The image descriptions generated by Computer Vision are based on a set of thousands of recognizable objects, which can be used to suggest *tags* for the image. These tags can be associated with the image as metadata that summarizes attributes of the image; and can be particularly useful if you want to index an image along with a set of key terms that might be used to search for images with specific attributes or contents.

For example, the tags returned for the Empire State building image include:

- skyscraper
- tower
- building

Detecting objects

The object detection capability is similar to tagging, in that the service can identify common objects; but rather than tagging, or providing tags for the recognized objects only, this service can also return what is known as bounding box coordinates. Not only will you get the type of object, but you will also receive a set of coordinates that indicate the top, left, width, and height of the object detected, which you can use to identify the location of the object in the image, like this:



Detecting brands

This feature provides the ability to identify commercial brands. The service has an existing database of thousands of globally recognized logos from commercial brands of products.

When you call the service and pass it an image, it performs a detection task and determine if any of the identified objects in the image are recognized brands. The service compares the brands against its database of popular brands spanning clothing, consumer electronics, and many more categories. If a known brand is detected, the service returns a response that contains the brand name, a confidence score (from 0 to 1 indicating how positive the identification is), and a bounding box (coordinates) for where in the image the detected brand was found.

For example, in the following image, a laptop has a Microsoft logo on its lid, which is identified and located by the Computer Vision service.

Detecting faces

The Computer Vision service can detect and analyze human faces in an image, including the ability to determine age and a bounding box rectangle for the location of the face(s). The facial analysis capabilities of the Computer Vision service are a subset of those provided by the dedicated Face Service. If you need basic face detection and analysis, combined with general image analysis capabilities, you can use the Computer Vision service; but for more comprehensive facial analysis and facial recognition functionality, use the Face service.

The following example shows an image of a person with their face detected and approximate age estimated.

Categorizing an image

Computer Vision can categorize images based on their contents. The service uses a parent/child hierarchy with a "current" limited set of categories. When analyzing an image, detected objects are compared to the existing categories to determine the best way to provide the categorization. As an example, one of the parent categories is **people_**. This image of a person on a roof is assigned a category of **people_**.

A slightly different categorization is returned for the following image, which is assigned to the category **people_group** because there are multiple people in the image:

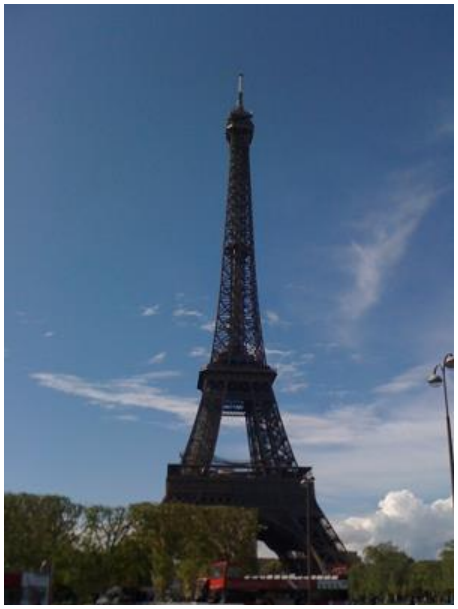
Review the 86-category list here.

Detecting domain-specific content

When categorizing an image, the Computer Vision service supports two specialized domain models:

- **Celebrities** - The service includes a model that has been trained to identify thousands of well-known celebrities from the worlds of sports, entertainment, and business.
- **Landmarks** - The service can identify famous landmarks, such as the Taj Mahal and the Statue of Liberty.

For example, when analyzing the following image for landmarks, the Computer Vision service identifies the Eiffel Tower, with a confidence of 99.41%.



Optical character recognition

The Computer Vision service can use optical character recognition (OCR) capabilities to detect printed and handwritten text in images. This capability is explored in the Read text with the Computer Vision service module on Microsoft Learn.

Additional capabilities

In addition to these capabilities, the Computer Vision service can:

- Detect image types - for example, identifying clip art images or line drawings.
- Detect image color schemes - specifically, identifying the dominant foreground, background, and overall colors in an image.
- Generate thumbnails - creating small versions of images.
- Moderate content - detecting images that contain adult content or depict violent, gory scenes.

Exercise - Analyze images with the Computer Vision service

The *Computer Vision* cognitive service uses pre-trained machine learning models to analyze images and extract information about them.

For example, suppose the fictitious retailer *Northwind Traders* has decided to implement a "smart store", in which AI services monitor the store to identify customers requiring assistance, and direct employees to help them. By using the Computer Vision service, images taken by cameras throughout the store can be analyzed to provide meaningful descriptions of what they depict.

In this lab, you'll use a simple command-line application to see the Computer Vision service in action. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Computer Vision service by creating either a **Computer Vision** resource or a **Cognitive Services** resource.

If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

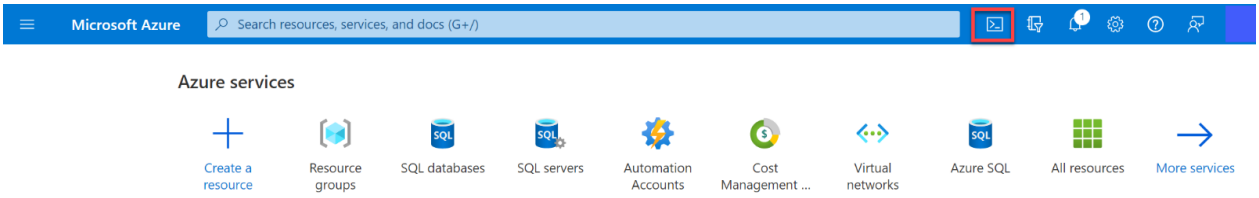
1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: *Choose any available region*:
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.

3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

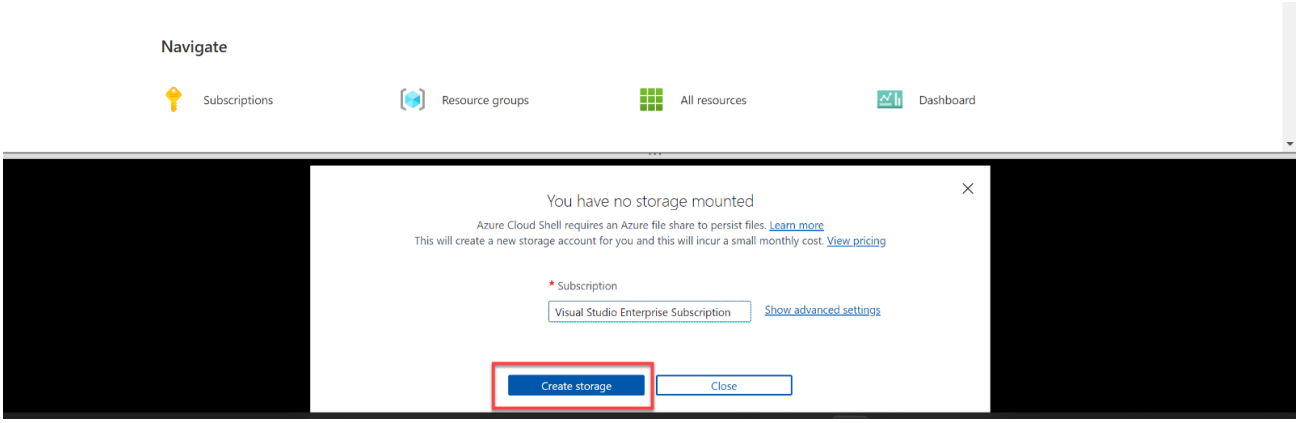
Run Cloud Shell

To test the capabilities of the Computer Vision service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

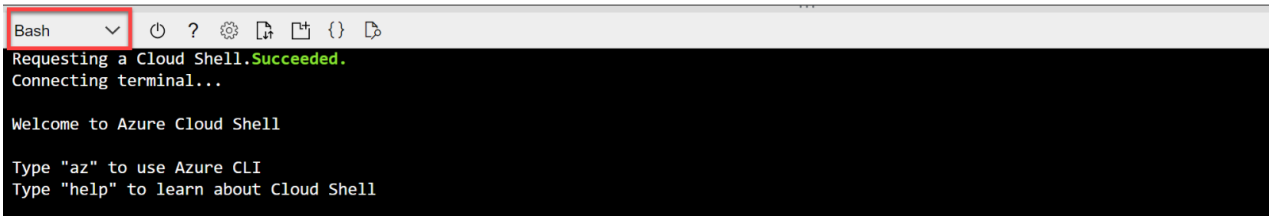
1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.



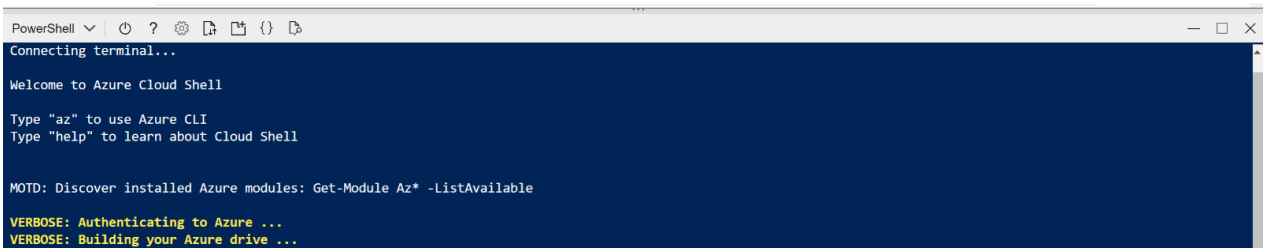
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.



4. Make sure the the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.



5. Wait for PowerShell to start. You should see the following screen in the Azure portal:



Configure and run a client application

Now that you have a Cloud Shell environment, you can run a simple application that uses the Computer Vision service to analyze an image.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

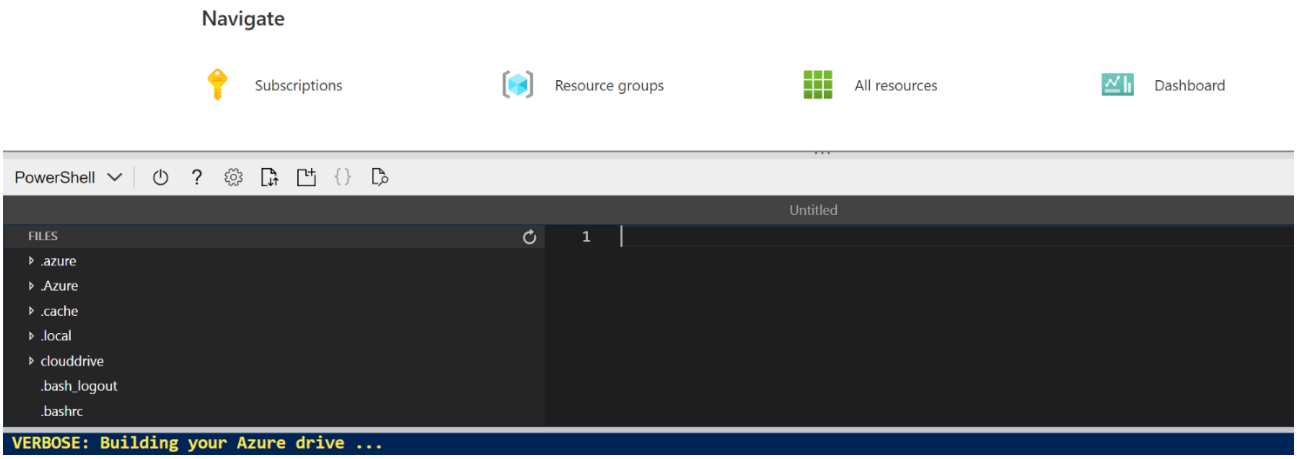
   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

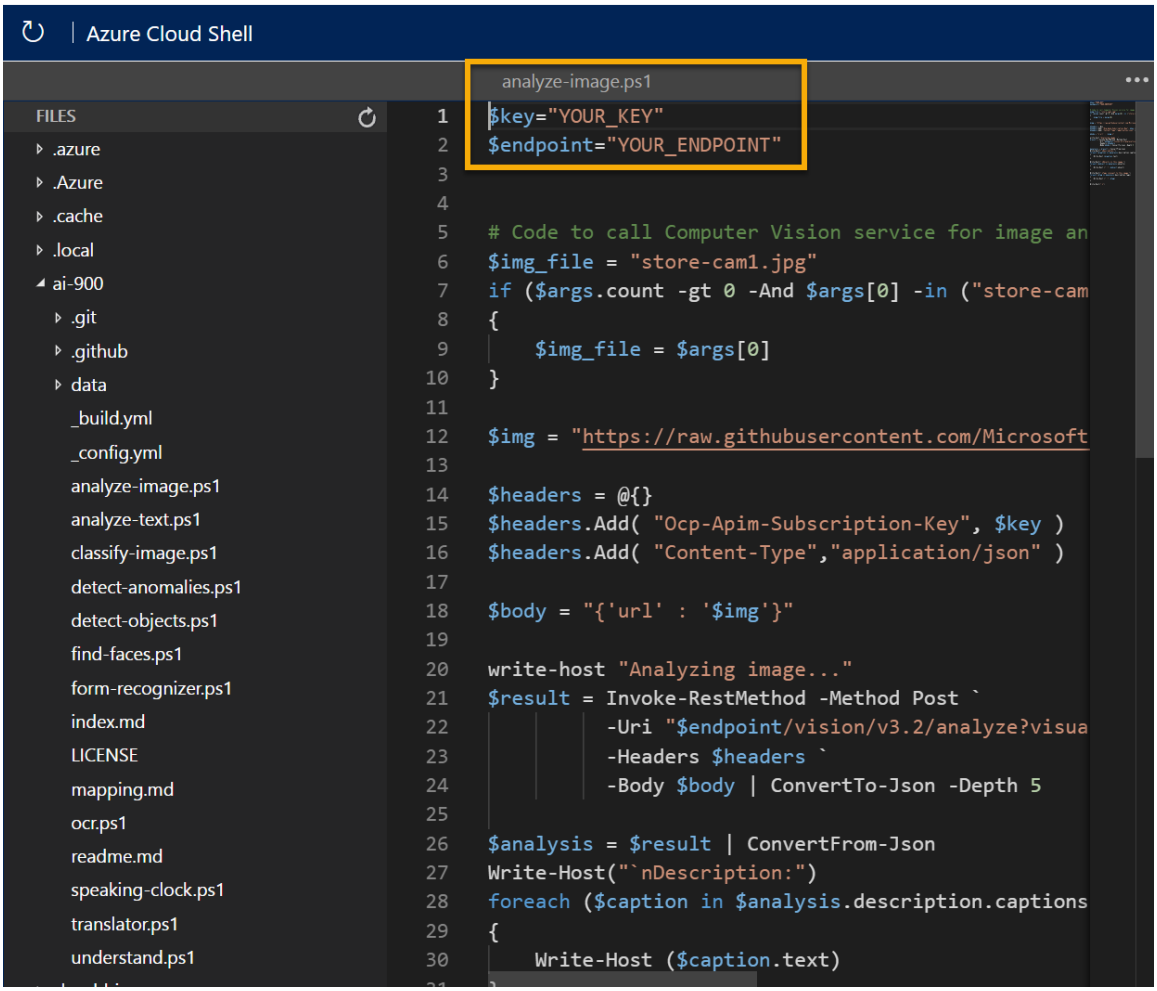
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **analyze-image.ps1**. This file contains some code that uses the Computer Vision service to analyze an image, as shown here:



4. Don't worry too much about the code, the important thing is that it needs the endpoint URL and either of the keys for your Cognitive Services resource. Copy these from the **Keys and Endpoints** page for your resource from the Azure portal and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_ENDPOINT** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

   After pasting the key and endpoint values, the first two lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $endpoint="https..."
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes.

   The sample client application will use your Computer Vision service to analyze the following image, taken by a camera in the Northwind Traders store:

6. In the PowerShell pane, enter the following commands to run the code:

   Copy

   ```
   cd ai-900
   ./analyze-image.ps1 store-camera-1.jpg
   ```

7. Review the results of the image analysis, which include:
   - A suggested caption that describes the image.
   - A list of objects identified in the image.
   - A list of "tags" that are relevant to the image.
8. Now let's try another image:

   To analyze the second image, enter the following command:

   Copy

   ```
   ./analyze-image.ps1 store-camera-2.jpg
   ```

9. Review the results of the image analysis for the second image.
10. Let's try one more:

    To analyze the third image, enter the following command:

    Copy

    ```
    ./analyze-image.ps1 store-camera-3.jpg
    ```

11. Review the results of the image analysis for the third image.

Learn more

This simple app shows only some of the capabilities of the Computer Vision service. To learn more about what you can do with this service, see the Computer Vision page.

Knowledge check

1. You want to use the Computer Vision service to analyze images. You also want to use the Language service to analyze text. You want developers to require only one key and endpoint to access all of your services. What kind of resource should you create in your Azure subscription?

- Computer Vision
- Cognitive Services
- Custom Vision

2. You want to use the Computer Vision service to identify the location of individual items in an image. Which of the following features should you retrieve?

- Objects
- Tags
- Categories

3. You want to use the Computer Vision service to analyze images of locations and identify well-known buildings. What should you do?

- Retrieve the objects in the image.
- Retrieve the categories for the image, specifying the celebrities domain
- Retrieve the categories for the image, specifying the landmarks domain

Part 3/6_2/6

Classify images with the Custom Vision service

Image classification is a common workload in artificial intelligence (AI) applications. It harnesses the predictive power of machine learning to enable AI systems to identify real-world items based on images.

Introduction

Image classification is a common workload in artificial intelligence (AI) applications. It harnesses the predictive power of machine learning to enable AI systems to identify real-world items based on images.

Uses of image classification

Some potential uses for image classification include:

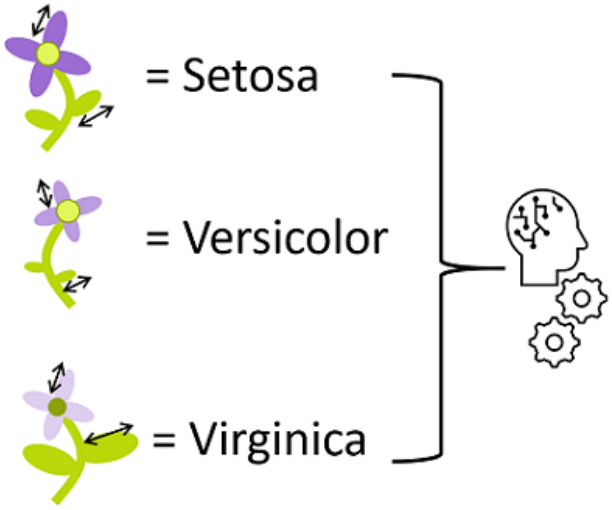
- **Product identification**: performing visual searches for specific products in online searches or even, in-store using a mobile device.
- **Disaster investigation**: identifying key infrastructure for major disaster preparation efforts. For example, identifying bridges and roads in aerial images can help disaster relief teams plan ahead in regions that are not well mapped.
- **Medical diagnosis:** evaluating images from X-ray or MRI devices could quickly classify specific issues found as cancerous tumors, or many other medical conditions related to medical imaging diagnosis.

Learning objectives

Learn how to use the Custom Vision service to create an image classification solution. In this module you will:

- Identify image classification scenarios and technologies
- Provision a Custom Vision resource and use the Custom Vision portal
- Train an image classification model
- Publish and consume an image classification model

Understand classification



You can use a machine learning *classification* technique to predict which category, or *class*, something belongs to. Classification machine learning models use a set of inputs, which we call *features*, to calculate a probability score for each possible class and predict a *label* that indicates the most likely class that an object belongs to.

For example, the features of a flower might include the measurements of its petals, stem, sepals, and other quantifiable characteristics. A machine learning model could be trained by applying an algorithm to these measurements that calculates the most likely species of the flower - its class.

Understand image classification

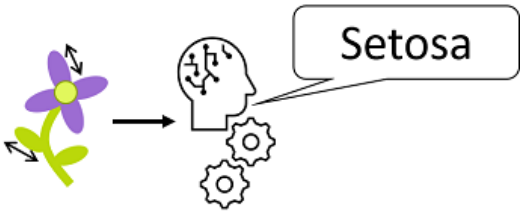
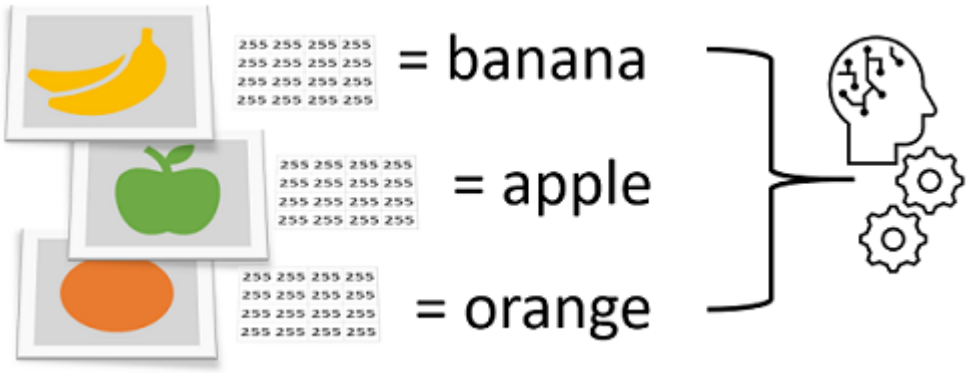


Image classification is a machine learning technique in which the object being classified is an image, such as a photograph.

To create an image classification model, you need data that consists of features and their labels. The existing data is a set of categorized images. Digital images are made up of an array of pixel values, and these are used as features to train the model based on the known image classes.



The model is trained to match the patterns in the pixel values to a set of class labels. After the model has been trained, you can use it with new sets of features to predict unknown label values.

Azure's Custom Vision service

Most modern image classification solutions are based on *deep learning* techniques that make use of *convolutional neural networks* (CNNs) to uncover patterns in the pixels that correspond to particular classes. Training an effective CNN is a complex task that requires considerable expertise in data science and machine learning.

Common techniques used to train image classification models have been encapsulated into the **Custom Vision** cognitive service in Microsoft Azure; making it easy to train a model and publish it as a software service with minimal knowledge of deep learning techniques. You can use the Custom Vision cognitive service to train image classification models and deploy them as services for applications to use.

Get started with image classification on Azure

You can perform image classification using the Custom Vision service, available as part of the Azure Cognitive Services offerings. This is generally easier and quicker than writing your own model training code, and enables people with little or no machine learning expertise to create an effective image classification solution.

Azure resources for Custom Vision

Creating an image classification solution with Custom Vision consists of two main tasks. First you must use existing images to train the model, and then you must publish the model so that client applications can use it to generate predictions.

For each of these tasks, you need a resource in your Azure subscription. You can use the following types of resource:

- **Custom Vision**: A dedicated resource for the custom vision service, which can be *training*, a *prediction*, or *both* resources.
- **Cognitive Services**: A general cognitive services resource that includes Custom Vision along with many other cognitive services. You can use this type of resource for *training*, *prediction*, or both.

The separation of training and prediction resources is useful when you want to track resource utilization for model training separately from client applications using the model to predict image classes. However, it can make development of an image classification solution a little confusing.

The simplest approach is to use a general Cognitive Services resource for both training and prediction. This means you only need to concern yourself with one *endpoint* (the HTTP address at which your service is hosted) and *key* (a secret value used by client applications to authenticate themselves).

If you choose to create a Custom Vision resource, you will be prompted to choose *training*, *prediction*, or *both* - and it's important to note that if you choose "both", then ***two*** resources are created - one for training and one for prediction.

It's also possible to take a mix-and-match approach in which you use a dedicated Custom Vision resource for training, but deploy your model to a Cognitive Services resource for prediction. For this to work, the training and prediction resources must be created in the same region.

Model training

To train a classification model, you must upload images to your training resource and label them with the appropriate class labels. Then, you must train the model and evaluate the training results.

You can perform these tasks in the *Custom Vision portal*, or if you have the necessary coding experience you can use one of the Custom Vision service programming language-specific software development kits (SDKs).

One of the key considerations when using images for classification, is to ensure that you have sufficient images of the objects in question and those images should be of the object from many different angles.

Model evaluation

Model training process is an iterative process in which the Custom Vision service repeatedly trains the model using some of the data, but holds some back to evaluate the model. At the end of the training process, the performance for the trained model is indicated by the following evaluation metrics:

- **Precision**: What percentage of the class predictions made by the model were correct? For example, if the model predicted that 10 images are oranges, of which eight were actually oranges, then the precision is 0.8 (80%).
- **Recall**: What percentage of class predictions did the model correctly identify? For example, if there are 10 images of apples, and the model found 7 of them, then the recall is 0.7 (70%).
- **Average Precision (AP)**: An overall metric that takes into account both precision and recall).

Using the model for prediction

After you've trained the model, and you're satisfied with its evaluated performance, you can publish the model to your prediction resource. When you publish the model, you can assign it a name (the default is "Iteration*X*", where X is the number of times you have trained the model).

To use your model, client application developers need the following information:

- **Project ID**: The unique ID of the Custom Vision project you created to train the model.
- **Model name**: The name you assigned to the model during publishing.
- **Prediction endpoint**: The HTTP address of the endpoints for the *prediction* resource to which you published the model (***not*** the training resource).
- **Prediction key**: The authentication key for the *prediction* resource to which you published the model (***not*** the training resource).

Exercise - Create an image classification solution

The *Computer Vision* cognitive service provides useful pre-built models for working with images, but you'll often need to train your own model for computer vision. For example, suppose the Northwind Traders retail company wants to create an automated checkout system that identifies the grocery items customers want to buy based on an image taken by a camera at the checkout. To do this, you'll need to train a classification model that can classify the images to identify the item being purchased.

In Azure, you can use the ***Custom Vision*** cognitive service to train an image classification model based on existing images. There are two elements to creating an image classification solution. First, you must train a model to recognize different classes using existing images. Then, when the model is trained you must publish it as a service that can be consumed by applications.

To test the capabilities of the Custom Vision service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Custom Vision service by creating either a **Custom Vision** resource or a **Cognitive Services** resource.

Note

Not every resource is available in every region. Whether you create a Custom Vision or Cognitive Services resource, only resources created in **certain regions** can be used to access Custom Vision services. For simplicity, a region is pre-selected for you in the configuration instructions below.

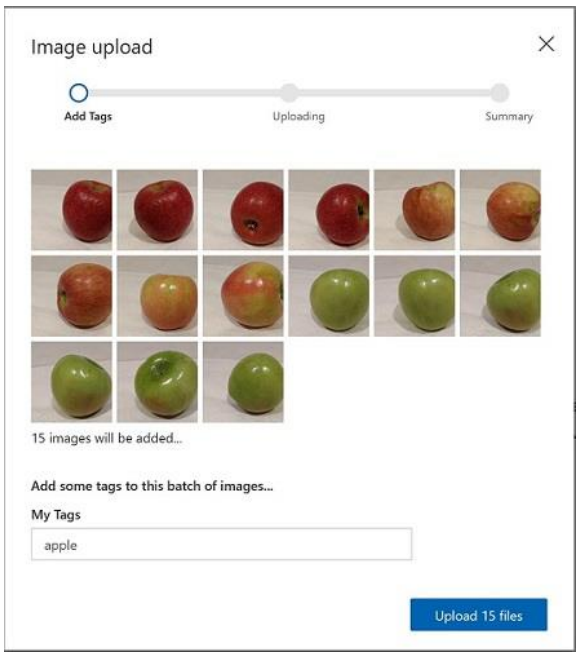
Create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: East US
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

Create a Custom Vision project

To train an object detection model, you need to create a Custom Vision project based on your training resource. To do this, you'll use the Custom Vision portal.

1. Download and extract the training images from https://aka.ms/fruit-images. These images are provided in a zipped folder, which when extracted contains subfolders called **apple**, **banana**, and **orange**.
2. In another browser tab, open the Custom Vision portal at https://customvision.ai. If prompted, sign in using the Microsoft account associated with your Azure subscription and agree to the terms of service.
3. In the Custom Vision portal, create a new project with the following settings:
    - **Name**: Grocery Checkout
    - **Description**: Image classification for groceries
    - **Resource**: *The Custom Vision resource you created previously*

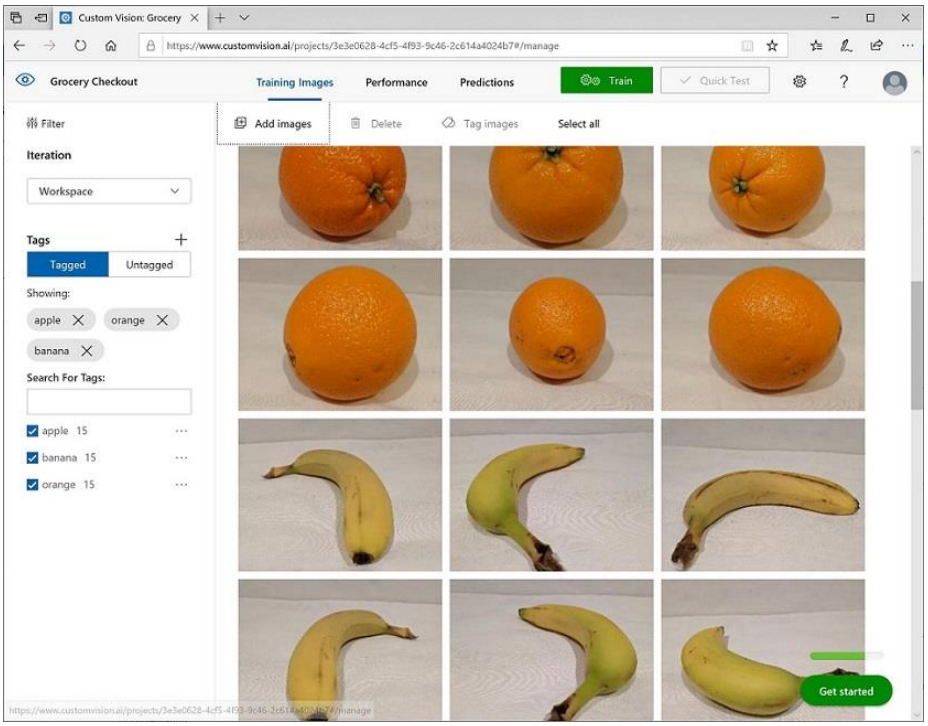


- **Project Types**: Classification
- **Classification Types**: Multiclass (single tag per image)
- **Domains**: Food

4. Click **[+] Add images**, and select all of the files in the **apple** folder you extracted previously. Then upload the image files, specifying the tag *apple*, like this:

5. Repeat the previous step to upload the images in the **banana** folder with the tag *banana*, and the images in the **orange** folder with the tag *orange*.

6. Explore the images you have uploaded in the Custom Vision project - there should be 15 images of each class, like this:

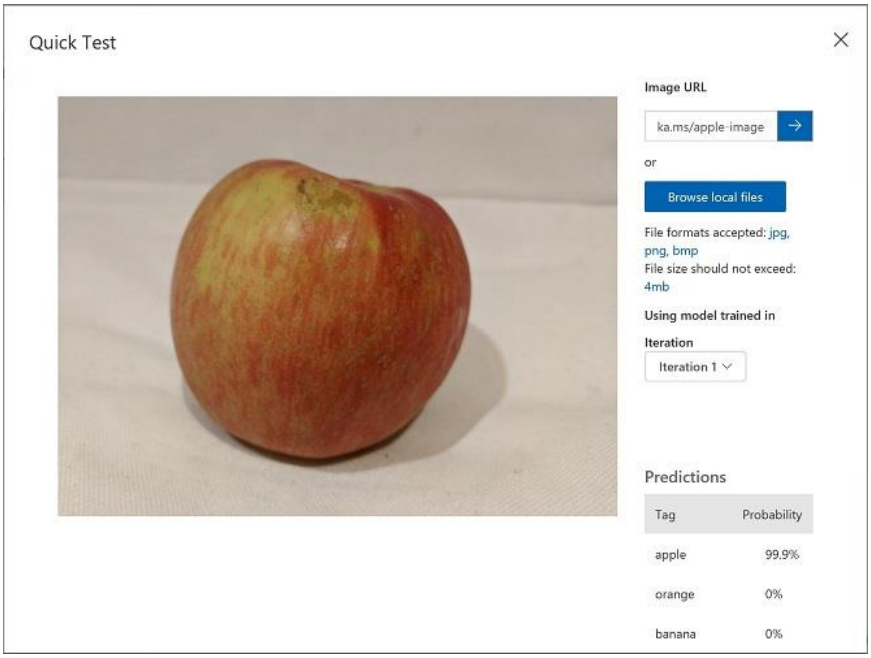


7. In the Custom Vision project, above the images, click **Train** to train a classification model using the tagged images. Select the **Quick Training** option, and then wait for the training iteration to complete (this may take a minute or so).
8. When the model iteration has been trained, review the *Precision*, *Recall*, and *AP* performance metrics - these measure the prediction accuracy of the classification model, and should all be high.

Test the model

Before publishing this iteration of the model for applications to use, you should test it.

1. Above the performance metrics, click **Quick Test**.
2. In the **Image URL** box, type https://aka.ms/apple-image and click ➔
3. View the predictions returned by your model - the probability score for *apple* should be the highest, like this:



4. Close the **Quick Test** window.

Publish the image classification model

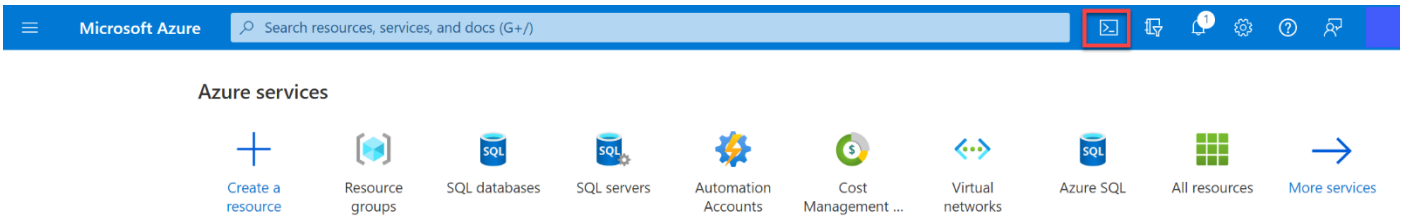
Now you're ready to publish your trained model and use it from a client application.

1. Click ✓ **Publish** to publish the trained model with the following settings:
    - **Model name**: groceries
    - **Prediction Resource**: *The prediction resource you created previously*.
2. After publishing, click the *Prediction URL* (🌐) icon to see information required to use the published model. Later, you will need the appropriate URL and Prediction-Key values to get a prediction from an Image URL, so keep this dialog box open and carry on to the next task.

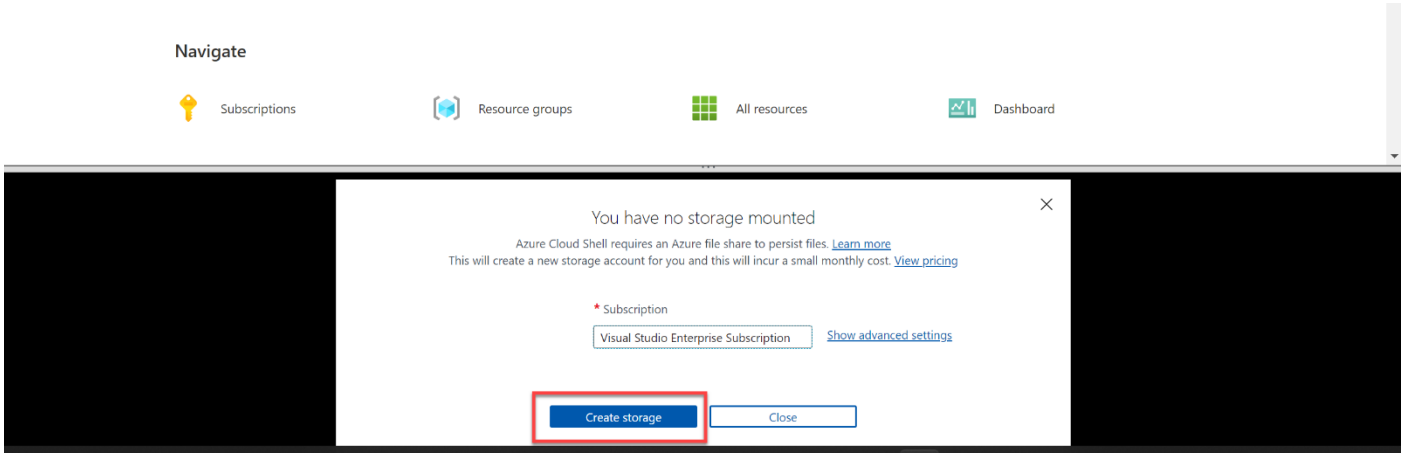
Run Cloud Shell

To test the capabilities of the Custom Vision service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

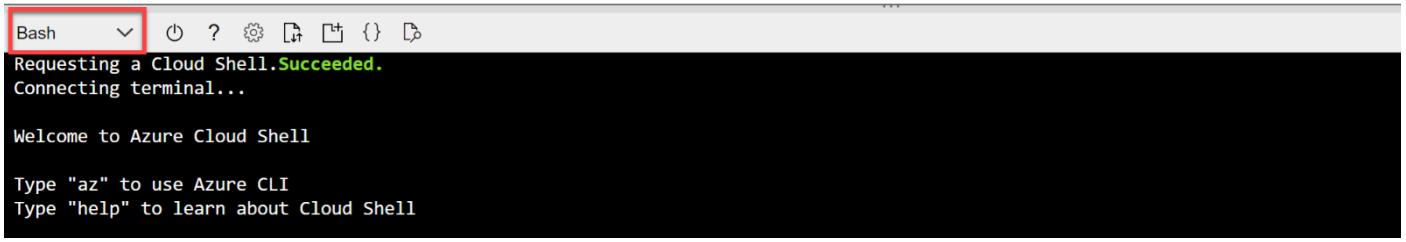
1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.



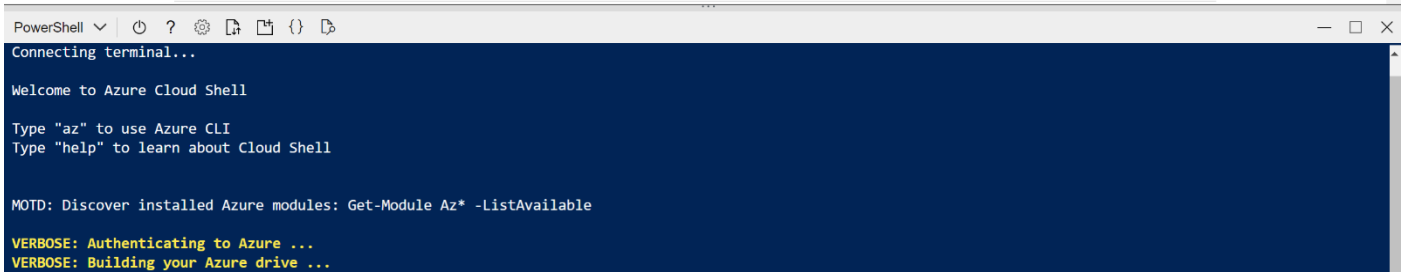
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.



4. Make sure the the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.



5. Wait for PowerShell to start. You should see the following screen in the Azure portal:



Configure and run a client application

Now that you have a Cloud Shell environment, you can run a simple application that uses the Custom Vision service to analyze an image.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

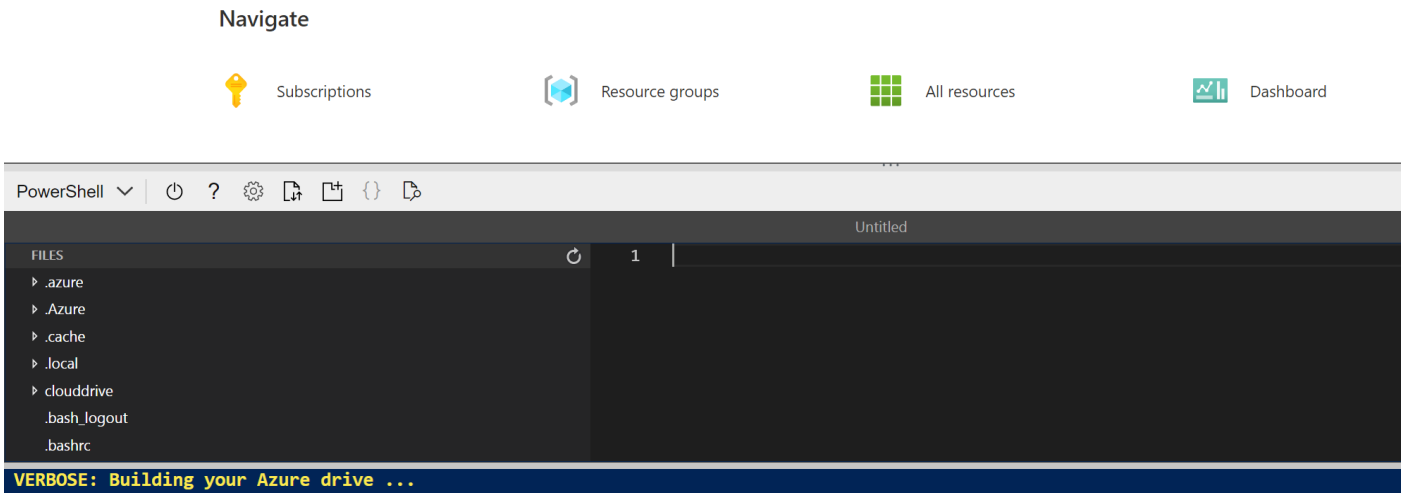
   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

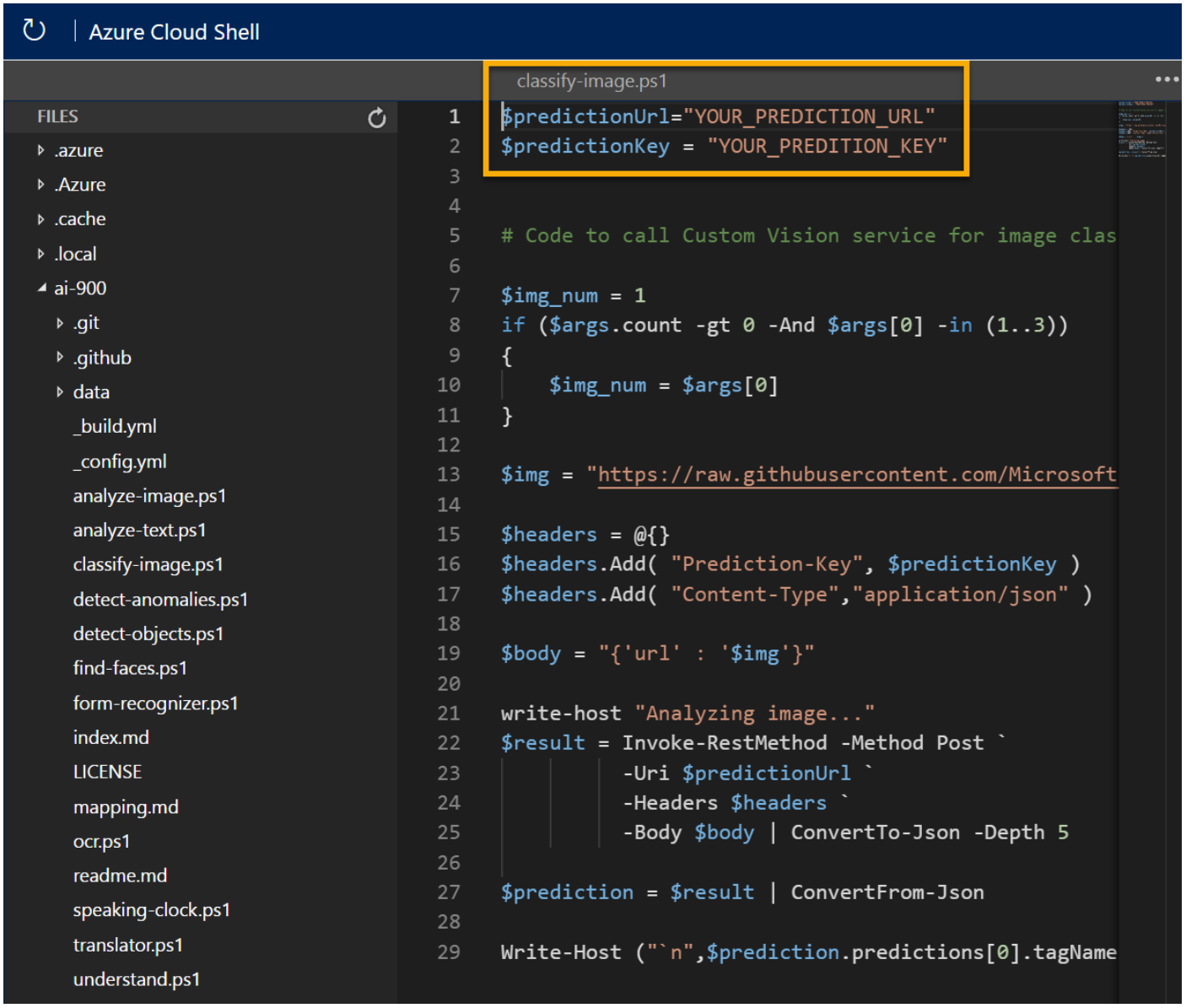
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **classify-image.ps1**. This file contains some code that uses the Custom Vision model to analyze an image, as shown here:



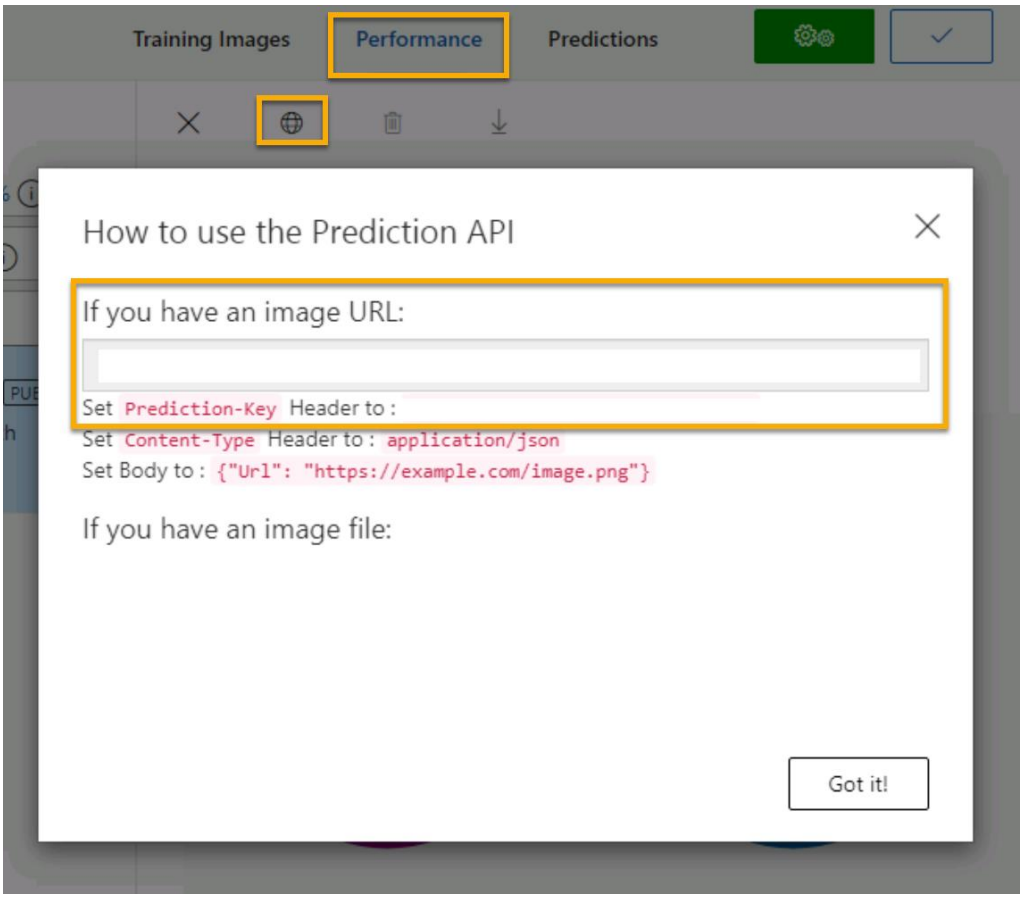
4. Don't worry too much about the details of the code, the important thing is that it needs the prediction URL and key for your Custom Vision model when using an image URL.

   Get the *prediction URL* from the dialog box in your Custom Vision project.

   Note

Remember, you reviewed the *prediction URL* after you published the image classification model. To find the *prediction URL*, navigate to the **Performance** tab in your project, then click on **Prediction URL** (if the screen is compressed, you may just see a globe icon). A dialogue box will appear. Copy the url for **If you have an image URL**. Paste it into the code editor, replacing **YOUR_PREDICTION_URL**.

Using the same dialog box, get the *prediction key*. Copy the prediction key displayed after *Set Prediction-Key Header to*. Paste it in the code editor, replacing the **YOUR_PREDICTION_KEY** placeholder value.



After pasting the Prediction URL and Prediction Key values, the first two lines of code should look similar to this:

PowerShellCopy

```
$predictionUrl="https..."
$predictionKey ="1a2b3c4d5e6f7g8h9i0j...."
```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   You will use the sample client application to classify several images into the apple, banana, or orange category.

6. We will classify this image:

   In the PowerShell pane, enter the following commands to run the code:

   Copy

   ```
   cd ai-900
   ./classify-image.ps1 1
   ```

7. Review the prediction, which should be **apple**.
8. Now let's try another image:

Please run this command:

Copy

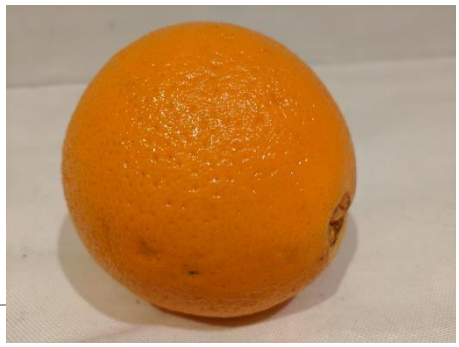
```
./classify-image.ps1 2
```

9. Verify that the model classifies this image as **banana**.
10. Finally, let's try the third test image:

Please run this command:

Copy

```
./classify-image.ps1 3
```



11. Verify that the model classifies this image as **orange**.

Learn more

This simple app shows only some of the capabilities of the Custom Vision service. To learn more about what you can do with this service, see the Custom Vision page.

Knowledge check

1. You plan to use the Custom Vision service to train an image classification model. You want to create a resource that can only be used for model training, and not for prediction. Which kind of resource should you create in your Azure subscription?

- Custom Vision
- Cognitive Services
- Computer Vision

2. You train an image classification model that achieves less than satisfactory evaluation metrics. How might you improve it?

- Reduce the size of the images used to train the model.
- Add a new label for "unknown" classes.
- Add more images to the training set.

3. You have published an image classification model. What information must you provide to developers who want to use it?

- Only the project ID.
- The project ID, the model name, and the key and endpoint for the prediction resource
- The project ID, iteration number, and the key and endpoint for the training resource.

Part 3/6_3/6

Detect objects in images with the Custom Vision service

Object detection is a form of computer vision in which artificial intelligence (AI) agents can identify and locate specific types of object in an image or camera feed.

Introduction

Object detection is a form of machine learning based computer vision in which a model is trained to recognize individual types of objects in an image, and to identify their location in the image.

Uses of object detection

Some sample applications of object detection include:

- **Checking for building safety**: Evaluating the safety of a building by analyzing footage of its interior for fire extinguishers or other emergency equipment.
- **Driving assistance**: Creating software for self-driving cars or vehicles with *lane assist* capabilities. The software can detect whether there is a car in another lane, and whether the driver's car is within its own lanes.
- **Detecting tumors**: Medical imaging such as an MRI or x-rays that can detect known objects for medical diagnosis.

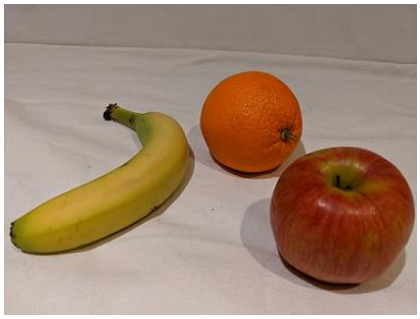
Learning objectives

Learn how to use the Custom Vision service to create an image detection solution. In this module you will:

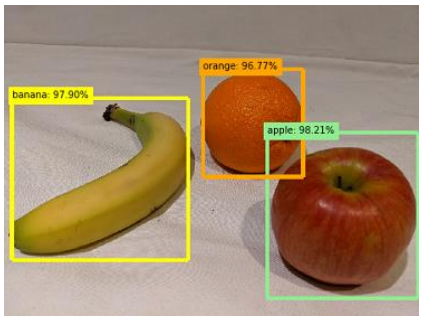
- Identify services in Azure for creating an object detection solution.
- Provision a Custom Vision resource.
- Train an object detection model.
- Publish and consume an object detection model.

What is object detection?

Let's look at example of object detection. Consider the following image:



An object detection model might be used to identify the individual objects in this image and return the following information:



Notice that an object detection model returns the following information:

- The *class* of each object identified in the image.
- The probability score of the object classification (which you can interpret as the *confidence* of the predicted class being correct)
- The coordinates of a *bounding box* for each object.

Note

Object detection vs. image classification

Image classification is a machine learning based form of computer vision in which a model is trained to categorize images based on the primary subject matter they contain. *Object detection* goes further than this to classify individual objects within the image, and to return the coordinates of a bounding box that indicates the object's location.

Get started with object detection on Azure

You can create an object detection machine learning model by using advanced deep learning techniques. However, this approach requires significant expertise and a large volume of training data. The **Custom Vision** cognitive service in Azure enables you to create object detection models that meet the needs of many computer vision scenarios with minimal deep learning expertise and fewer training images.

Azure resources for Custom Vision

Creating an object detection solution with Custom Vision consists of three main tasks. First you must use upload and tag images, then you can train the model, and finally you must publish the model so that client applications can use it to generate predictions.

For each of these tasks, you need a resource in your Azure subscription. You can use the following types of resource:

- **Custom Vision**: A dedicated resource for the custom vision service, which can be either a *training*, a *prediction* or a both resource.
- **Cognitive Services**: A general cognitive services resource that includes Custom Vision along with many other cognitive services. You can use this type of resource for *training*, *prediction*, or both.

The separation of training and prediction resources is useful when you want to track resource utilization for model training separately from client applications using the model to predict image classes. However, it can make development of an image classification solution a little confusing.

The simplest approach is to use a general Cognitive Services resource for both training and prediction. This means you only need to concern yourself with one *endpoint* (the HTTP address at which your service is hosted) and *key* (a secret value used by client applications to authenticate themselves).

If you choose to create a Custom Vision resource, you will be prompted to choose *training*, *prediction*, or *both* - and it's important to note that if you choose "both", then ***two*** resources are created - one for training and one for prediction.

It's also possible to take a mix-and-match approach in which you use a dedicated Custom Vision resource for training, but deploy your model to a Cognitive Services resource for prediction. For this to work, the training and prediction resources must be created in the same region.

Image tagging

Before you can train an object detection model, you must tag the classes and bounding box coordinates in a set of training images. This process can be time-consuming, but the *Custom Vision portal* provides a graphical interface that makes it straightforward. The interface will automatically suggest areas of the image where discrete objects are detected, and you can apply a class label to these suggested bounding boxes or drag to adjust the bounding box area. Additionally, after tagging and training with an initial dataset, the Computer Vision service can use *smart tagging* to suggest classes and bounding boxes for images you add to the training dataset.

Key considerations when tagging training images for object detection are ensuring that you have sufficient images of the objects in question, preferably from multiple angles; and making sure that the bounding boxes are defined tightly around each object.

Model training and evaluation

To train the model, you can use the *Custom Vision portal*, or if you have the necessary coding experience you can use one of the Custom Vision service programming language-specific software development kits (SDKs). Training an object detection model can take some time, depending on the number of training images, classes, and objects within each image.

Model training process is an iterative process in which the Custom Vision service repeatedly trains the model using some of the data, but holds some back to evaluate the model. At the end of the training process, the performance for the trained model is indicated by the following evaluation metrics:

- **Precision**: What percentage of class predictions did the model correctly identify? For example, if the model predicted that 10 images are oranges, of which eight were actually oranges, then the precision is 0.8 (80%).
- **Recall**: What percentage of the class predictions made by the model were correct? For example, if there are 10 images of apples, and the model found 7 of them, then the recall is 0.7 (70%).
- **Mean Average Precision (mAP)**: An overall metric that takes into account both precision and recall across all classes.

Using the model for prediction

After you've trained the model, and you're satisfied with its evaluated performance, you can publish the model to your prediction resource. When you publish the model, you can assign it a name (the default is "Iteration*X*", where X is the number of times you have trained the model).

To use you model, client application developers need the following information:

- **Project ID**: The unique ID of the Custom Vision project you created to train the model.
- **Model name**: The name you assigned to the model during publishing.
- **Prediction endpoint**: The HTTP address of the endpoints for the *prediction* resource to which you published the model (***not*** the training resource).
- **Prediction key**: The authentication key for the *prediction* resource to which you published the model (***not*** the training resource).

Exercise - Create an object detection solution

Object detection is a form of computer vision in which a machine learning model is trained to classify individual instances of objects in an image, and indicate a *bounding box* that marks its location. You can think of this as a progression from *image classification* (in which the model answers the question "what is this an image of?") to building solutions where we can ask the model "what objects are in this image, and where are they?".

For example, a grocery store might use an object detection model to implement an automated checkout system that scans a conveyor belt using a camera, and can identify specific items without the need to place each item on the belt and scan them individually.

The **Custom Vision** cognitive service in Microsoft Azure provides a cloud-based solution for creating and publishing custom object detection models. In Azure, you can use the Custom Vision service to train an image classification model based on existing images. There are two elements to creating an image classification solution. First, you must train a model to recognize different classes using existing images. Then, when the model is trained you must publish it as a service that can be consumed by applications.

To test the capabilities of the Custom Vision service to detect objects in images, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Custom Vision service by creating either a **Custom Vision** resource or a **Cognitive Services** resource.

Note

Not every resource is available in every region. Whether you create a Custom Vision or Cognitive Services resource, only resources created in **certain regions** can be used to access Custom Vision services. For simplicity, a region is pre-selected for you in the configuration instructions below.

Create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: East US
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

Create a Custom Vision project

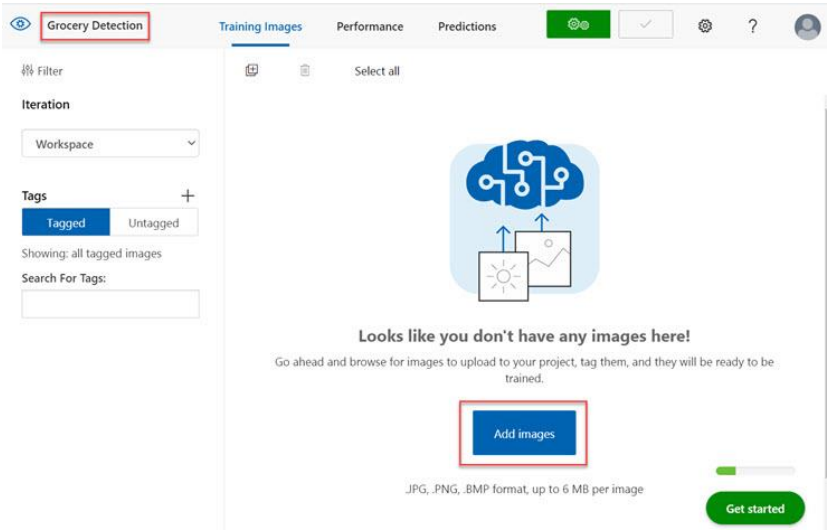
To train an object detection model, you need to create a Custom Vision project based on your training resource. To do this, you'll use the Custom Vision portal.

1. In a new browser tab, open the Custom Vision portal at https://customvision.ai, and sign in using the Microsoft account associated with your Azure subscription.
2. Create a new project with the following settings:
    - **Name**: Grocery Detection
    - **Description**: Object detection for groceries.
    - **Resource**: *The resource you created previously*
    - **Project Types**: Object Detection
    - **Domains**: General
3. Wait for the project to be created and opened in the browser.

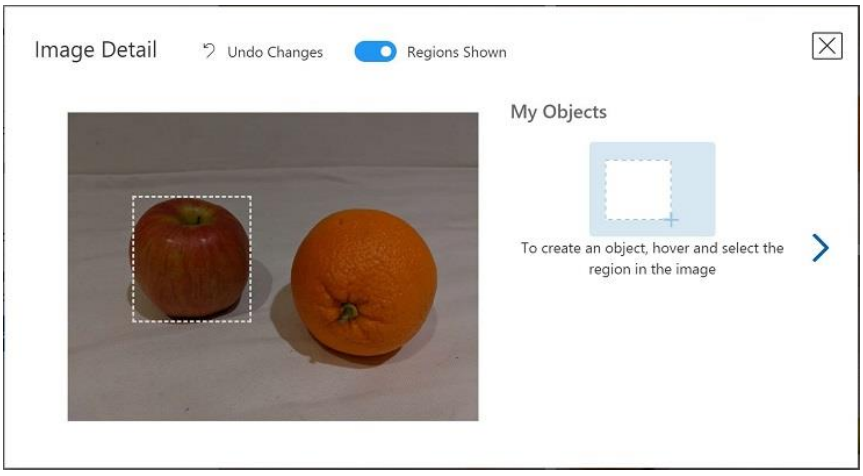
Add and tag images

To train an object detection model, you need to upload images that contain the classes you want the model to identify, and tag them to indicate bounding boxes for each object instance.

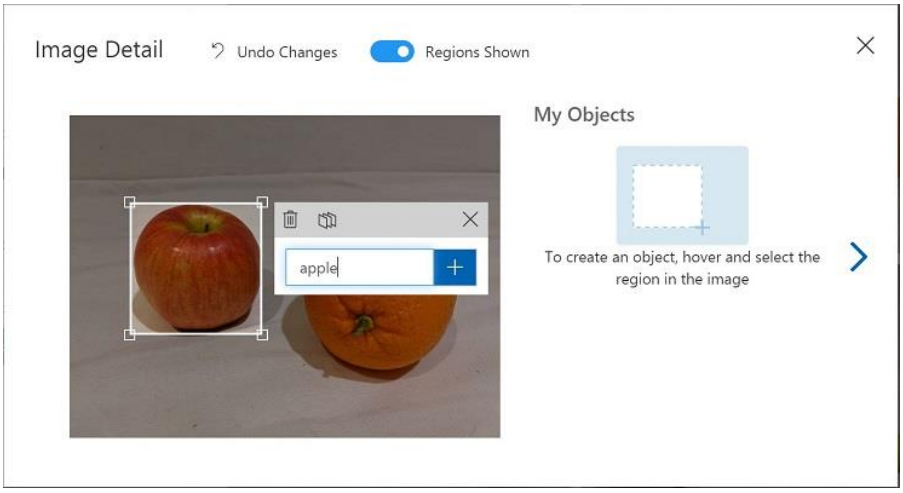
1. Download and extract the training images from https://aka.ms/fruit-objects. The extracted folder contains a collection of images of fruit.
2. In the Custom Vision portal https://customvision.ai, make sure you are working in your object detection project *Grocery Detection*. Then select **Add images** and upload all of the images in the extracted folder.



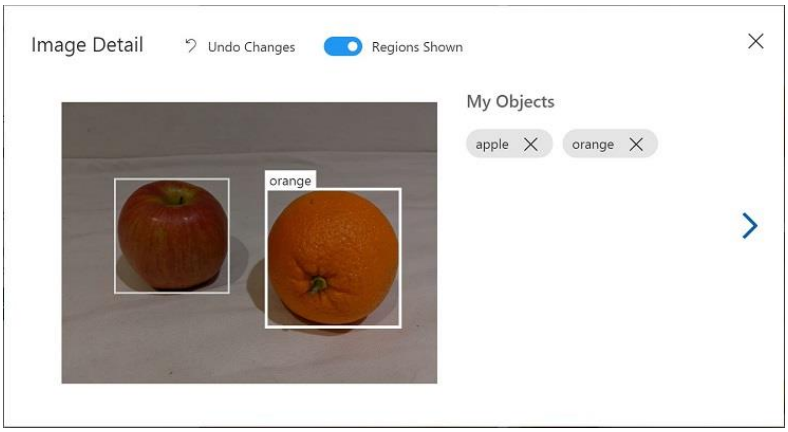
3. After the images have been uploaded, select the first one to open it.
4. Hold the mouse over any object in the image until an automatically detected region is displayed like the image below. Then select the object, and if necessary resize the region to surround it.



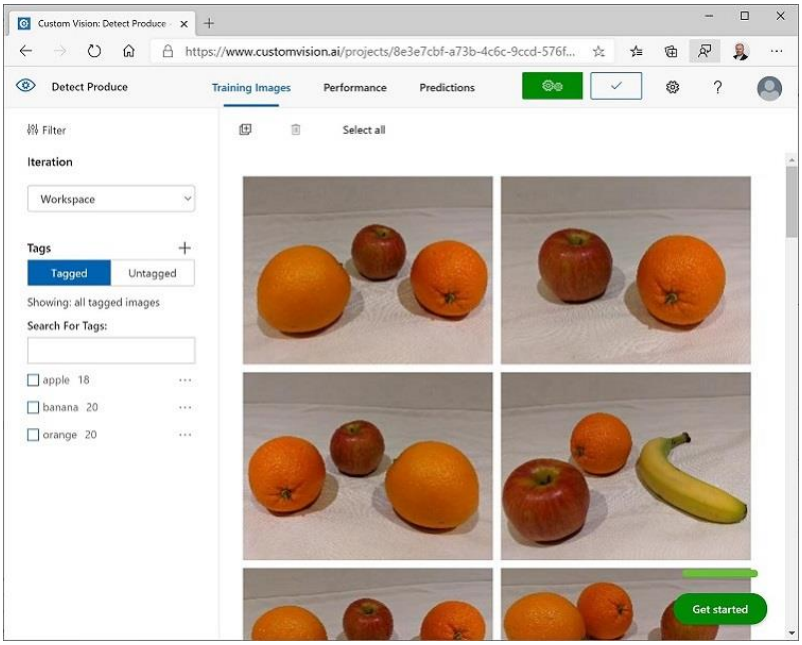
   Alternatively, you can simply drag around the object to create a region.
5. When the region surrounds the object, add a new tag with the appropriate object type (*apple*, *banana*, or *orange*) as shown here:



6. Select and tag each other object in the image, resizing the regions and adding new tags as required.



7. Use the > link on the right to go to the next image, and tag its objects. Then just keep working through the entire image collection, tagging each apple, banana, and orange.
8. When you have finished tagging the last image, close the **Image Detail** editor and on the **Training Images** page, under **Tags**, select **Tagged** to see all of your tagged images:



Train and test a model

Now that you've tagged the images in your project, you're ready to train a model.

1. In the Custom Vision project, click **Train** to train an object detection model using the tagged images. Select the **Quick Training** option.

2. Wait for training to complete (it might take ten minutes or so), and then review the *Precision*, *Recall*, and *mAP* performance metrics - these measure the prediction goodness of the object detection model, and should all be high.
3. At the top right of the page, click **Quick Test**, and then in the **Image URL** box, enter https://aka.ms/apple-orange and view the prediction that is generated. Then close the **Quick Test** window.

Publish the object detection model

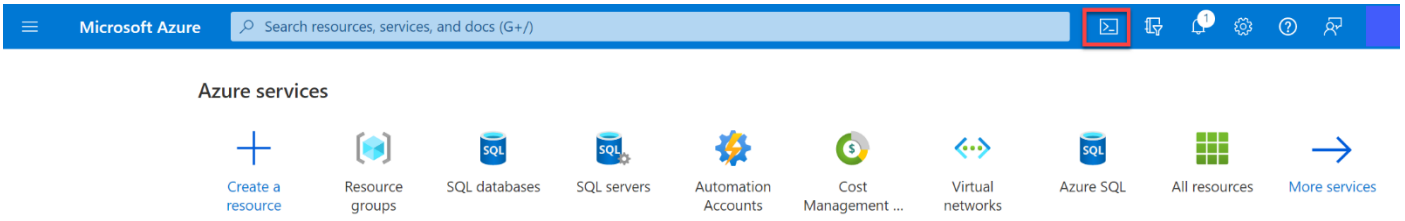
Now you're ready to publish your trained model and use it from a client application.

1. Click ✓ **Publish** to publish the trained model with the following settings:
    - **Model name**: detect-produce
    - **Prediction Resource**: *The resource you created previously*.
2. After publishing, click the *Prediction URL* (🌐) icon to see information required to use the published model. Later, you will need the appropriate URL and Prediction-Key values to get a prediction from an Image URL, so keep this dialog box open and carry on to the next task.

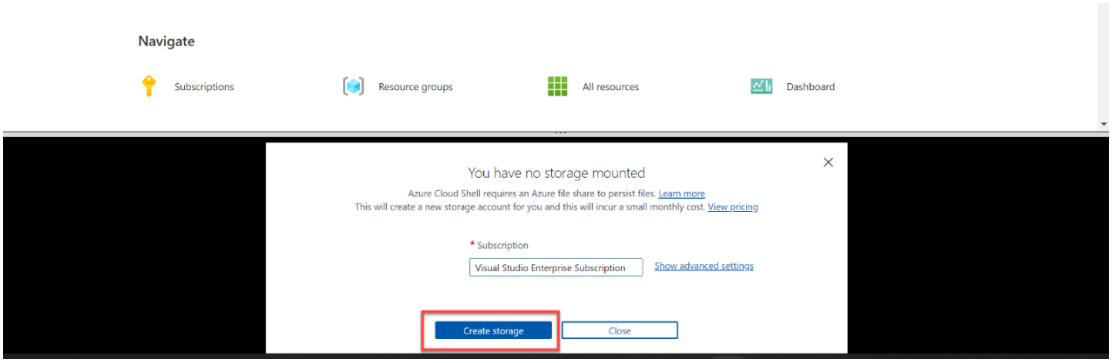
Run Cloud Shell

To test the capabilities of the Custom Vision service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

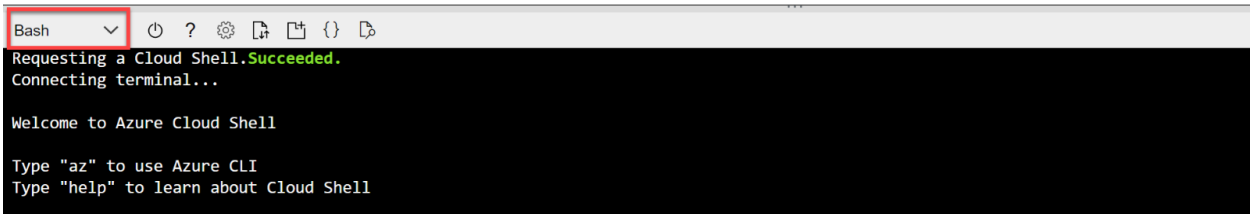
1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.



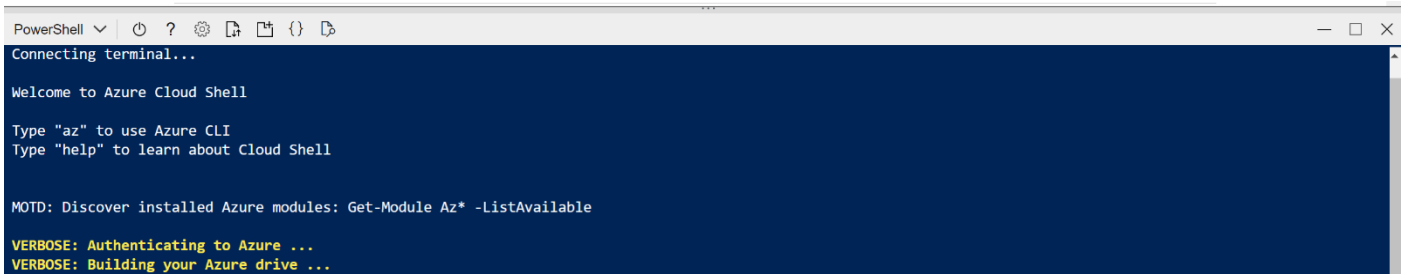
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.



4. Make sure the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.



5. Wait for PowerShell to start. You should see the following screen in the Azure portal:



Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the Custom Vision service to detect objects in an image.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900

   Note

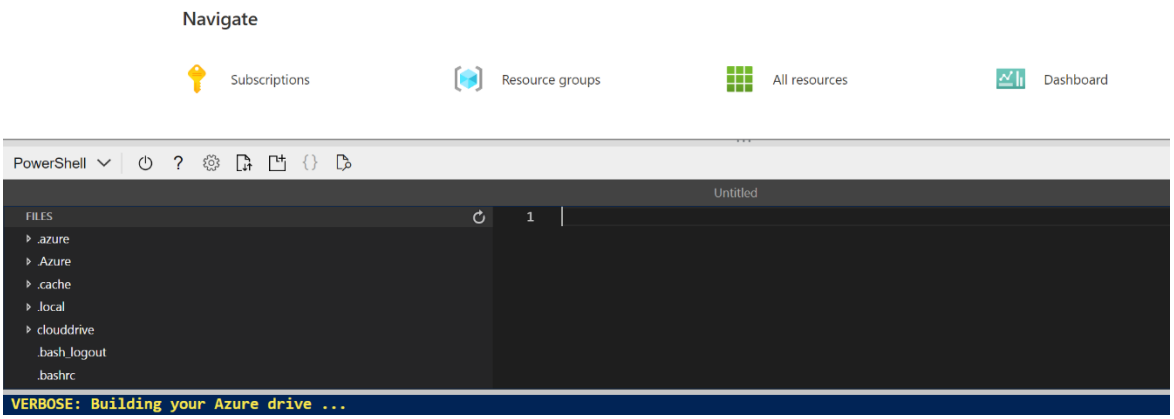
   If you already used this command in another lab to clone the *ai-900* repository, you can skip this step.

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

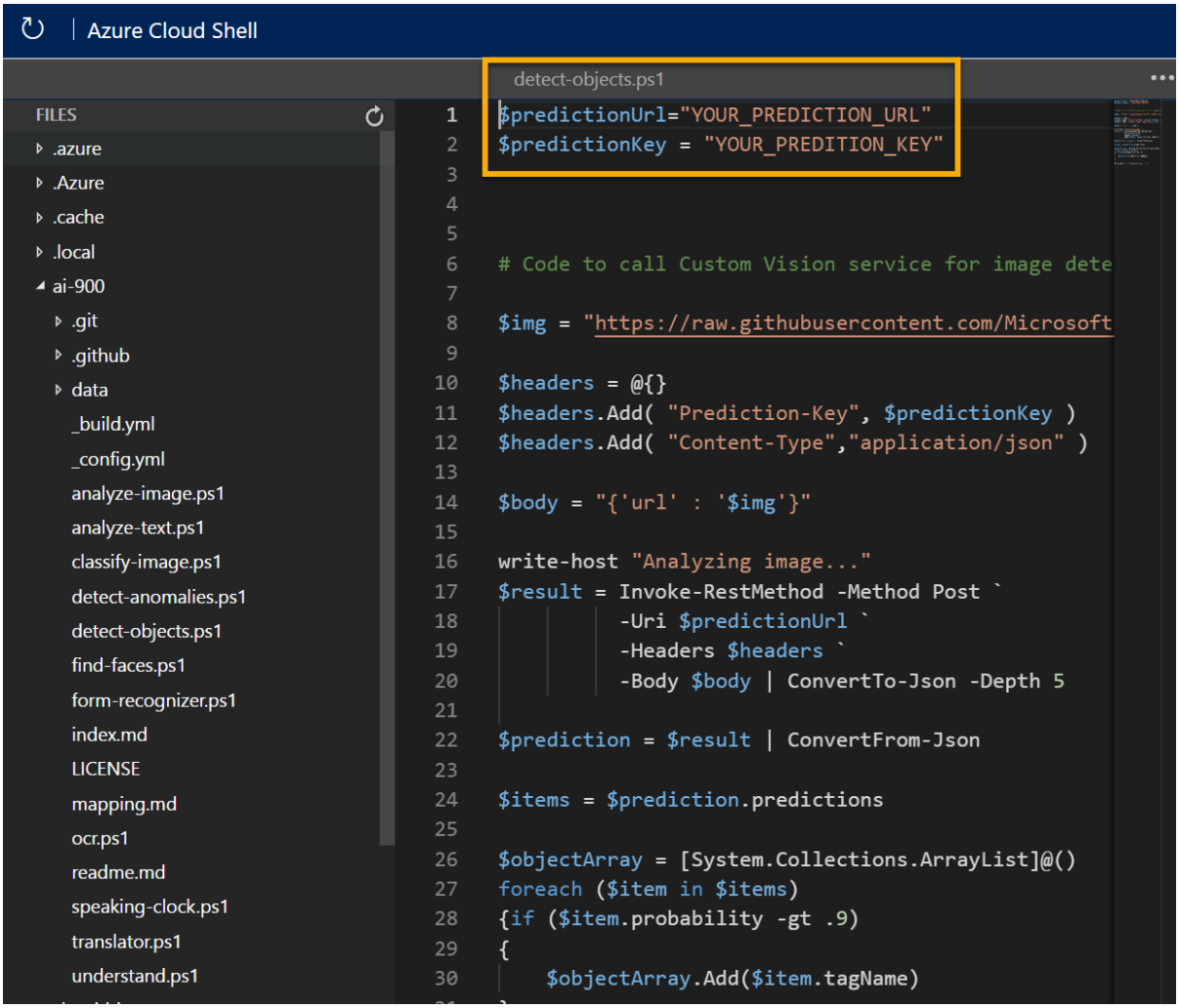
   Copy

   code .

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **detect-objects.ps1**. This file contains some code that uses the Custom Vision service to detect objects an image, as shown here:



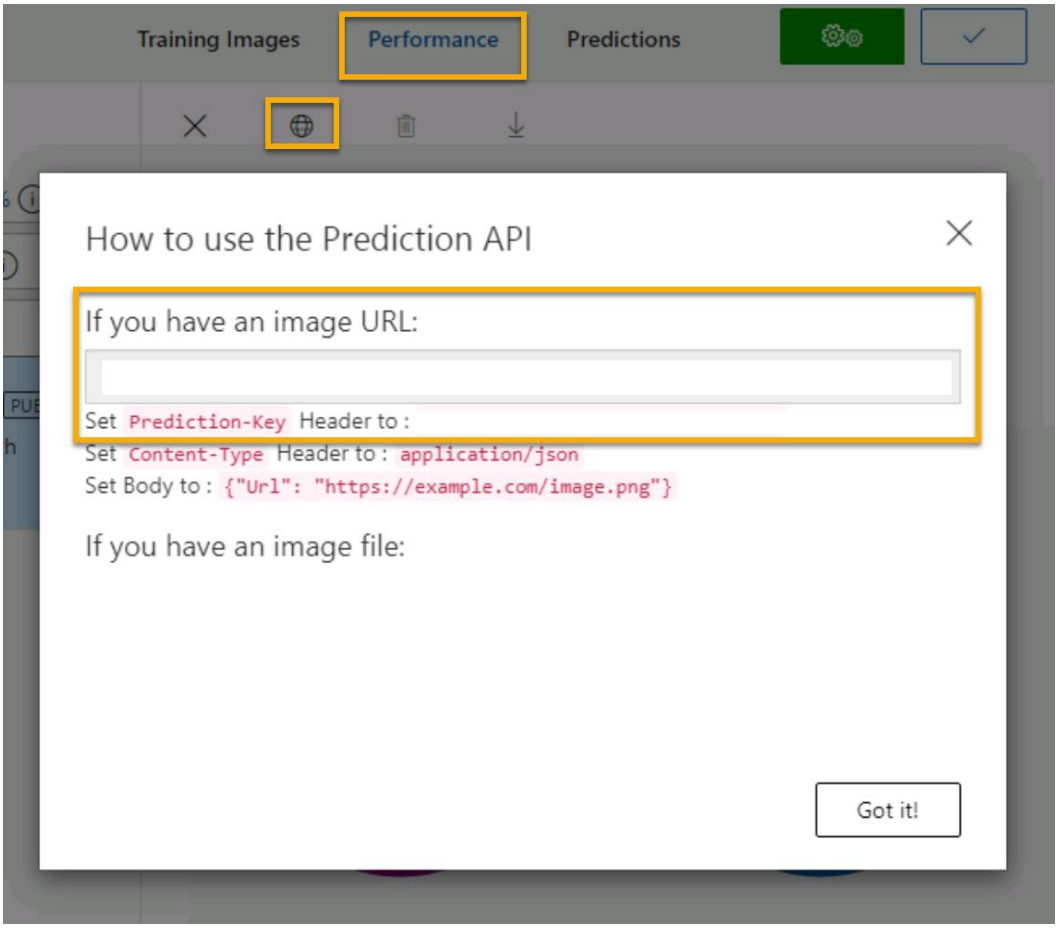
4. Don't worry too much about the details of the code, the important thing is that it needs the prediction URL and key for your Custom Vision model when using an image URL.

   Get the *prediction URL* from the dialog box in your Custom Vision project.

   Note

   Remember, you reviewed the *prediction URL* after you published the image classification model. To find the *prediction URL*, navigate to the **Performance** tab in your project, then click on **Prediction URL** (if the screen is compressed, you may just see a globe icon). A dialogue box will appear. Copy the url for **If you have an image URL**. Paste it into the code editor, replacing **YOUR_PREDICTION_URL**.

   Using the same dialog box, get the *prediction key*. Copy the prediction key displayed after *Set Prediction-Key Header to*. Paste it in the code editor, replacing the **YOUR_PREDICTION_KEY** placeholder value.



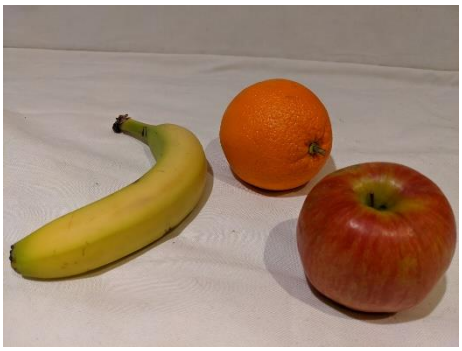
After pasting the Prediction URL and Prediction Key values, the first two lines of code should look similar to this:

PowerShellCopy

```
$predictionUrl="https..."
$predictionKey ="1a2b3c4d5e6f7g8h9i0j...."
```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   You will use the sample client application to detect objects in this image:



6. In the PowerShell pane, enter the following command to run the code:

   Copy

   ```
   cd ai-900
   ./detect-objects.ps1
   ```

7. Review the prediction, which should be *apple orange banana**.

Learn more

This simple app shows only some of the capabilities of the Custom Vision service. To learn more about what you can do with this service, see the Custom Vision page.

Knowledge check

1. Which of the following results does an object detection model typically return for an image?

- A class label and probability score for the image
- Bounding box coordinates that indicates the region of the image where all of the objects it contains are located
- A class label, probability, and bounding box for each object in the image

2. You plan to use a set of images to train an object detection model, and then publish the model as a predictive service. You want to use a single Azure resource with the same key and endpoint for training and prediction. What kind of Azure resource should you create?

- Cognitive Services
- Custom Vision
- Computer Vision

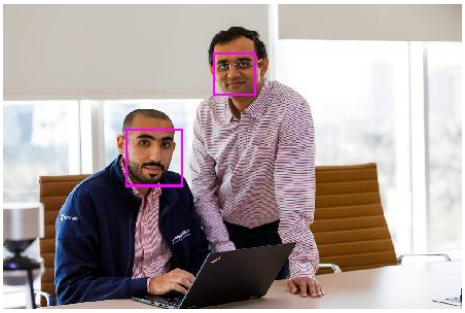
Part 3/6_4/6

Detect and analyze faces with the Face service

Face detection, analysis, and recognition is an important capability for artificial intelligence (AI) solutions. The Face cognitive service in Azure makes it easy integrate these capabilities into your applications.

Introduction

Face detection and analysis is an area of artificial intelligence (AI) in which we use algorithms to locate and analyze human faces in images or video content.

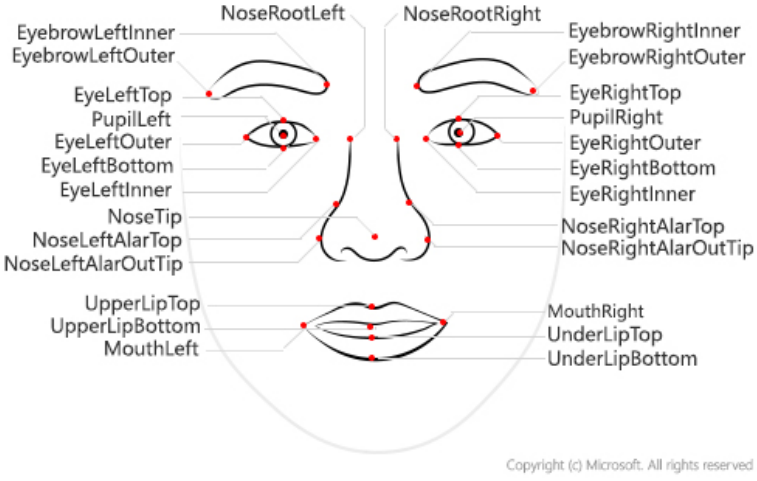


Face detection

Face detection involves identifying regions of an image that contain a human face, typically by returning *bounding box* coordinates that form a rectangle around the face, like this:

Facial analysis

Moving beyond simple face detection, some algorithms can also return other information, such as facial landmarks (nose, eyes, eyebrows, lips, and others).



These facial landmarks can be used as features with which to train a machine learning model.

Facial recognition

A further application of facial analysis is to train a machine learning model to identify known individuals from their facial features. This usage is more generally known as *facial recognition*, and involves using multiple images of each person you want to recognize to train a model so that it can detect those individuals in new images on which it wasn't trained.

Uses of face detection and analysis

There are many applications for face detection, analysis, and recognition. For example,

- Security - facial recognition can be used in building security applications, and increasingly it is used in smart phones operating systems for unlocking devices.
- Social media - facial recognition can be used to automatically tag known friends in photographs.
- Intelligent monitoring - for example, an automobile might include a system that monitors the driver's face to determine if the driver is looking at the road, looking at a mobile device, or shows signs of tiredness.
- Advertising - analyzing faces in an image can help direct advertisements to an appropriate demographic audience.
- Missing persons - using public cameras systems, facial recognition can be used to identify if a missing person is in the image frame.
- Identity validation - useful at ports of entry kiosks where a person holds a special entry permit.

When used responsibly, facial recognition is an important and useful technology that can improve efficiency, security, and customer experiences. Face is a building block for creating a facial recognition system.

Get started with Face analysis on Azure

Microsoft Azure provides multiple cognitive services that you can use to detect and analyze faces, including:

- **Computer Vision**, which offers face detection and some basic face analysis, such as returning the bounding box coordinates around an image.
- **Video Indexer**, which you can use to detect and identify faces in a video.
- **Face**, which offers pre-built algorithms that can detect, recognize, and analyze faces.

Of these, Face offers the widest range of facial analysis capabilities.

Face

Face can return the rectangle coordinates for any human faces that are found in an image, as well as a series of attributes related to those faces such as:

- **Blur**: how blurred the face is (which can be an indication of how likely the face is to be the main focus of the image)
- **Exposure**: aspects such as underexposed or over exposed and applies to the face in the image and not the overall image exposure
- **Glasses**: if the person is wearing glasses
- **Head pose**: the face's orientation in a 3D space
- **Noise**: refers to visual noise in the image. If you have taken a photo with a high ISO setting for darker settings, you would notice this noise in the image. The image looks grainy or full of tiny dots that make the image less clear
- **Occlusion**: determines if there may be objects blocking the face in the image

Responsible AI use

Important

To support Microsoft's **Responsible AI Standard**, a new **Limited Access policy** has been implemented for the Face service and Computer Vision service.

Anyone can use the Face service to:

- Detect the location of faces in an image

- Determine if a face is wearing glasses
- Determine if there's occlusion, blur, noise, or over/under exposure for any of the faces
- Return the head pose coordinates for each face in an image

The Limited Access policy requires customers to submit an intake form to access additional Face service capabilities including:

- The ability to compare faces for similarity
- The ability to identify named individuals in an image

Azure resources for Face

To use Face, you must create one of the following types of resource in your Azure subscription:

- **Face**: Use this specific resource type if you don't intend to use any other cognitive services, or if you want to track utilization and costs for Face separately.
- **Cognitive Services**: A general cognitive services resource that includes Computer Vision along with many other cognitive services; such as Computer Vision, Text Analytics, Translator Text, and others. Use this resource type if you plan to use multiple cognitive services and want to simplify administration and development.

Whichever type of resource you choose to create, it will provide two pieces of information that you will need to use it:

- A **key** that is used to authenticate client applications.
- An **endpoint** that provides the HTTP address at which your resource can be accessed.

Note

If you create a Cognitive Services resource, client applications use the same key and endpoint regardless of the specific service they are using.

Tips for more accurate results

There are some considerations that can help improve the accuracy of the detection in the images:

- image format - supported images are JPEG, PNG, GIF, and BMP
- file size - 6 MB or smaller
- face size range - from 36 x 36 up to 4096 x 4096. Smaller or larger faces will not be detected
- other issues - face detection can be impaired by extreme face angles, occlusion (objects blocking the face such as a hand). Best results are obtained when the faces are full-frontal or as near as possible to full-frontal.

Exercise - Detect and analyze faces with the Face service

Computer vision solutions often require an artificial intelligence (AI) solution to be able to detect human faces. For example, suppose the retail company Northwind Traders wants to locate where customers are standing in a store to best assist them. One way to accomplish this is to determine if there are any faces in the images, and if so, to identify the bounding box coordinates around the faces.

To test the capabilities of the Face service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Face service by creating either a **Face** resource or a **Cognitive Services** resource.

If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.

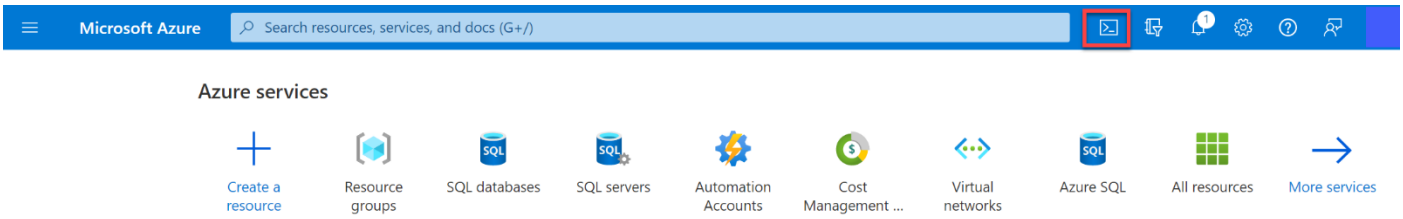
- **Resource group**: *Select or create a resource group with a unique name*.
- **Region**: *Choose any available region*:
- **Name**: *Enter a unique name*.
- **Pricing tier**: S0
- **I confirm I have read and understood the notices**: Selected.

3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

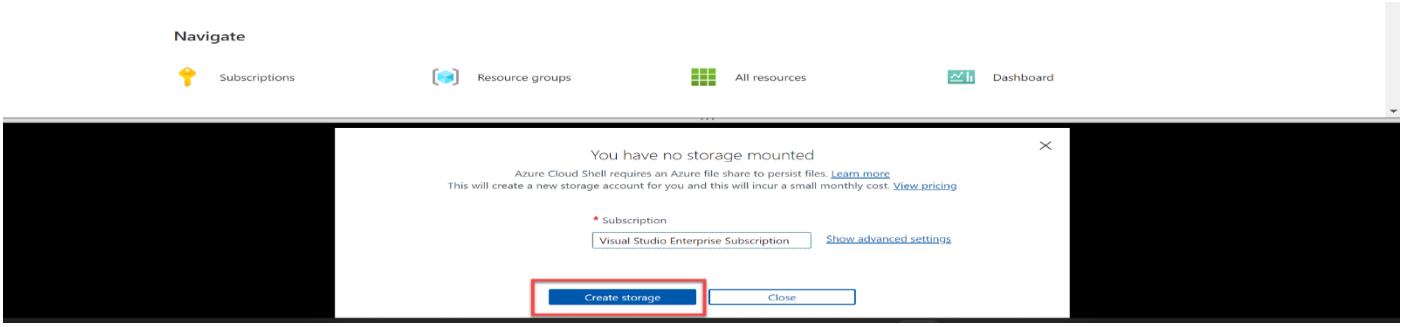
Run Cloud Shell

To test the capabilities of the Face service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

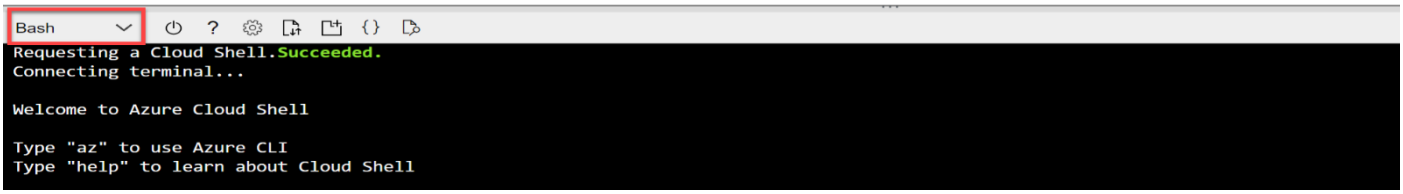
1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.



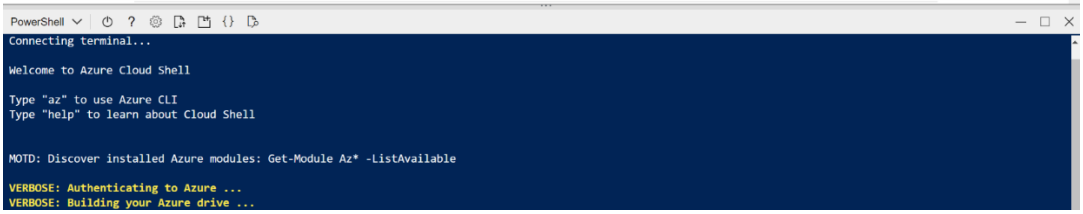
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.



4. Make sure the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.



5. Wait for PowerShell to start. You should see the following screen in the Azure portal:



Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the Face service.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

   Tip

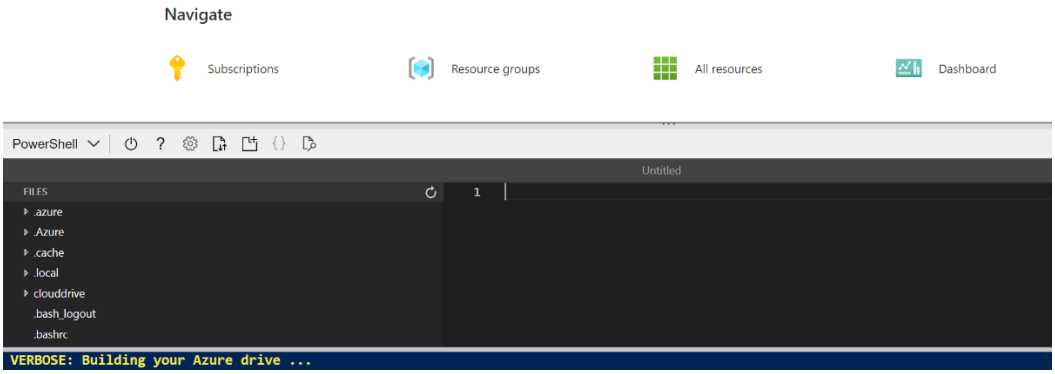
   If you already used this command in another lab to clone the *ai-900* repository, you can skip this step.

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

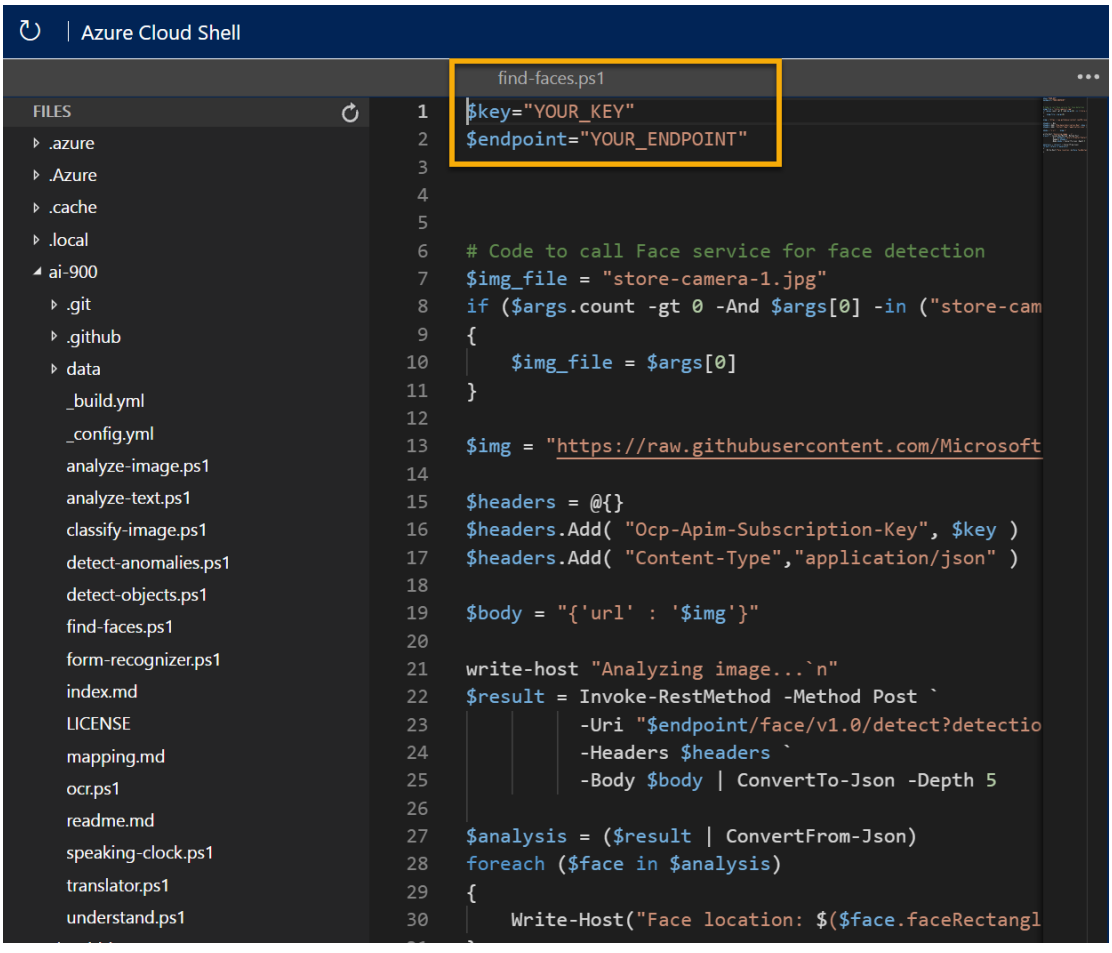
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **find-faces.ps1**. This file contains some code that uses the Face service to detect and analyze faces in an image, as shown here:



4. Don't worry too much about the details of the code, the important thing is that it needs the endpoint URL and either of the keys for your Cognitive Services resource. Copy these from the **Keys and Endpoints** page for your resource from the Azure portal and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_ENDPOINT** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

   After pasting the key and endpoint values, the first two lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $endpoint="https..."
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   The sample client application will use your Face service to analyze the following image, taken by a camera in the Northwind Traders store:

6. In the PowerShell pane, enter the following commands to run the code:

   Copy

   ```
   cd ai-900
   ./find-faces.ps1 store-camera-1.jpg
   ```

7. Review the returned information, which includes the location of the face in the image. The location of a face is indicated by the top-left coordinates, and the width and height of a *bounding box*, as shown here:

Note

From June 21st 2022, Face service capabilities that return personally identifiable features are restricted. See **https://azure.microsoft.com/blog/responsible-ai-investments-and-safeguards-for-facial-recognition/** for details.

8. Now let's try another image:

   To analyze the second image, enter the following command:

   Copy

   ```
   ./find-faces.ps1 store-camera-2.jpg
   ```

9. Review the results of the face analysis for the second image.
10. Let's try one more:

    To analyze the third image, enter the following command:

    Copy

    ```
    ./find-faces.ps1 store-camera-3.jpg
    ```

11. Review the results of the face analysis for the third image.

Learn more

This simple app shows only some of the capabilities of the Face service. To learn more about what you can do with this service, see the Face page.

Part 3/6_5/6

Read text with the Computer Vision service

Optical character recognition (OCR) enables artificial intelligence (AI) systems to read text in images, enabling applications to extract information from photographs, scanned documents, and other sources of digitized text.

Introduction

The ability for computer systems to process written or printed text is an area of artificial intelligence (AI) where *computer vision* intersects with *natural language processing*. You need computer vision capabilities to "read" the text, and then you need natural language processing capabilities to make sense of it.

The basic foundation of processing printed text is *optical character recognition* (OCR), in which a model can be trained to recognize individual shapes as letters, numerals, punctuation, or other elements of text. Much of the early work on implementing this kind of capability was performed by postal services to support automatic sorting of mail based on postal codes. Since then, the state-of-the-art for reading text has moved on, and it's now possible to build models that can detect printed or handwritten text in an image and read it line-by-line or even word-by-word.

At the other end of the scale, there is *machine reading comprehension* (MRC), in which an AI system not only reads the text characters, but can use a semantic model to interpret what the text is about.

In this module, we'll focus on the use of OCR technologies to detect text in images and convert it into a text-based data format, which can then be stored, printed, or used as the input for further processing or analysis.

Uses of OCR

The ability to recognize printed and handwritten text in images, is beneficial in many scenarios such as:

- note taking
- digitizing forms, such as medical records or historical documents
- scanning printed or handwritten checks for bank deposits

Get started with OCR on Azure

The ability to extract text from images is handled by the Computer Vision service, which also provides image analysis capabilities.

Azure resources for Computer Vision

The first step towards using the Computer Vision service is to create a resource for it in your Azure subscription. You can use either of the following resource types:

- **Computer Vision**: A specific resource for the Computer Vision service. Use this resource type if you don't intend to use any other cognitive services, or if you want to track utilization and costs for your Computer Vision resource separately.
- **Cognitive Services**: A general cognitive services resource that includes Computer Vision along with many other cognitive services; such as Text Analytics, Translator Text, and others. Use this resource type if you plan to use multiple cognitive services and want to simplify administration and development.

Whichever type of resource you choose to create, it will provide two pieces of information that you will need to use it:

- A **key** that is used to authenticate client applications.
- An **endpoint** that provides the HTTP address at which your resource can be accessed.

Note

If you create a Cognitive Services resource, client applications use the same key and endpoint regardless of the specific service they are using.

Use the Computer Vision service to read text

Many times an image contains text. It can be typewritten text or handwritten. Some common examples are images with road signs, scanned documents that are in an image format such as JPEG or PNG file formats, or even just a picture taken of a white board that was used during a meeting.

The Computer Vision service provides two application programming interfaces (APIs) that you can use to read text in images: the **OCR** API and the **Read** API.

The OCR API

The OCR API is designed for quick extraction of small amounts of text in images. It operates synchronously to provide immediate results, and can recognize text in numerous languages.

When you use the OCR API to process an image, it returns a hierarchy of information that consists of:

- **Regions** in the image that contain text
- **Lines** of text in each region
- **Words** in each line of text

For each of these elements, the OCR API also returns *bounding box* coordinates that define a rectangle to indicate the location in the image where the region, line, or word appears.

The Read API

The OCR method can have issues with false positives when the image is considered text-dominate. The Read API uses the latest recognition models and is optimized for images that have a significant amount of text or has considerable visual noise.

The Read API is a better option for scanned documents that have a lot of text. The Read API also has the ability to automatically determine the proper recognition model to use, taking into consideration lines of text and supporting images with printed text as well as recognizing handwriting.

Because the Read API can work with larger documents, it works asynchronously so as not to block your application while it is reading the content and returning results to your application. This means that to use the Read API, your application must use a three-step process:

1. Submit an image to the API, and retrieve an *operation ID* in response.
2. Use the operation ID to check on the status of the image analysis operation, and wait until it has completed.
3. Retrieve the results of the operation.

The results from the Read API are arranged into the following hierarchy:

- **Pages** - One for each page of text, including information about the page size and orientation.
- **Lines** - The lines of text on a page.
- **Words** - The words in a line of text.

Each line and word includes bounding box coordinates indicating its position on the page.

Exercise - Read text with the Computer Vision service

A common computer vision challenge is to detect and interpret text in an image. This kind of processing is often referred to as *optical character recognition* (OCR).

To test the capabilities of the OCR service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Use the Computer Vision Service to Read Text in an Image

The **Computer Vision** cognitive service provides support for OCR tasks, including:

- An **OCR** API that you can use to read text in multiple languages. This API can be used synchronously, and works well when you need to detect and read a small amount of text in an image.
- A **Read** API that is optimized for larger documents. This API is used asynchronously, and can be used for both printed and handwritten text.

Create a *Cognitive Services* resource

You can use the Computer Vision service by creating either a **Computer Vision** resource or a **Cognitive Services** resource.

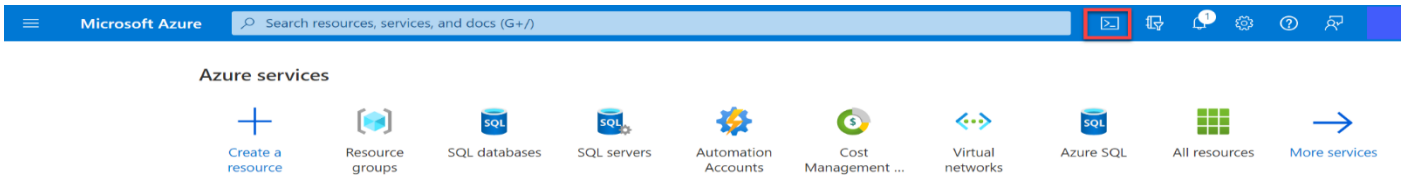
If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: *Choose any available region*:
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

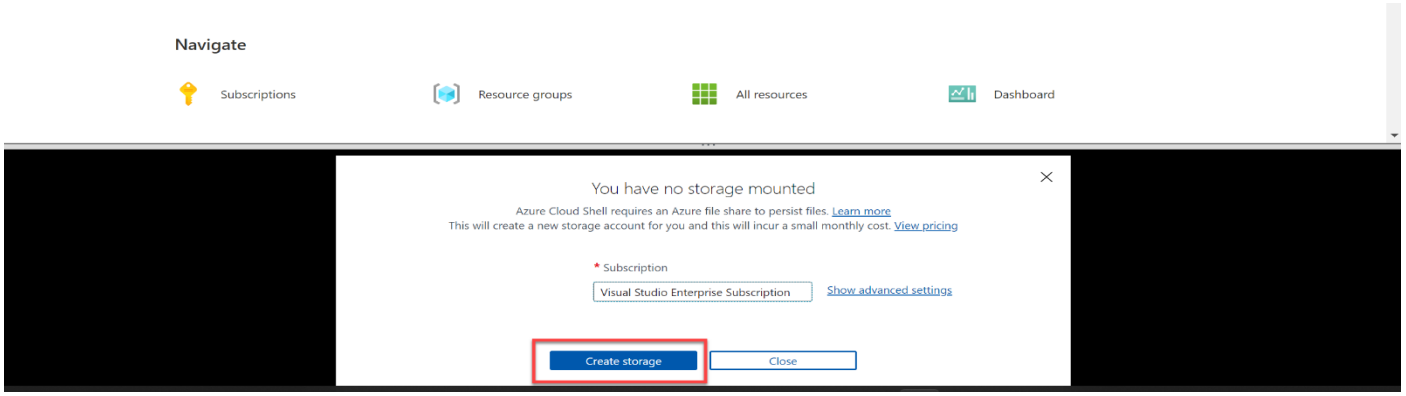
Run Cloud Shell

To test the capabilities of the Custom Vision service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

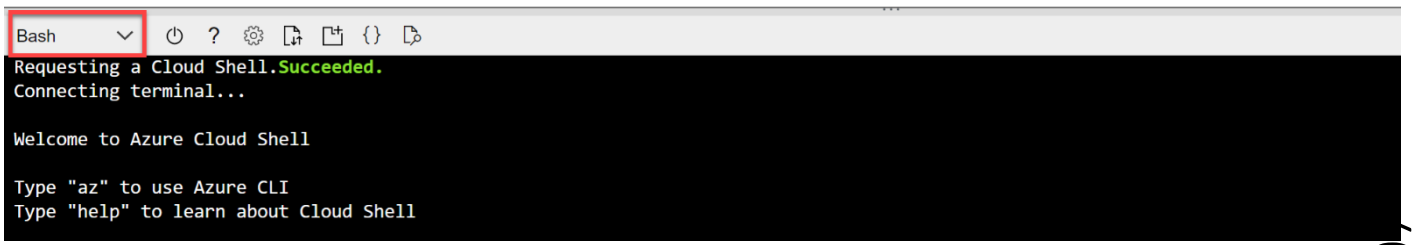
1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.



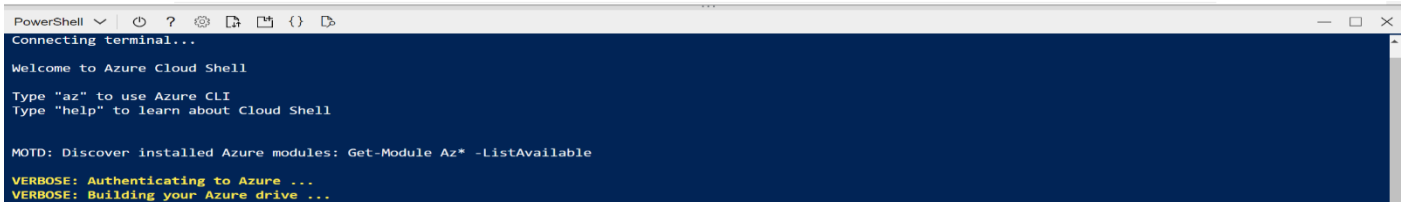
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.



4. Make sure the the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.



5. Wait for PowerShell to start. You should see the following screen in the Azure portal:



Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the OCR service.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

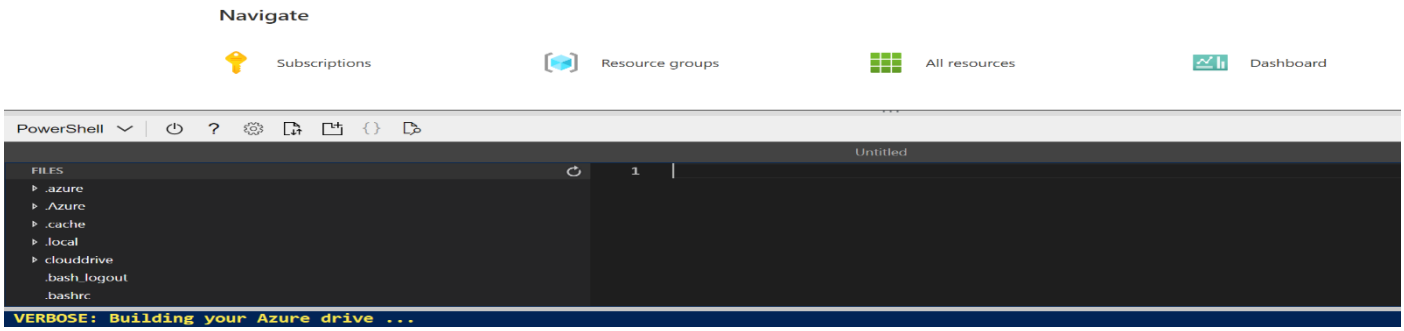
   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

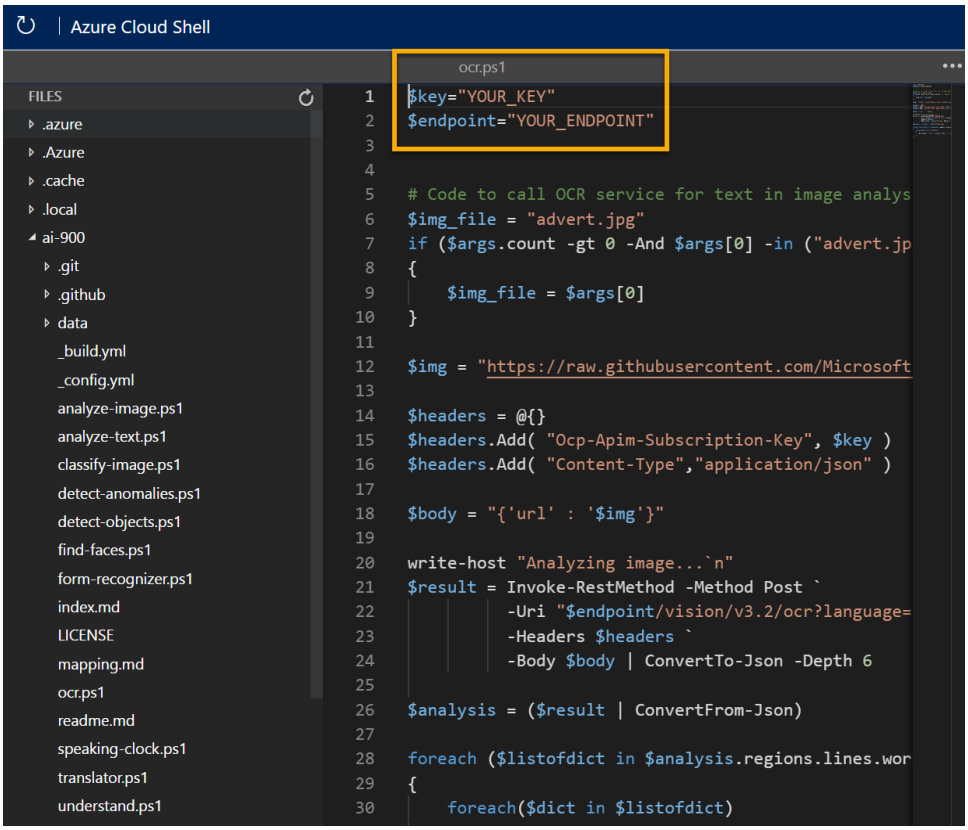
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **ocr.ps1**. This file contains some code that uses the Computer Vision service to detect and analyze text in an image, as shown here:



4. Don't worry too much about the details of the code, the important thing is that it needs the endpoint URL and either of the keys for your Cognitive Services resource. Copy these from the **Keys and Endpoints** page for your resource from the Azure portal and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_ENDPOINT** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

   After pasting the key and endpoint values, the first two lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $endpoint="https..."
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**. Now that you've set up the key and endpoint, you can use your Cognitive Services resource to extract text from an image.

   Let's use the **OCR** API, which enables you to synchronously analyze an image and read any text it contains. In this case, you have an advertising image for the fictional Northwind Traders retail company that includes some text.

   The sample client application will analyze the following image:

6. In the PowerShell pane, enter the following commands to run the code to read the text:

   Copy

   ```
   cd ai-900
   ./ocr.ps1 advert.jpg
   ```

7. Review the details found in the image. The text found in the image is organized into a hierarchical structure of regions, lines, and words, and the code reads these to retrieve the results.

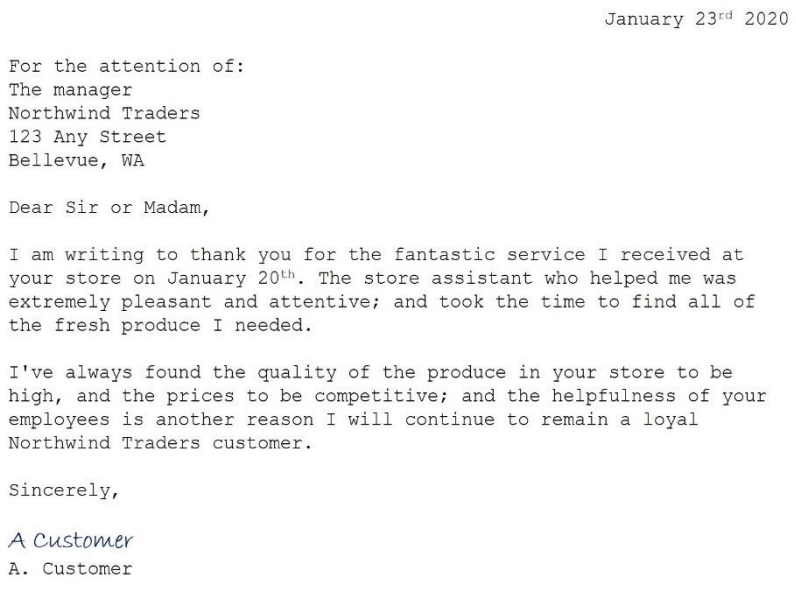
   Note that the location of text is indicated by the top-left coordinates, and the width and height of a *bounding box*, as shown here:

8. Now let's try another image:

   To analyze the second image, enter the following command:

   Copy

   ```
   ./ocr.ps1 letter.jpg
   ```


January 23rd 2020

For the attention of:
The manager
Northwind Traders
123 Any Street
Bellevue, WA

Dear Sir or Madam,

I am writing to thank you for the fantastic service I received at your store on January 20th. The store assistant who helped me was extremely pleasant and attentive; and took the time to find all of the fresh produce I needed.

I've always found the quality of the produce in your store to be high, and the prices to be competitive; and the helpfulness of your employees is another reason I will continue to remain a loyal Northwind Traders customer.

Sincerely,

A Customer
A. Customer

9. Review the results of the analysis for the second image. It should also return the text and bounding boxes of the text.

Learn more

This simple app shows only some of the OCR capabilities of the Computer Vision service. To learn more about what you can do with this service, see the OCR page.

Knowledge check

1. You want to extract text from images and then use the Text Analytics service to analyze the text. You want developers to require only one key and endpoint to access all of your services. What kind of resource should you create in your Azure subscription?

- Computer Vision

Incorrect. This resource type only supports image analysis. A separate resource (with its own key and endpoint) would be required for text analytics.

- Cognitive Services

Correct. A Cognitive Services resource support both Computer Vision for text extraction, and Text Analytics for text analysis.

- Text Analytics

2. You plan to use the Computer Vision service to read text in a large PDF document. Which API should you use?

- The Read API

Correct: Not only is the Read API better suited for larger images but it runs asynchronously so it will not block your application while it is running

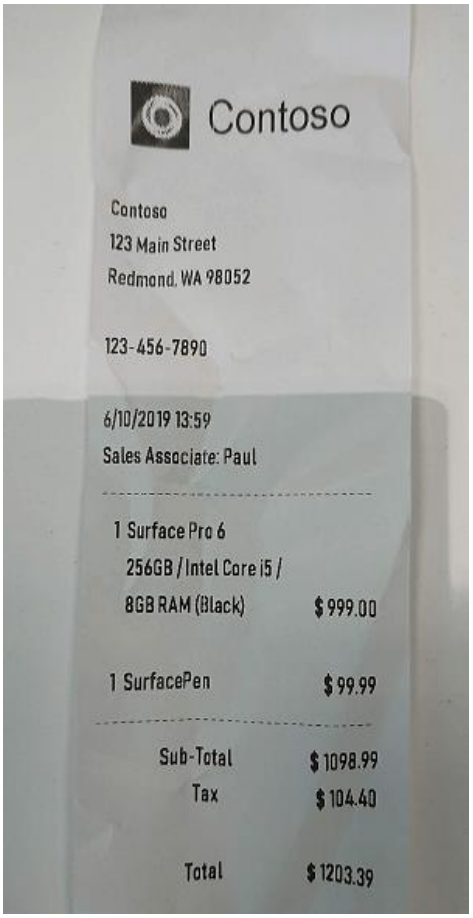
- The OCR API

Incorrect: The OCR API is not optimized for large documents and doesn't support PDF format.

- The Recognize Text API

Part 3/6_6/6

Analyze receipts with the Form Recognizer service



Processing invoices and receipts is a common task in many business scenarios. Increasingly, organizations are turning to artificial intelligence (AI) to automate data extraction from scanned receipts.

Introduction

A common problem in many organizations is the need to process receipt or invoice data. For example, a company might require expense claims to be submitted electronically with scanned receipts, or invoices might need to be digitized and routed to the correct accounts department. Typically after a document is scanned, someone will still need to manually enter the extracted text into a database.

Increasingly, organizations with large volumes of receipts and invoices to process are looking for artificial intelligence (AI) solutions that can not only extract the text data from receipts, but also intelligently interpret the information they contain.

Azure's Form Recognizer service can solve for this issue by digitizing fields from forms using optical character recognition (OCR). Azure's OCR technologies extract the contents and structure from forms, such as key, value pairs (eg. Quantity: 3).

Using the Form Recognizer service, we can input an image of a receipt like the one above, and return useful information that might be required for an expense claim, including:

- The name, address, and telephone number of the merchant.

- The date and time of the purchase.
- The quantity and price of each item purchased.
- The subtotal, tax, and total amounts.

In this module you will:

- Identify suitable Azure services for processing receipts
- Provision a Form Recognizer resource
- Use a Form Recognizer resource to extract information from a receipt

---

Get started with receipt analysis on Azure

The **Form Recognizer** in Azure provides intelligent form processing capabilities that you can use to automate the processing of data in documents such as forms, invoices, and receipts. It combines state-of-the-art optical character recognition (OCR) with predictive models that can interpret form data by:

- Matching field names to values.
- Processing tables of data.
- Identifying specific types of field, such as dates, telephone numbers, addresses, totals, and others.

Form Recognizer supports automated document processing through:

- **A pre-built receipt model** that is provided out-of-the-box, and is trained to recognize and extract data from sales receipts.
- **Custom models**, which enable you to extract what are known as key/value pairs and table data from forms. Custom models are trained using your own data, which helps to tailor this model to your specific forms. Starting with only five samples of your forms, you can train the custom model. After the first training exercise, you can evaluate the results and consider if you need to add more samples and re-train.

Note

The next hands-on exercise will only step through **a pre-built receipt model**. If you would like to train a **custom model** you can refer to the **Form Recognizer documentation** for quickstarts.

Azure resources to access Form Recognizer services

To use the Form recognizer, you need to either create a **Form Recognizer** resource or a **Cognitive Services** resource in your Azure subscription. Both resource types give access to the Form Recognizer service.

After the resource has been created, you can create client applications that use its **key** and **endpoint** to connect submit forms for analysis.

Using the pre-built receipt model

Currently the pre-built receipt model is designed to recognize common receipts, in English, that are common to the USA. Examples are receipts used at restaurants, retail locations, and gas stations. The model is able to extract key information from the receipt slip:

- time of transaction
- date of transaction
- merchant information
- taxes paid
- receipt totals
- other pertinent information that may be present on the receipt
- all text on the receipt is recognized and returned as well

Use the following guidelines to get the best results when using a custom model.

- Images must be JPEG, PNG, BMP, PDF, or TIFF formats
- File size must be less than 50 MB

- Image size between 50 x 50 pixels and 10000 x 10000 pixels
- For PDF documents, no larger than 17 inches x 17 inches

There is a free tier subscription plan for the receipt model along with paid subscriptions. For the free tier, only the first two pages will be processed when passing in PDF or TIFF formatted documents.

Exercise - Analyze receipts with Form Recognizer

In the artificial intelligence (AI) field of computer vision, optical character recognition (OCR) is commonly used to read printed or handwritten documents. Often, the text is simply extracted from the documents into a format that can be used for further processing or analysis.

A more advanced OCR scenario is the extraction of information from forms, such as purchase orders or invoices, with a semantic understanding of what the fields in the form represent. The **Form Recognizer** service is specifically designed for this kind of AI problem.

Form Recognizer uses machine learning models trained to extract text from images of invoices, receipts, and more. While other computer vision models can capture text, Form Recognizer also captures the structure of the text, such as key/value pairs and information in tables. This way, instead of having to manually type in entries from a form into a database, you can automatically capture the relationships between text from the original file.

To test the capabilities of the Form Recognizer service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Form Recognizer service by creating either a **Form Recognizer** resource or a **Cognitive Services** resource.

If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: *Choose any available region*:
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

Run Cloud Shell

To test the capabilities of the Form Recognizer service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.

4. Make sure the the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.
5. Wait for PowerShell to start. You should see the following screen in the Azure portal:

Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the Form Recognizer service.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

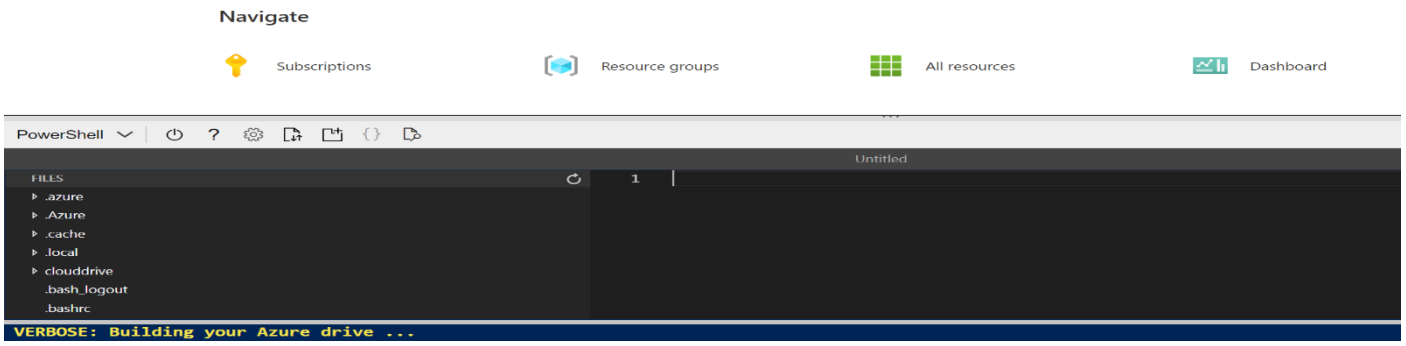
   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

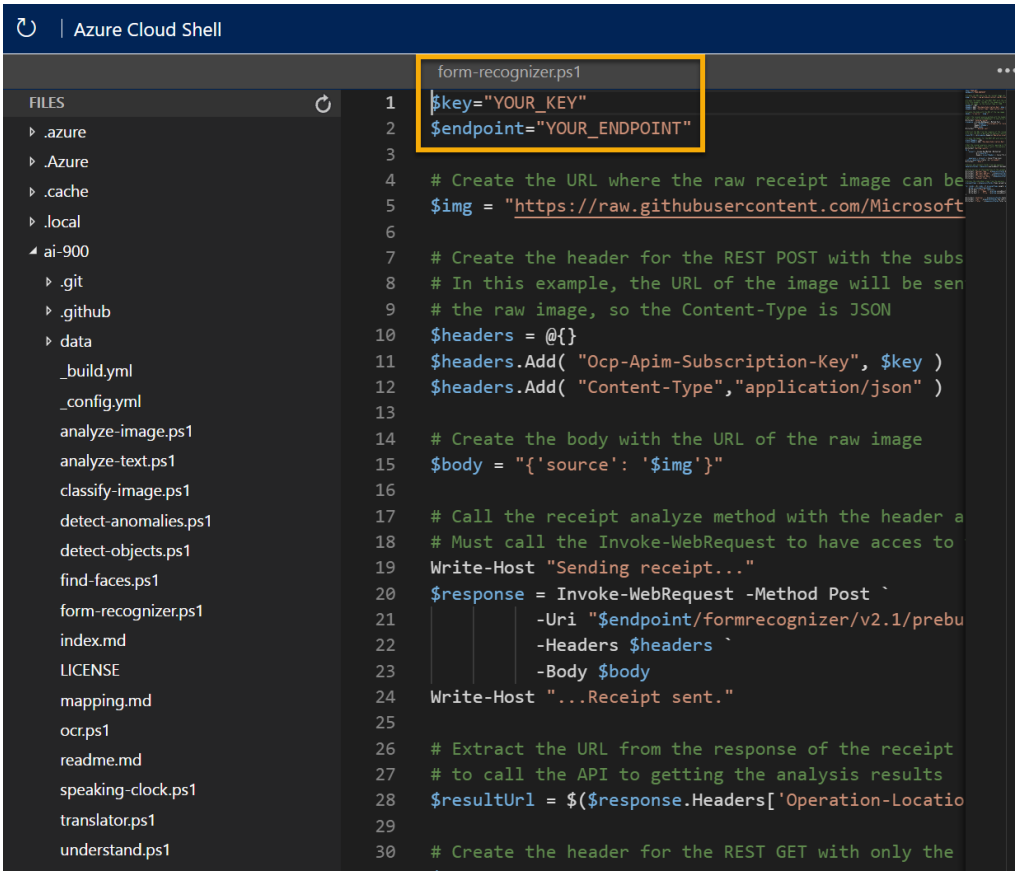
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **form-recognizer.ps1**. This file contains some code that uses the Form Recognizer service to analyze the fields in a receipt, as shown here:



4. Don't worry too much about the details of the code, the important thing is that it needs the endpoint URL and either of the keys for your Cognitive Services resource. Copy these from the **Keys and Endpoints** page for your resource from the Azure portal and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_ENDPOINT** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

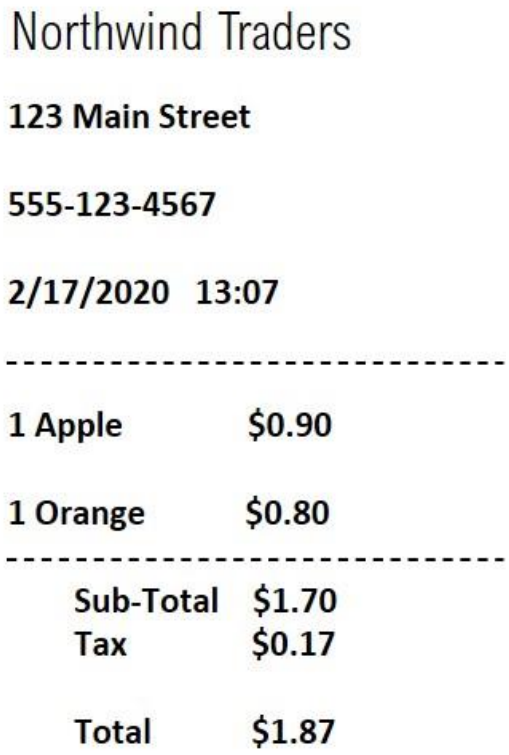
   After pasting the key and endpoint values, the first two lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $endpoint="https..."
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**. Now that you've set up the key and endpoint, you can use your resource to analyze fields from a receipt. In this case, you'll use the Form Recognizer's built-in model to analyze a receipt for the fictional Northwind Traders retail company.

   The sample client application will analyze the following image:



6. In the PowerShell pane, enter the following commands to run the code to read the text:

   Copy

   ```
   cd ai-900
   ./form-recognizer.ps1
   ```

7. Review the returned results. See that Form Recognizer is able to interpret the data in the form, correctly identifying the merchant address and phone number, and the transaction date and time, as well as the line items, subtotal, tax, and total amounts.

Learn more

This simple app shows only some of the Form Recognizer capabilities of the Computer Vision service. To learn more about what you can do with this service, see the Form Recognizer page.

Knowledge check

1. You plan to use the Form Recognizer pre-built receipt model. Which kind of Azure resource should you create?

- Computer Vision resource
- Form Recognizer or Cognitive Services resource

Correct: Both the Form Recognizer resource and Cognitive Services resource provide access to the Form Recognizer service

- Only Form Recognizer resource

Incorrect: The Cognitive Services resource also includes the Form Recognizer service

2. You are using the Form Recognizer service to analyze receipts that you have scanned into JPG format images. What is the maximum file size of JPG file you can submit to the pre-built receipt model?

- 2 MB

Incorrect: The maximum file size for the pre-built receipt model is 50 MB

- 50 MB

Correct: The maximum file size for the pre-built receipt model is 50 MB

- 200 MB

Summary

The pre-built receipt model is a part of the Form Recognizer service. It is optimized to extract information from receipts. Capable of reading from restaurant, retail, and gas company receipts, it can help automate expense workflows for businesses by extracting key data from the receipts.

You can find out more about reading text with the Computer Vision service in the service documentation.

Clean-up

It's a good idea at the end of a project to identify whether you still need the resources you created. Resources left running can cost you money.

If you are continuing on to other modules in this learning path you can keep your resources for use in other labs.

If you have finished learning, you can delete the resource group or individual resources from your Azure subscription:

1. In the Azure portal, in the **Resource groups** page, open the resource group you specified when creating your resource.
2. Click **Delete resource group**, type the resource group name to confirm you want to delete it, and select **Delete**. You can also choose to delete individual resources by selecting the resource(s), clicking on the three dots to see more options, and clicking **Delete**.

Module complete:

Part 4/6

Microsoft Azure AI Fundamentals: Explore natural language processing

Natural language processing supports applications that can see, hear, speak with, and understand users. Using text analytics, translation, and language understanding services, Microsoft Azure makes it easy to build applications that support natural language.

Part 4/6_1/5

Analyze text with the Language service

Explore text mining and text analysis with the Language service's Natural Language Processing (NLP) features, which include sentiment analysis, key phrase extraction, named entity recognition, and language detection.

Introduction

Analyzing text is a process where you evaluate different aspects of a document or phrase, in order to gain insights into the content of that text. For the most part, humans are able to read some text and understand the meaning behind it. Even without considering grammar rules for the language the text is written in, specific insights can be identified in the text.

As an example, you might read some text and identify some key phrases that indicate the main talking points of the text. You might also recognize names of people or well-known landmarks such as the Eiffel Tower. Although difficult at times, you might also be able to get a sense for how the person was feeling when they wrote the text, also commonly known as sentiment.

Text Analytics Techniques

Text analytics is a process where an artificial intelligence (AI) algorithm, running on a computer, evaluates these same attributes in text, to determine specific insights. A person will typically rely on their own experiences and knowledge to achieve the insights. A computer must be provided with similar knowledge to be able to perform the task. There are some commonly used techniques that can be used to build software to analyze text, including:

- Statistical analysis of terms used in the text. For example, removing common "stop words" (words like "the" or "a", which reveal little semantic information about the text), and performing *frequency analysis* of the remaining words (counting how often each word appears) can provide clues about the main subject of the text.
- Extending frequency analysis to multi-term phrases, commonly known as *N-grams* (a two-word phrase is a *bi-gram*, a three-word phrase is a *tri-gram*, and so on).
- Applying *stemming* or *lemmatization* algorithms to normalize words before counting them - for example, so that words like "power", "powered", and "powerful" are interpreted as being the same word.
- Applying linguistic structure rules to analyze sentences - for example, breaking down sentences into tree-like structures such as a *noun phrase*, which itself contains *nouns*, *verbs*, *adjectives*, and so on.
- Encoding words or terms as numeric features that can be used to train a machine learning model. For example, to classify a text document based on the terms it contains. This technique is often used to perform *sentiment analysis*, in which a document is classified as positive or negative.
- Creating *vectorized* models that capture semantic relationships between words by assigning them to locations in n-dimensional space. This modeling technique might, for example, assign values to the words "flower" and "plant" that locate them close to one another, while "skateboard" might be given a value that positions it much further away.

While these techniques can be used to great effect, programming them can be complex. In Microsoft Azure, the **Language** cognitive service can help simplify application development by using pre-trained models that can:

- Determine the language of a document or text (for example, French or English).
- Perform sentiment analysis on text to determine a positive or negative sentiment.
- Extract key phrases from text that might indicate its main talking points.
- Identify and categorize entities in the text. Entities can be people, places, organizations, or even everyday items such as dates, times, quantities, and so on.

In this module, you'll explore some of these capabilities and gain an understanding of how you might apply them to applications such as:

- A social media feed analyzer to detect sentiment around a political campaign or a product in market.
- A document search application that extracts key phrases to help summarize the main subject matter of documents in a catalog.
- A tool to extract brand information or company names from documents or other text for identification purposes.

These examples are just a small sample of the many areas that the Language service can help with text analytics.

Get started with text analysis

The Language service is a part of the Azure Cognitive Services offerings that can perform advanced natural language processing over raw text.

Azure resources for the Language service

To use the Language service in an application, you must provision an appropriate resource in your Azure subscription. You can choose to provision either of the following types of resource:

- A **Language** resource - choose this resource type if you only plan to use natural language processing services, or if you want to manage access and billing for the resource separately from other services.
- A **Cognitive Services** resource - choose this resource type if you plan to use the Language service in combination with other cognitive services, and you want to manage access and billing for these services together.

Language detection

Use the language detection capability of the Language service to identify the language in which text is written. You can submit multiple documents at a time for analysis. For each document submitted to it, the service will detect:

- The language name (for example "English").
- The ISO 6391 language code (for example, "en").
- A score indicating a level of confidence in the language detection.

For example, consider a scenario where you own and operate a restaurant where customers can complete surveys and provide feedback on the food, the service, staff, and so on. Suppose you have received the following reviews from customers:

Review 1: "*A fantastic place for lunch. The soup was delicious.*"

Review 2: "*Comida maravillosa y gran servicio.*"

Review 3: "*The croque monsieur avec frites was terrific. Bon appetit!*"

You can use the text analytics capabilities in the Language service to detect the language for each of these reviews; and it might respond with the following results:

Document	Language Name	ISO 6391 Code	Score
Review 1	English	en	1.0
Review 2	Spanish	es	1.0
Review 3	English	en	0.9

Notice that the language detected for review 3 is English, despite the text containing a mix of English and French. The language detection service will focus on the ***predominant*** language in the text. The service uses an algorithm to determine the predominant language, such as length of phrases or total amount of text for the language compared to other languages in the text. The predominant language will be the value returned, along with the language code. The confidence score may be less than 1 as a result of the mixed language text.

Ambiguous or mixed language content

There may be text that is ambiguous in nature, or that has mixed language content. These situations can present a challenge to the service. An ambiguous content example would be a case where the document contains limited text, or only punctuation. For example, using the service to analyze the text ":-)", results in a value of **unknown** for the language name and the language identifier, and a score of **NaN** (which is used to indicate *not a number*).

Sentiment analysis

The text analytics capabilities in the Language service can evaluate text and return sentiment scores and labels for each sentence. This capability is useful for detecting positive and negative sentiment in social media, customer reviews, discussion forums and more.

Using the pre-built machine learning classification model, the service evaluates the text and returns a sentiment score in the range of 0 to 1, with values closer to 1 being a positive sentiment. Scores that are close to the middle of the range (0.5) are considered neutral or indeterminate.

For example, the following two restaurant reviews could be analyzed for sentiment:

"*We had dinner at this restaurant last night and the first thing I noticed was how courteous the staff was. We were greeted in a friendly manner and taken to our table right away. The table was clean, the chairs were comfortable, and the food was amazing.*"

and

"*Our dining experience at this restaurant was one of the worst I've ever had. The service was slow, and the food was awful. I'll never eat at this establishment again.*"

The sentiment score for the first review might be around 0.9, indicating a positive sentiment; while the score for the second review might be closer to 0.1, indicating a negative sentiment.

Indeterminate sentiment

A score of 0.5 might indicate that the sentiment of the text is indeterminate, and could result from text that does not have sufficient context to discern a sentiment or insufficient phrasing. For example, a list of words in a sentence that has no structure, could result in an indeterminate score. Another example where a score may be 0.5 is in the case where the wrong language code was used. A language code (such as "en" for English, or "fr" for French) is used to inform the service which language the text is in. If you pass text in French but tell the service the language code is **en** for English, the service will return a score of precisely 0.5.

Key phrase extraction

Key phrase extraction is the concept of evaluating the text of a document, or documents, and then identifying the main talking points of the document(s). Consider the restaurant scenario discussed previously. Depending on the volume of surveys that you have collected, it can take a long time to read through the reviews. Instead, you can use the key phrase extraction capabilities of the Language service to summarize the main points.

You might receive a review such as:

"*We had dinner here for a birthday celebration and had a fantastic experience. We were greeted by a friendly hostess and taken to our table right away. The ambiance was relaxed, the food was amazing, and service was terrific. If you like great food and attentive service, you should try this place.*"

Key phrase extraction can provide some context to this review by extracting the following phrases:

- attentive service
- great food
- birthday celebration
- fantastic experience
- table
- friendly hostess
- dinner
- ambiance
- place

Not only can you use sentiment analysis to determine that this review is positive, you can use the key phrases to identify important elements of the review.

Entity recognition

You can provide the Language service with unstructured text and it will return a list of *entities* in the text that it recognizes. The service can also provide links to more information about that entity on the web. An entity is

essentially an item of a particular type or a category; and in some cases, subtype, such as those as shown in the following table.

Type	SubType	Example
Person		"Bill Gates", "John"
Location		"Paris", "New York"
Organization		"Microsoft"
Quantity	Number	"6" or "six"
Quantity	Percentage	"25%" or "fifty percent"
Quantity	Ordinal	"1st" or "first"
Quantity	Age	"90 day old" or "30 years old"
Quantity	Currency	"10.99"
Quantity	Dimension	"10 miles", "40 cm"
Quantity	Temperature	"45 degrees"
DateTime		"6:30PM February 4, 2012"
DateTime	Date	"May 2nd, 2017" or "05/02/2017"
DateTime	Time	"8am" or "8:00"
DateTime	DateRange	"May 2nd to May 5th"
DateTime	TimeRange	"6pm to 7pm"
DateTime	Duration	"1 minute and 45 seconds"
DateTime	Set	"every Tuesday"
URL		"https://www.bing.com"
Email		"support@microsoft.com"
US-based Phone Number		"(312) 555-0176"
IP Address		"10.0.1.125"

The service also supports *entity linking* to help disambiguate entities by linking to a specific reference. For recognized entities, the service returns a URL for a relevant *Wikipedia* article.

For example, suppose you use the Language service to detect entities in the following restaurant review extract:

"*I ate at the restaurant in Seattle last week.*"

Entity	Type	SubType	Wikipedia URL
Seattle	Location		https://en.wikipedia.org/wiki/Seattle
last week	DateTime	DateRange	

Exercise - Analyze text

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) that deals with written and spoken language. You can use NLP to build solutions that extracting semantic meaning from text or speech, or that formulate meaningful responses in natural language.

Microsoft Azure *Cognitive Services* includes the text analytics capabilities in the *Language* service, which provides some out-of-the-box NLP capabilities, including the identification of key phrases in text, and the classification of text based on sentiment.

For example, suppose the fictional *Margie's Travel* organization encourages customers to submit reviews for hotel stays. You could use the Language service to summarize the reviews by extracting key phrases, determine which

reviews are positive and which are negative, or analyze the review text for mentions of known entities such as locations or people.

To test the capabilities of the Language service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Language service by creating either a **Language** resource or a **Cognitive Services** resource.

If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Select the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: *Choose any available region*:
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource.

Get the key and endpoint for your Cognitive Services resource

1. Wait for deployment to complete. Then go to your Cognitive Services resource, and on the **Overview** page, select the link to manage the keys for the service. You will need the endpoint and keys to connect to your Cognitive Services resource from client applications.
2. View the **Keys and Endpoint** page for your resource. You will need the **key** and **endpoint** to connect from client applications.

Run Cloud Shell

To test the text analytics capabilities of the Language service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.
4. Make sure the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.
5. Wait for PowerShell to start. You should see the following screen in the Azure portal:

Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the Language service.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

    Copy

    ```
    git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
    ```

    Tip

If you already used this command in another lab to clone the *ai-900* repository, you can skip this step.

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:

3. In the **Files** pane on the left, expand **ai-900** and select **analyze-text.ps1**. This file contains some code that uses the Language service:

4. Don't worry too much about the details of the code. In the Azure portal, navigate to your Cognitive Services resource. Then select the **Keys and Endpoints** page on the left hand pane. Copy the key and endpoint from the page and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_ENDPOINT** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

   After replacing the key and endpoint values, the first lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $endpoint="https..."
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   The sample client application will use Cognitive Services' Language service to detect language, extract key phrases, determine sentiment, and extract known entities for reviews.

6. In the Cloud Shell, enter the following command to run the code:

   Copy

   ```
   cd ai-900
   ./analyze-text.ps1 review1.txt
   ```

   You will be reviewing this text:

   Good Hotel and staff The Royal Hotel, London, UK 3/2/2018 Clean rooms, good service, great location near Buckingham Palace and Westminster Abbey, and so on. We thoroughly enjoyed our stay. The courtyard is very peaceful and we went to a restaurant which is part of the same group and is Indian (West coast so plenty of fish) with a Michelin Star. We had the taster menu which was fabulous. The rooms were very well appointed with a kitchen, lounge, bedroom and enormous bathroom. Thoroughly recommended.

7. Review the output.

8. In the PowerShell pane, enter the following command to run the code:

   Copy

   ```
   ./analyze-text.ps1 review2.txt
   ```

   You will be reviewing this text:

   Tired hotel with poor service The Royal Hotel, London, United Kingdom 5/6/2018 This is a old hotel (has been around since 1950's) and the room furnishings are average - becoming a bit old now and require changing. The internet didn't work and had to come to one of their office rooms to check in for my flight home. The website says it's close to the British Museum, but it's too far to walk.

9. Review the output.

10. In the PowerShell pane, enter the following command to run the code:

    Copy

```
./analyze-text.ps1 review3.txt
```

You will be reviewing this text:

Good location and helpful staff, but on a busy road. The Lombard Hotel, San Francisco, USA 8/16/2018 We stayed here in August after reading reviews. We were very pleased with location, just behind Chestnut Street, a cosmopolitan and trendy area with plenty of restaurants to choose from. The Marina district was lovely to wander through, very interesting houses. Make sure to walk to the San Francisco Museum of Fine Arts and the Marina to get a good view of Golden Gate bridge and the city. On a bus route and easy to get into centre. Rooms were clean with plenty of room and staff were friendly and helpful. The only down side was the noise from Lombard Street so ask to have a room furthest away from traffic noise.

11. Review the output.
12. In the PowerShell pane, enter the following command to run the code:

    Copy

    ```
    ./analyze-text.ps1 review4.txt
    ```

    You will be reviewing this text:

    Very noisy and rooms are tiny The Lombard Hotel, San Francisco, USA 9/5/2018 Hotel is located on Lombard street which is a very busy SIX lane street directly off the Golden Gate Bridge. Traffic from early morning until late at night especially on weekends. Noise would not be so bad if rooms were better insulated but they are not. Had to put cotton balls in my ears to be able to sleep--was too tired to enjoy the city the next day. Rooms are TINY. I picked the room because it had two queen size beds--but the room barely had space to fit them. With family of four in the room it was tight. With all that said, rooms are clean and they've made an effort to update them. The hotel is in Marina district with lots of good places to eat, within walking distance to Presidio. May be good hotel for young stay-up-late adults on a budget
13. Review the output.

Learn more

This simple app shows only some of the capabilities of the Language service. To learn more about what you can do with this service, see the Language service page.

Knowledge check

1. You want to use the Language service to determine the key talking points in a text document. Which feature of the service should you use?

- Sentiment analysis
- Key phrase extraction

Correct. Key phrases can be used to identify the main talking points in a text document.

- Entity detection

2. You use the Language service to perform sentiment analysis on a document, and a score of 0.99 is returned. What does this score indicate about the document sentiment?

- The document is positive.

Correct. Score values closer to 1 indicated a more positive sentiment where scores closer to 0 indicated negative sentiment.

- The document is neutral.
- The document is negative.

3. When might you see **NaN** returned for a score in Language Detection?

- When the score calculated by the service is outside the range of 0 to 1
- When the predominant language in the text is mixed with other languages

Incorrect. The service will return a score of less than 1 for mixed language text, using the predominant language in the text.

- When the language is ambiguous

Correct. The service will return NaN when it cannot determine the language in the provided text.

Part 4/6_2/5

Recognize and synthesize speech

Learn how to recognize and synthesize speech by using Azure Cognitive Services.

Introduction

Increasingly, we expect artificial intelligence (AI) solutions to accept vocal commands and provide spoken responses. Consider the growing number of home and auto systems that you can control by speaking to them - issuing commands such as "turn off the lights", and soliciting verbal answers to questions such as "will it rain today?"

To enable this kind of interaction, the AI system must support two capabilities:

- **Speech recognition** - the ability to detect and interpret spoken input.
- **Speech synthesis** - the ability to generate spoken output.

Speech recognition

Speech recognition is concerned with taking the spoken word and converting it into data that can be processed - often by transcribing it into a text representation. The spoken words can be in the form of a recorded voice in an audio file, or live audio from a microphone. Speech patterns are analyzed in the audio to determine recognizable patterns that are mapped to words. To accomplish this feat, the software typically uses multiple types of models, including:

- An *acoustic* model that converts the audio signal into phonemes (representations of specific sounds).
- A *language* model that maps phonemes to words, usually using a statistical algorithm that predicts the most probable sequence of words based on the phonemes.

The recognized words are typically converted to text, which you can use for various purposes, such as.

- Providing closed captions for recorded or live videos
- Creating a transcript of a phone call or meeting
- Automated note dictation
- Determining intended user input for further processing

Speech synthesis

Speech synthesis is in many respects the reverse of speech recognition. It is concerned with vocalizing data, usually by converting text to speech. A speech synthesis solution typically requires the following information:

- The text to be spoken.
- The voice to be used to vocalize the speech.

To synthesize speech, the system typically *tokenizes* the text to break it down into individual words, and assigns phonetic sounds to each word. It then breaks the phonetic transcription into *prosodic* units (such as phrases, clauses, or sentences) to create phonemes that will be converted to audio format. These phonemes are then synthesized as audio by applying a voice, which will determine parameters such as pitch and timbre; and generating an audio wave form that can be output to a speaker or written to a file.

You can use the output of speech synthesis for many purposes, including:

- Generating spoken responses to user input.

- Creating voice menus for telephone systems.
- Reading email or text messages aloud in hands-free scenarios.
- Broadcasting announcements in public locations, such as railway stations or airports.

Get started with speech on Azure

Microsoft Azure offers both speech recognition and speech synthesis capabilities through the **Speech** cognitive service, which includes the following application programming interfaces (APIs):

- The **Speech-to-Text** API
- The **Text-to-Speech** API

Azure resources for the Speech service

To use the Speech service in an application, you must create an appropriate resource in your Azure subscription. You can choose to create either of the following types of resource:

- A **Speech** resource - choose this resource type if you only plan to use the Speech service, or if you want to manage access and billing for the resource separately from other services.
- A **Cognitive Services** resource - choose this resource type if you plan to use the Speech service in combination with other cognitive services, and you want to manage access and billing for these services together.

The speech-to-text API

You can use the speech-to-text API to perform real-time or batch transcription of audio into a text format. The audio source for transcription can be a real-time audio stream from a microphone or an audio file.

The model that is used by the speech-to-text API, is based on the Universal Language Model that was trained by Microsoft. The data for the model is Microsoft-owned and deployed to Microsoft Azure. The model is optimized for two scenarios, conversational and dictation. You can also create and train your own custom models including acoustics, language, and pronunciation if the pre-built models from Microsoft do not provide what you need.

Real-time transcription

Real-time speech-to-text allows you to transcribe text in audio streams. You can use real-time transcription for presentations, demos, or any other scenario where a person is speaking.

In order for real-time transcription to work, your application will need to be listening for incoming audio from a microphone, or other audio input source such as an audio file. Your application code streams the audio to the service, which returns the transcribed text.

Batch transcription

Not all speech-to-text scenarios are real time. You may have audio recordings stored on a file share, a remote server, or even on Azure storage. You can point to audio files with a shared access signature (SAS) URI and asynchronously receive transcription results.

Batch transcription should be run in an asynchronous manner because the batch jobs are scheduled on a *best-effort basis*. Normally a job will start executing within minutes of the request but there is no estimate for when a job changes into the running state.

The text-to-speech API

The text-to-speech API enables you to convert text input to audible speech, which can either be played directly through a computer speaker or written to an audio file.

Speech synthesis voices

When you use the text-to-speech API, you can specify the voice to be used to vocalize the text. This capability offers you the flexibility to personalize your speech synthesis solution and give it a specific character.

The service includes multiple pre-defined voices with support for multiple languages and regional pronunciation, including *standard* voices as well as *neural* voices that leverage *neural networks* to overcome common limitations in speech synthesis with regard to intonation, resulting in a more natural sounding voice. You can also develop custom voices and use them with the text-to-speech API

Supported Languages

Both the speech-to-text and text-to-speech APIs support a variety of languages. Use the links below to find details about the supported languages:

- Speech-to-text languages.
- Text-to-speech languages.

Exercise - Use the Speech service

To build software that can interpret audible speech and respond appropriately, you can use the **Speech** cognitive service, which provides a simple way to transcribe spoken language into text and vice-versa.

For example, suppose you want to create a smart device that can respond verbally to spoken questions, such as "What time is it?" The response should be the local time.

To test the capabilities of the Speech service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Speech service by creating either a **Speech** resource or a **Cognitive Services** resource.

If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: *Choose any available region*:
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource.

Get the Key and Location for your Cognitive Services resource

1. Wait for deployment to complete. Then go to your Cognitive Services resource, and on the **Overview** page, click the link to manage the keys for the service. You will need the endpoint and keys to connect to your Cognitive Services resource from client applications.
2. View the **Keys and Endpoint** page for your resource. You will need the **location/region** and **key** to connect from client applications.

Run Cloud Shell

To test the capabilities of the Speech service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.
4. Make sure the the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.

5. Wait for PowerShell to start. You should see the following screen in the Azure portal:

Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the Speech service.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

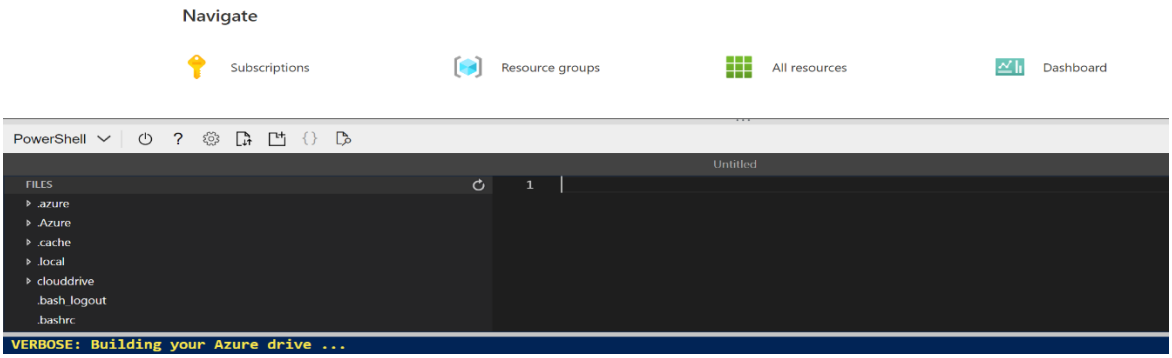
   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

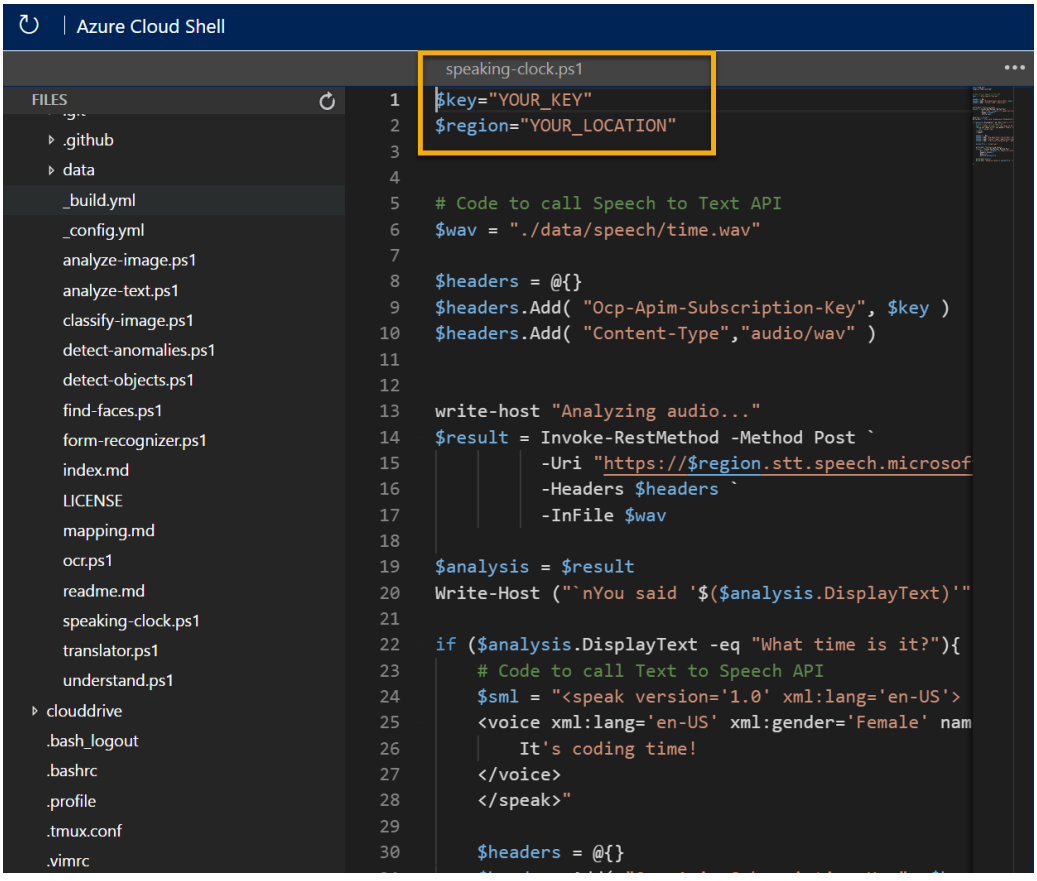
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **speaking-clock.ps1**. This file contains some code that uses the Speech service to recognize and synthesize speech:



4. Don't worry too much about the details of the code, the important thing is that it needs the region/location and either of the keys for your Cognitive Services resource. Copy these from the **Keys and Endpoints** page for your resource from the Azure portal and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_LOCATION** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

   After pasting the key and region/location values, the first lines of code should look similar to this:

   PowerShellCopy

   ```
   $key = "1a2b3c4d5e6f7g8h9i0j...."
   $region="somelocation"
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   The sample client application will use your Speech service to transcribe spoken input and synthesize an appropriate spoken response. A real application would accept the input from a microphone and send the response to a speaker, but in this simple example, we'll use pre-recorded input in a file and save the response as another file.

   Use the video player below to hear the input audio the application will process:

6. In the PowerShell pane, enter the following command to run the code:

   Copy

   ```
   cd ai-900
   ./speaking-clock.ps1
   ```

7. Review the output, which should have successfully recognized the text "What time is it?" and saved an appropriate response in a file named *output.wav*.

   Use the following video player to hear the spoken output generated by the application:

Learn more

This simple app shows only some of the capabilities of the Speech service. To learn more about what you can do with this service, see the Speech page.

Part 4/6_3/5

Translate text and speech

Automated translation capabilities in an AI solution enable closer collaboration by removing language barriers.

Introduction

As organizations and individuals increasingly need to collaborate with people in other cultures and geographic locations, the removal of language barriers has become a significant problem.

One solution is to find bilingual, or even multilingual, people to translate between languages. However the scarcity of such skills, and the number of possible language combinations can make this approach difficult to scale. Increasingly, automated translation, sometimes known as *machine translation*, is being employed to solve this problem.

Literal and semantic translation

Early attempts at machine translation applied *literal* translations. A literal translation is where each word is translated to the corresponding word in the target language. This approach presents some issues. For one case, there may not be an equivalent word in the target language. Another case is where literal translation can change the meaning of the phrase or not get the context correct.

For example, the French phrase "*éteindre la lumière*" can be translated to English as "*turn off the light*". However, in French you might also say "*fermer la lumiere*" to mean the same thing. The French verb *fermer* literally means

to "*close*", so a literal translation based only on the words would indicate, in English, "*close the light*"; which for the average English speaker, doesn't really make sense, so to be useful, a translation service should take into account the semantic context and return an English translation of "*turn off the light*".

Artificial intelligence systems must be able to understand, not only the words, but also the *semantic* context in which they are used. In this way, the service can return a more accurate translation of the input phrase or phrases. The grammar rules, formal versus informal, and colloquialisms all need to be considered.

Text and speech translation

Text translation can be used to translate documents from one language to another, translate email communications that come from foreign governments, and even provide the ability to translate web pages on the Internet. Many times you will see a *Translate* option for posts on social media sites, or the Bing search engine can offer to translate entire web pages that are returned in search results.

Speech translation is used to translate between spoken languages, sometimes directly (speech-to-speech translation) and sometimes by translating to an intermediary text format (speech-to-text translation).

Get started with translation in Azure

Microsoft Azure provides cognitive services that support translation. Specifically, you can use the following services:

- The **Translator** service, which supports text-to-text translation.
- The **Speech** service, which enables speech-to-text and speech-to-speech translation.

Azure resources for Translator and Speech

Before you can use the Translator or Speech services, you must provision appropriate resources in your Azure subscription.

There are dedicated **Translator** and **Speech** resource types for these services, which you can use if you want to manage access and billing for each service individually.

Alternatively, you can create a **Cognitive Services** resource that provides access to both services through a single Azure resource, consolidating billing and enabling applications to access both services through a single endpoint and authentication key.

Text translation with the Translator service

The Translator service is easy to integrate in your applications, websites, tools, and solutions. The service uses a Neural Machine Translation (NMT) model for translation, which analyzes the semantic context of the text and renders a more accurate and complete translation as a result.

Translator service language support

The Translator service supports text-to-text translation between more than 60 languages. When using the service, you must specify the language you are translating ***from*** and the language you are translating ***to*** using ISO 639-1 language codes, such as *en* for English, *fr* for French, and *zh* for Chinese. Alternatively, you can specify cultural variants of languages by extending the language code with the appropriate 3166-1 cultural code - for example, *en-US* for US English, *en-GB* for British English, or *fr-CA* for Canadian French.

When using the Translator service, you can specify one ***from*** language with multiple ***to*** languages, enabling you to simultaneously translate a source document into multiple languages.

Optional Configurations

The Translator API offers some optional configuration to help you fine-tune the results that are returned, including:

- **Profanity filtering**. Without any configuration, the service will translate the input text, without filtering out profanity. Profanity levels are typically culture-specific but you can control profanity translation by either marking the translated text as profane or by omitting it in the results.
- **Selective translation**. You can tag content so that it isn't translated. For example, you may want to tag code, a brand name, or a word/phrase that doesn't make sense when localized.

Speech translation with the Speech service

The Speech service includes the following application programming interfaces (APIs):

- **Speech-to-text** - used to transcribe speech from an audio source to text format.
- **Text-to-speech** - used to generate spoken audio from a text source.
- **Speech Translation** - used to translate speech in one language to text or speech in another.

You can use the **Speech Translation** API to translate spoken audio from a streaming source, such as a microphone or audio file, and return the translation as text or an audio stream. This enables scenarios such as real-time closed captioning for a speech or simultaneous two-way translation of a spoken conversation.

Speech service language support

As with the Translator service, you can specify one source language and one or more target languages to which the source should be translated. You can translate speech into over 60 languages.

The source language must be specified using the extended language and culture code format, such as *es-US* for American Spanish. This requirement helps ensure that the source is understood properly, allowing for localized pronunciation and linguistic idioms.

The target languages must be specified using a two-character language code, such as *en* for English or *de* for German.

Exercise - Translate text and speech

One of the driving forces that has enabled human civilization to develop is the ability to communicate with one another. In most human endeavors, communication is key.

Artificial Intelligence (AI) can help simplify communication by translating text or speech between languages, helping to remove barriers to communication across countries and cultures.

To test the capabilities of the Translator service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Cognitive Services* resource

You can use the Translator service by creating either a **Translator** resource or a **Cognitive Services** resource.

If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Select the **+Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: *Choose any available region*:
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: S0
    - **I confirm I have read and understood the notices**: Selected.
3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the keys and location to connect from client applications.

Get the Key and Location for your Cognitive Services resource

1. Wait for deployment to complete. Then go to your Cognitive Services resource, and on the **Overview** page, select the link to manage the keys for the service. You will need the keys and location to connect to your Cognitive Services resource from client applications.

2. View the **Keys and Endpoint** page for your resource. You will need the **location/region** and **key** to connect from client applications.

Note

To use the Translator service you do not need to use the Cognitive Service endpoint. A global endpoint just for the Translator service is provided.

Run Cloud Shell

To test the capabilities of the Translation service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.
4. Make sure the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.
5. Wait for PowerShell to start. You should see the following screen in the Azure portal:

Configure and run a client application

Now that you have a custom model, you can run a simple client application that uses the Translation service.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

   Tip

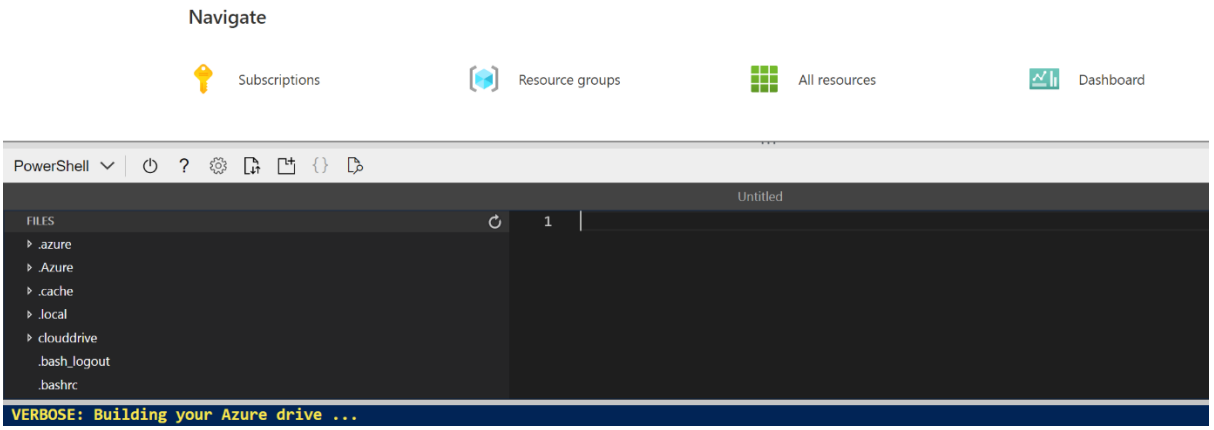
   If you already used this command in another lab to clone the *ai-900* repository, you can skip this step.

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

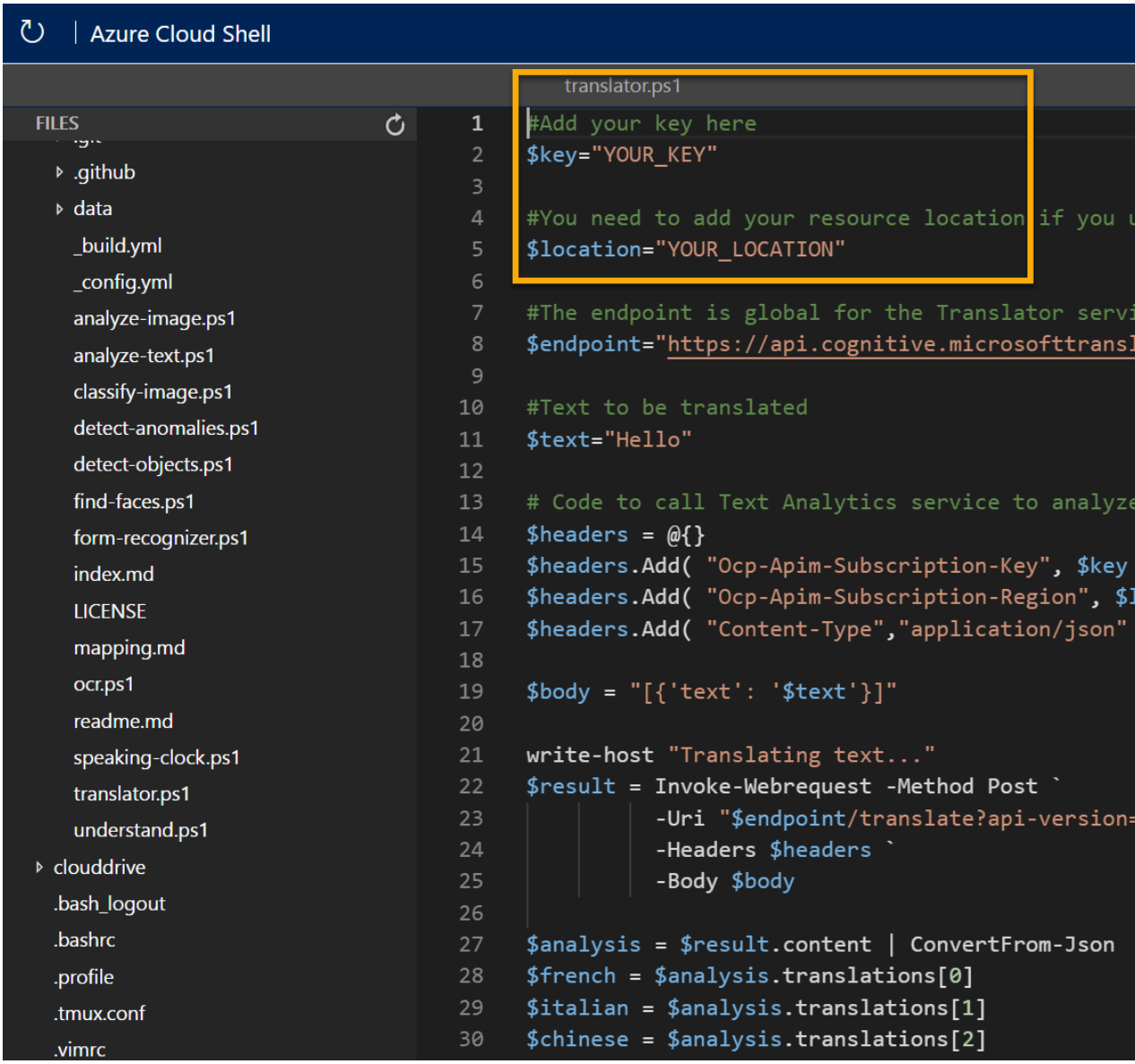
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **translator.ps1**. This file contains some code that uses the Translator service:



4. Don't worry too much about the details of the code, the important thing is that it needs the region/location and either of the keys for your Cognitive Services resource. Copy these from the **Keys and Endpoints** page for your resource from the Azure portal and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_LOCATION** placeholder values respectively.

   After pasting the key and location values, the first lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $location="somelocation"
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   The sample client application will use the Translator service to do several tasks:

   - Translate text from English into French, Italian, and Chinese.
   - Translate audio from English into text in French
     - Use the video player below to hear the input audio the application will process:

   Note

   A real application could accept the input from a microphone and send the response to a speaker, but in this simple example, we'll use pre-recorded input in an audio file.

   In the Cloud Shell pane, enter the following command to run the code:

Copy

```
cd ai-900

./translator.ps1
```

Review the output. Did you see the translation from text in English to French, Italian, and Chinese? Did you see the English audio "hello" translated into text in French?

Learn more

This simple app shows only some of the capabilities of the Translator service. To learn more about what you can do with this service, see the Translator page.

Knowledge check

1. You are developing an application that must take English input from a microphone and generate a real-time text-based transcription in Hindi. Which service should you use?

- Translator

Incorrect. The Translator service can translate from a text source, but not from an audio source.

- Speech

Correct. The Speech service can translate from audio sources to text.

- Language

2. You need to use the Translator service to translate email messages from Spanish into both English and French. What is the most efficient way to accomplish this goal?

- Make a single call to the service; specifying a "from" language of "es", a "to" language of "en", and another "to" language of "fr".

Correct. You can specify a single "from" language and multiple "to" languages.

- Make a single call to the service; specifying a "from" language of "es", and a "to" language of "en-fr".

- Make two calls to the service; one with a "from" language of "es" and a "to" language of "en", and another with a "from" language of "es" and a "to" language of "fr"

Incorrect. While this approach would work, it requires multiple calls to the service. It is not the most efficient option.

Part 4/6_4/5

Create a language model with Conversational Language Understanding

In this module, we'll introduce you to Conversational Language Understanding, and show how to create applications that understand language.

Introduction

In 1950, the British mathematician Alan Turing devised the *Imitation Game*, which has become known as the *Turing Test* and hypothesizes that if a dialog is natural enough, you may not know whether you're conversing with a human or a computer. As artificial intelligence (AI) grows ever more sophisticated, this kind of conversational interaction with applications and digital assistants is becoming more and more common, and in specific scenarios can result in human-like interactions with AI agents. Common scenarios for this kind of solution include customer support applications, reservation systems, and home automation among others.

To realize the aspiration of the imitation game, computers need not only to be able to accept language as input (either in text or audio format), but also to be able to interpret the semantic meaning of the input - in other words, *understand* what is being said.

On Microsoft Azure, conversational language understanding is supported through the **Language Service**. To work with Conversational Language Understanding, you need to take into account three core concepts: *utterances*, *entities*, and *intents*.

Utterances

An utterance is an example of something a user might say, and which your application must interpret. For example, when using a home automation system, a user might use the following utterances:

"*Switch the fan on.*"

"*Turn on the light.*"

Entities

An entity is an item to which an utterance refers. For example, **fan** and **light** in the following utterances:

"*Switch the **fan** on.*"

"*Turn on the **light**.*"

You can think of the **fan** and **light** entities as being specific instances of a general **device** entity.

Intents

An intent represents the purpose, or goal, expressed in a user's utterance. For example, for both of the previously considered utterances, the intent is to turn a device on; so in your Conversational Language Understanding application, you might define a **TurnOn** intent that is related to these utterances.

A Language Understanding application defines a model consisting of intents and entities. Utterances are used to train the model to identify the most likely intent and the entities to which it should be applied based on a given input. The home assistant application we've been considering might include multiple intents, like the following examples:

Intent	Related Utterances	Entities
Greeting	"Hello"	
	"Hi"	
	"Hey"	
	"Good morning"	
TurnOn	"Switch the fan on"	fan (device)
	"Turn the light on"	light (device)
	"Turn on the light"	light (device)
TurnOff	"Switch the fan off"	fan (device)
	"Turn the light off"	light (device)
	"Turn off the light"	light (device)
CheckWeather	"What is the weather for today?"	today (datetime)
	"Give me the weather forecast"	
	"What is the forecast for Paris?"	Paris (location)
	"What will the weather be like in Seattle tomorrow?"	Seattle (location), tomorrow (datetime)
None	"What is the meaning of life?"	
	"Is this thing on?"	

In this table there are numerous utterances used for each of the intents. The intent should be a concise way of grouping the utterance tasks. Of special interest is the ***None*** intent. You should consider always using the None intent to help handle utterances that do not map any of the utterances you have entered. The None intent is considered a fallback, and is typically used to provide a generic response to users when their requests don't match any other intent.

Tip

In a Conversational Language Understanding application, the **None** intent is created but left empty on purpose. The None intent is a required intent and can't be deleted or renamed. Fill it with utterances that are outside of your domain.

After defining the entities and intents with sample utterances in your Conversational Language Understanding application, you can train a language model to predict intents and entities from user input - even if it doesn't match the sample utterances exactly. You can then use the model from a client application to retrieve predictions and respond appropriately.

Getting started with Conversational Language Understanding

Creating an application with Conversational Language Understanding consists of two main tasks. First you must define entities, intents, and utterances with which to train the language model - referred to as *authoring* the model. Then you must publish the model so that client applications can use it for intent and entity *prediction* based on user input.

Azure resources for Conversational Language Understanding

For each of the authoring and prediction tasks, you need a resource in your Azure subscription. You can use the following types of resource:

- **Language Service**: A resource that enables you to build apps with industry-leading natural language understanding capabilities without machine learning expertise.
- **Cognitive Services**: A general cognitive services resource that includes Conversational Language Understanding along with many other cognitive services. You can only use this type of resource for *prediction*.

The separation of resources is useful when you want to track resource utilization for Language Service use separately from client applications using all Cognitive Services applications.

When your client application uses a Cognitive Services resource, you can manage access to all of the cognitive services being used, including the Language Service, through a single endpoint and key.

Authoring

After you've created an authoring resource, you can use it to author and train a Conversational Language Understanding application by defining the entities and intents that your application will predict as well as utterances for each intent that can be used to train the predictive model.

Conversational Language Understanding provides a comprehensive collection of prebuilt *domains* that include pre-defined intents and entities for common scenarios; which you can use as a starting point for your model. You can also create your own entities and intents.

When you create entities and intents, you can do so in any order. You can create an intent, and select words in the sample utterances you define for it to create entities for them; or you can create the entities ahead of time and then map them to words in utterances as you're creating the intents.

You can write code to define the elements of your model, but in most cases it's easiest to author your model using the Language Understanding portal - a web-based interface for creating and managing Conversational Language Understanding applications.

Tip

Best practice is to use the Language portal for authoring and to use the SDK for runtime predictions.

Creating intents

Define intents based on actions a user would want to perform with your application. For each intent, you should include a variety of utterances that provide examples of how a user might express the intent.

If an intent can be applied to multiple entities, be sure to include sample utterances for each potential entity; and ensure that each entity is identified in the utterance.

Creating entities

There are four types of entities:

- **Machine-Learned**: Entities that are learned by your model during training from context in the sample utterances you provide.
- **List**: Entities that are defined as a hierarchy of lists and sublists. For example, a **device** list might include sublists for **light** and **fan**. For each list entry, you can specify synonyms, such as **lamp** for **light**.
- **RegEx**: Entities that are defined as a *regular expression* that describes a pattern - for example, you might define a pattern like **[0-9]{3}-[0-9]{3}-[0-9]{4}** for telephone numbers of the form ***555-123-4567***.
- **Pattern.any**: Entities that are used with *patterns* to define complex entities that may be hard to extract from sample utterances.

Training the model

After you have defined the intents and entities in your model, and included a suitable set of sample utterances; the next step is to train the model. Training is the process of using your sample utterances to teach your model to match natural language expressions that a user might say to probable intents and entities.

After training the model, you can test it by submitting text and reviewing the predicted intents. Training and testing is an iterative process. After you train your model, you test it with sample utterances to see if the intents and entities are recognized correctly. If they're not, make updates, retrain, and test again.

Predicting

When you are satisfied with the results from the training and testing, you can publish your Conversational Language Understanding application to a prediction resource for consumption.

Client applications can use the model by connecting to the endpoint for the prediction resource, specifying the appropriate authentication key; and submit user input to get predicted intents and entities. The predictions are returned to the client application, which can then take appropriate action based on the predicted intent.

Exercise - Create a Conversational Language Understanding application

Increasingly, we expect computers to be able to use AI in order to understand spoken or typed commands in natural language. For example, you might want to implement a home automation system that enables you to control devices in your home by using voice commands such as "switch on the light" or "put the fan on", and have an AI-powered device understand the command and take appropriate action.

To test the capabilities of the Conversational Language Understanding service, we'll use a command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create a *Language service* resource

You can use the Conversational Language Understanding service by creating a **Language service** resource.

If you haven't already done so, create a **Language service** resource in your Azure subscription.

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Language service*, and create a **Language service** resource with the following settings:
    - Select additional features: *Keep the default features and click Continue to create your resource*
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select or create a resource group with a unique name*.
    - **Region**: East US 2
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: Standard (S)
    - **Legal Terms**: *Agree*
    - **Responsible AI Notice**: *Agree*
3. Review and create the resource, and wait for deployment to complete.

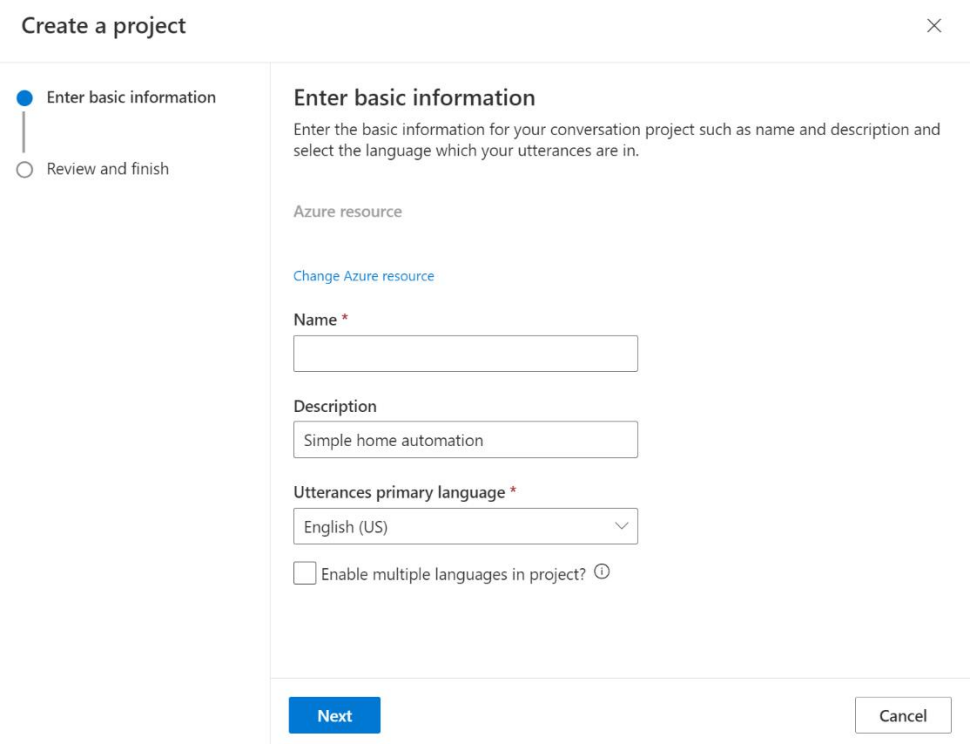
Create a Conversational Language Understanding App

To implement natural language understanding with Conversational Language Understanding, you create an app; and then add entities, intents, and utterances to define the commands you want the app to execute.

1. In a new browser tab, open the Language Studio portal at https://language.azure.com and sign in using the Microsoft account associated with your Azure subscription.
2. If prompted to choose a Language resource, select the following settings:
    - **Azure Directory**: The Azure directory containing your subscription.
    - **Azure subscription**: Your Azure subscription.
    - **Language resource**: The Language resource you created previously.

    Tip

    If you are ***not*** prompted to choose a language resource, it may be because you have multiple Language resources in your subscription; in which case: 1. On the bar at the top if the page, click the **Settings (⚙)** button. 2. On the **Settings** page, view the **Resources** tab. 3. Select your language resource, and click **Switch resource**. 4. At the top of the page, click **Language Studio** to return to the Language Studio home page.
3. At the top of the portal, in the **Create new** menu, select **Conversational language understanding**.
4. In the **Create a project** dialog box, on the **Enter basic information** page, enter the following details and click **Next**:
    - **Name**: *Create a unique name*
    - **Description**: Simple home automation
    - **Utterances primary language**: English
    - **Enable multiple languages in project**: *Do not select*



Tip

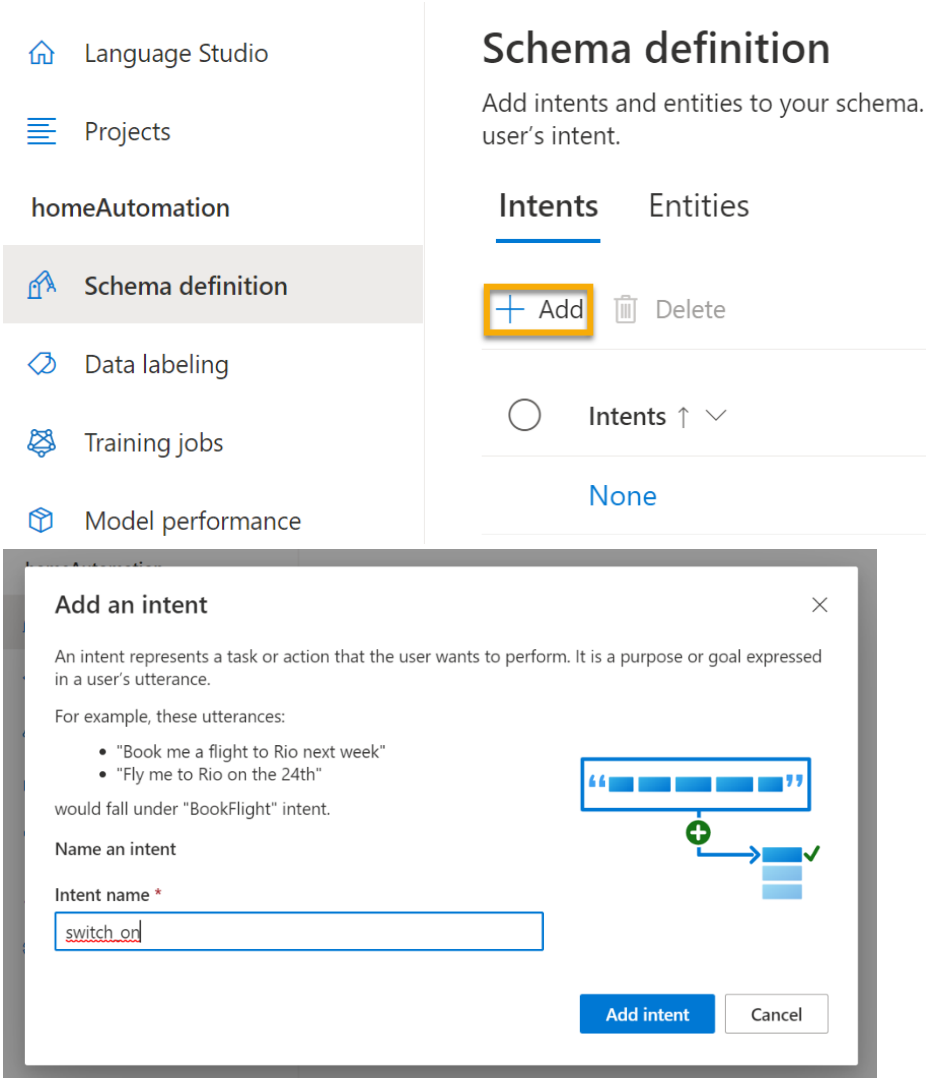
Write down your *project name*, you will use it later.

5. On the *Review and finish* page, click **Create**.

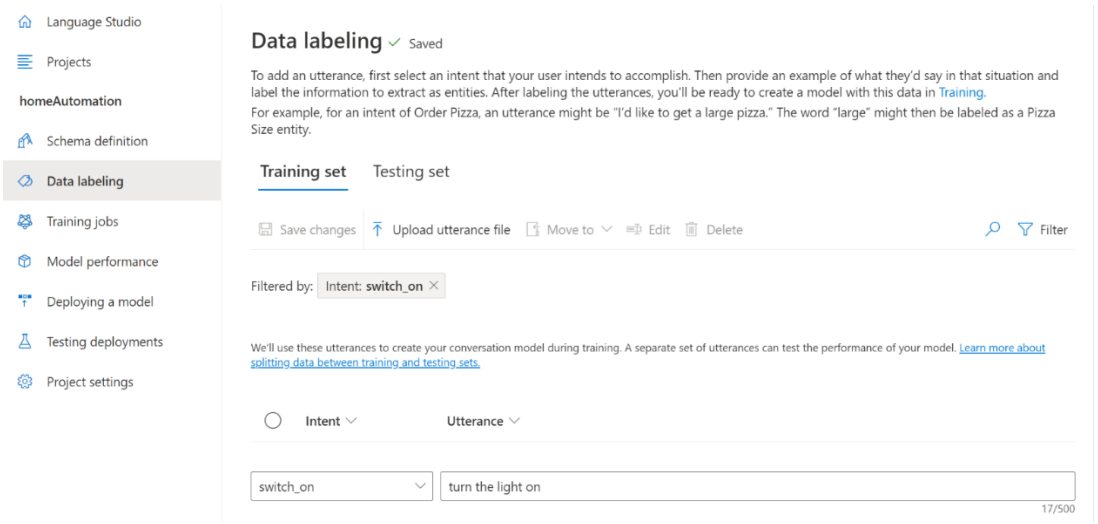
Create intents, utterances, and entities

An *intent* is an action you want to perform - for example, you might want to switch on a light, or turn off a fan. In this case, you'll define two intents: one to switch on a device, and another to switch off a device. For each intent, you'll specify sample *utterances* that indicate the kind of language used to indicate the intent.

1. In the **Schema definition** pane, ensure that **Intents** is selected Then click **Add**, and add an intent with the name **switch_on** (in lower-case) and click **Add intent**.



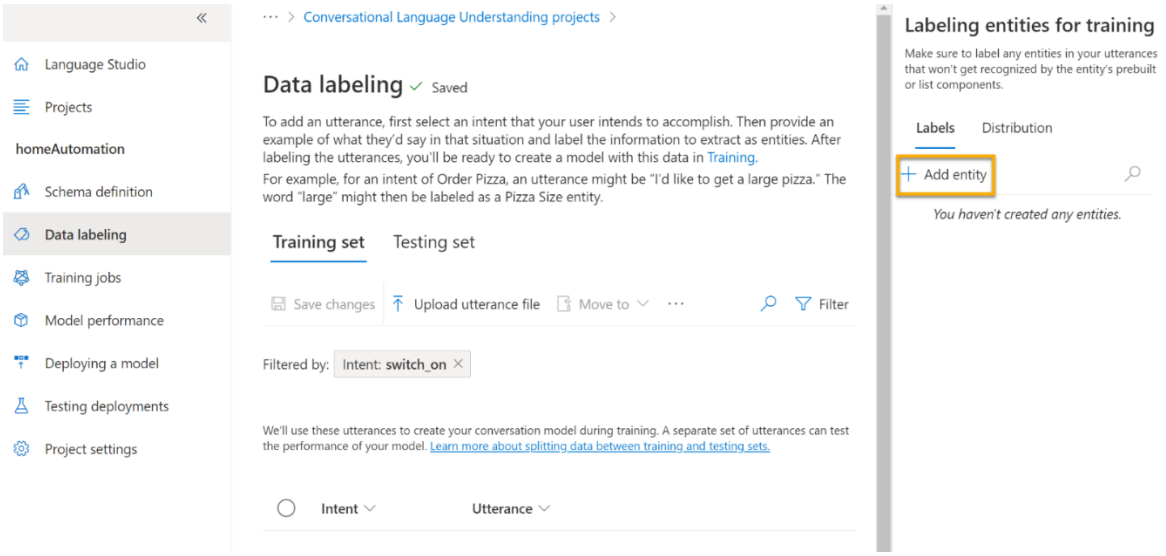
2. Select the **switch_on** intent. It will take you to the **Data labeling** page. Next to the **switch_on** intent, type the utterance ***turn the light on*** and press **Enter** to submit this utterance to the list.

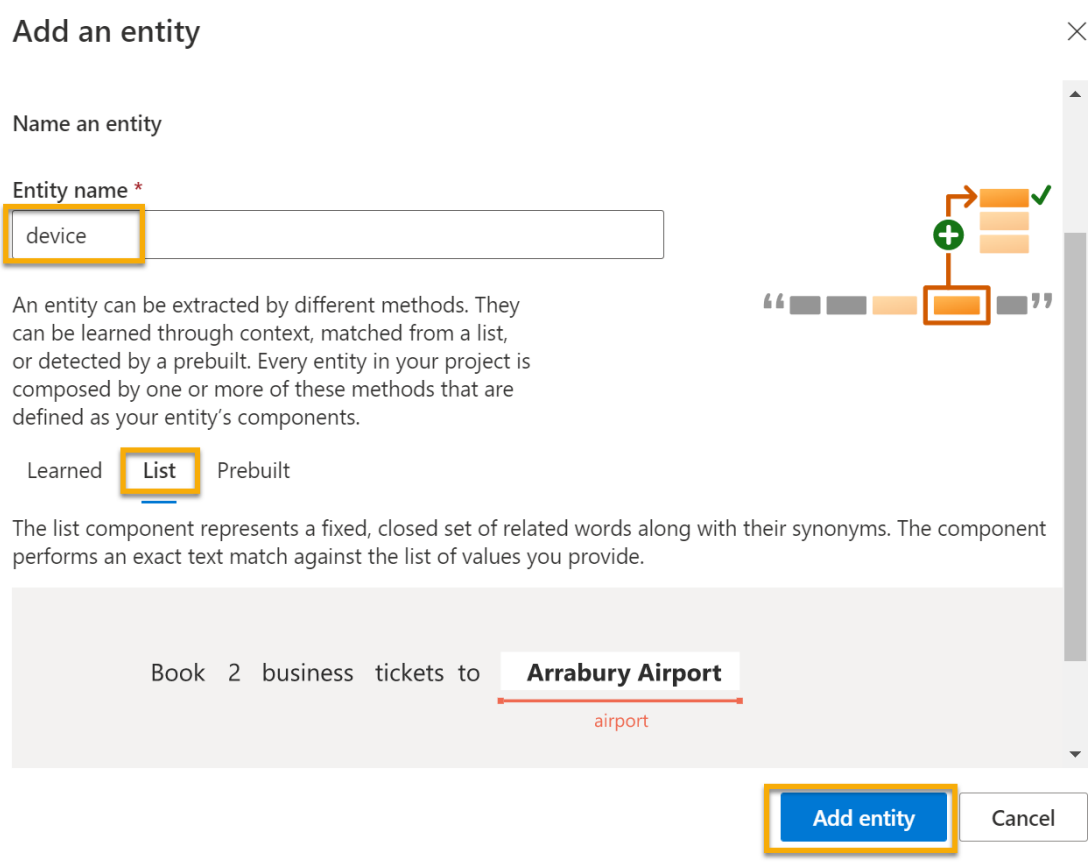


3. The language service needs at least five different utterance examples for each intent to sufficiently train the language model. Add five more utterance examples for the **switch_on** intent:

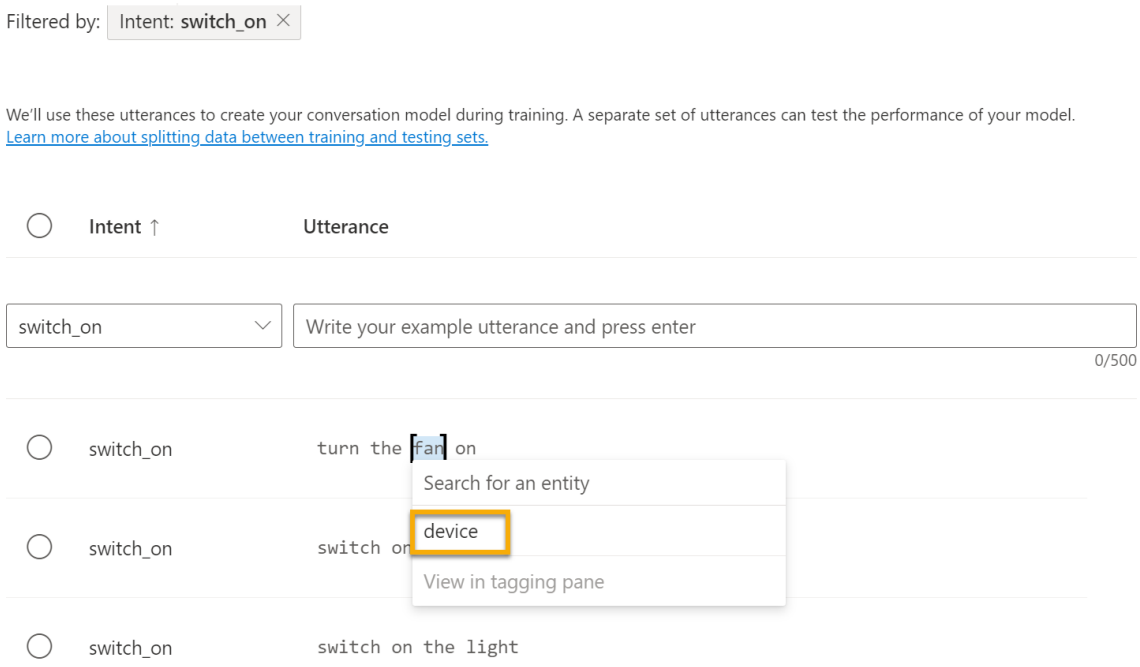
- ***switch on the fan***
- ***put the fan on***
- ***put the light on***
- ***switch on the light***
- ***turn the fan on***

4. On the **Labeling entities for training** pane on the right-hand side of the screen, select **Labels**, then select **Add entity**. Type **device** (in lower-case), select **List** and select **Add entity**.



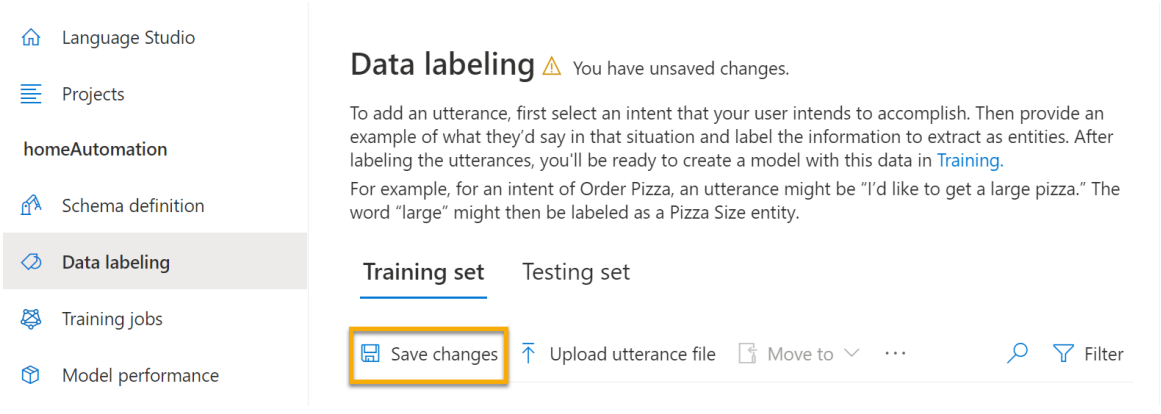


5. In the ***turn the fan on*** utterance, highlight the word "fan". Then in the list that appears, in the *Search for an entity* box select **device**.

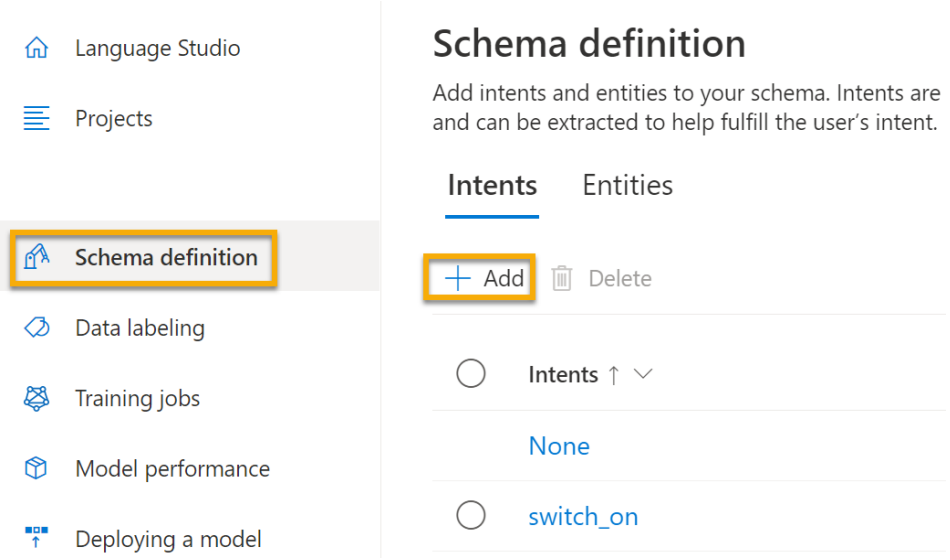


6. Do the same for all the utterances. Label the rest of the *fan* or *light* utterances with the **device** entity. When you're finished, verify that you have the following utterances and make sure to select **Save changes**:

intent	utterance	entity
switch_on	Put on the fan	Device - *select fan*
switch_on	Put on the light	Device - *select light*
switch_on	Switch on the light	Device - *select light*
switch_on	Turn the fan on	Device - *select fan*
switch_on	Switch on the fan	Device - *select fan*
switch_on	Turn the light on	Device - *select light*

7. 

8. In the pane on the left, click **Schema definition** and verify that your **switch_on** intent is listed. Then click **Add** and add a new intent with the name **switch_off** (in lower-case).



9. Click on the **switch_off** intent. It will take you to the **Data labeling** page. Next to the **switch_off** intent, add the utterance ***turn the light off***.
10. Add five more utterance examples to the **switch_off** intent.
    - ***switch off the fan***
    - ***put the fan off***
    - ***put the light off***
    - ***turn off the light***
    - ***switch the fan off***
11. Label the words *light* or *fan* with the **device** entity. When you're finished, verify that you have the following utterances and make sure to select **Save changes**:

intent	utterance	entity
switch_off	Put the fan off	Device - *select fan*
switch_off	Put the light off	Device - *select light*
switch_off	Turn off the light	Device - *select light*
switch_off	Switch the fan off	Device - *select fan*
switch_off	Switch off the fan	Device - *select fan*
switch_off	Turn the light off	Device - *select light*

Train the model

Now you're ready to use the intents and entities you have defined to train the conversational language model for your app.

1. On the left hand side of Language Studio, select **Training jobs**, then select **Start a training job**. Use the following settings:
    - **Train a new model**: *Selected and choose a model name*
    - **Training mode**: Standard training (free)
    - **Data Splitting**: *select Automatically split the testing set from the training data, keep default percentages*
    - Click **Train** at the bottom of the page.
2. Wait for training to complete.

Deploy and test the model

To use your trained model in a client application, you must deploy it as an endpoint to which the client applications can send new utterances; from which intents and entities will be predicted.

1. On the left-hand side of Language Studio, click **Deploying a model**.
2. Select your model name and click **Add deployment**. Use these settings:
    - **Create or select an existing deployment name**: *Select create a new deployment name. Add a unique name*
    - **Assign trained model to your deployment name**: *Select the name of the trained model*
    - Click **Deploy**

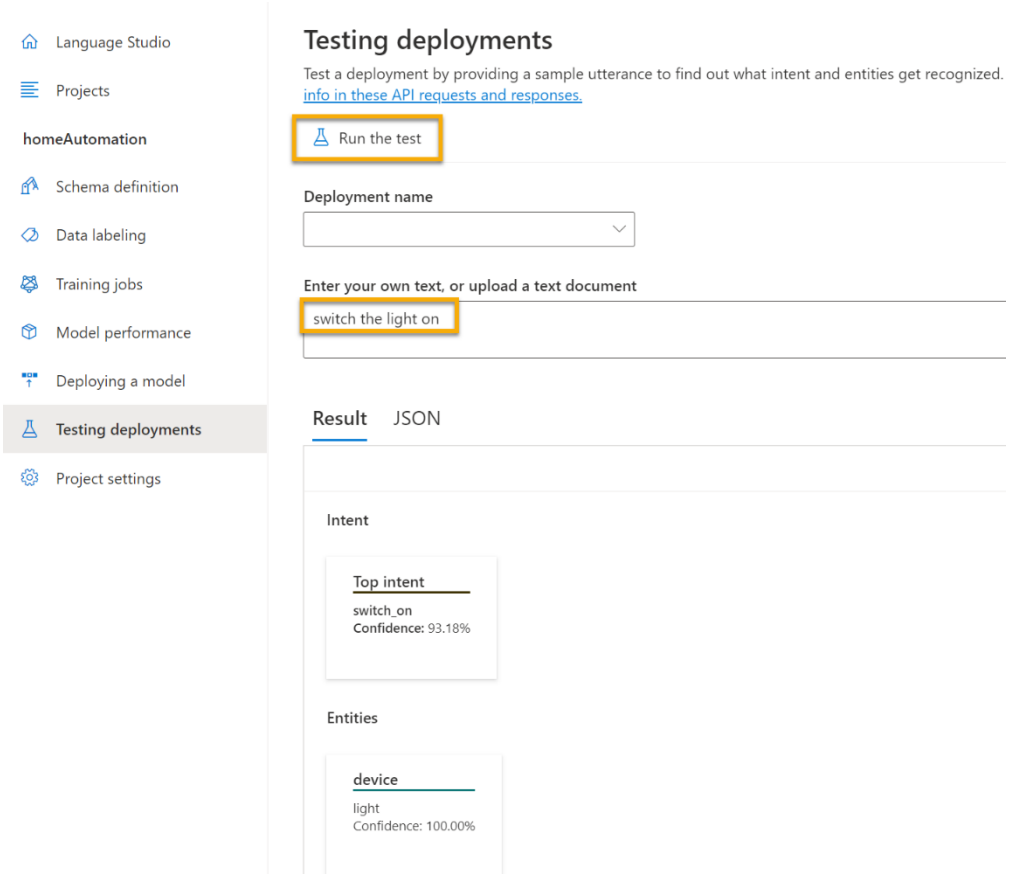
Tip

Write down your *deployment name*, you will use it later.

3. When the model is deployed, click **Testing deployments** on the left-hand side of the page, and then select your deployed model under **Deployment name**.

4. Enter the following text, and then select **Run the test**:

switch the light on



Review the result that is returned, noting that it includes the predicted intent (which should be **switch_on**) and the predicted entity (**device**) with confidence scores that indicates the probability the model calculated for the predicted intent and entity. The JSON tab shows the comparative confidence for each potential intent (the one with the highest confidence score is the predicted intent)

5. Clear the text box and test the model with the following utterances under *Enter your own text*:

- *turn off the fan*
- *put the light on*
- *put the fan off*

Run Cloud Shell

Now let's try out your deployed model. To do so, we'll use a command-line application that runs in the Cloud Shell on Azure.

1. Leaving the browser tab with Language Studio open, switch back to browser tab containing the Azure portal.
2. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. Clicking the button opens a Cloud Shell pane at the bottom of the portal.
3. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
4. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.
5. Make sure the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.
6. Wait for PowerShell to start. You should see the following screen in the Azure portal:

Configure and run a client application

Now let's open and edit a pre-written script, which will run the client application.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

   Note

   If you already used this command in another lab to clone the *ai-900* repository, you can skip this step.

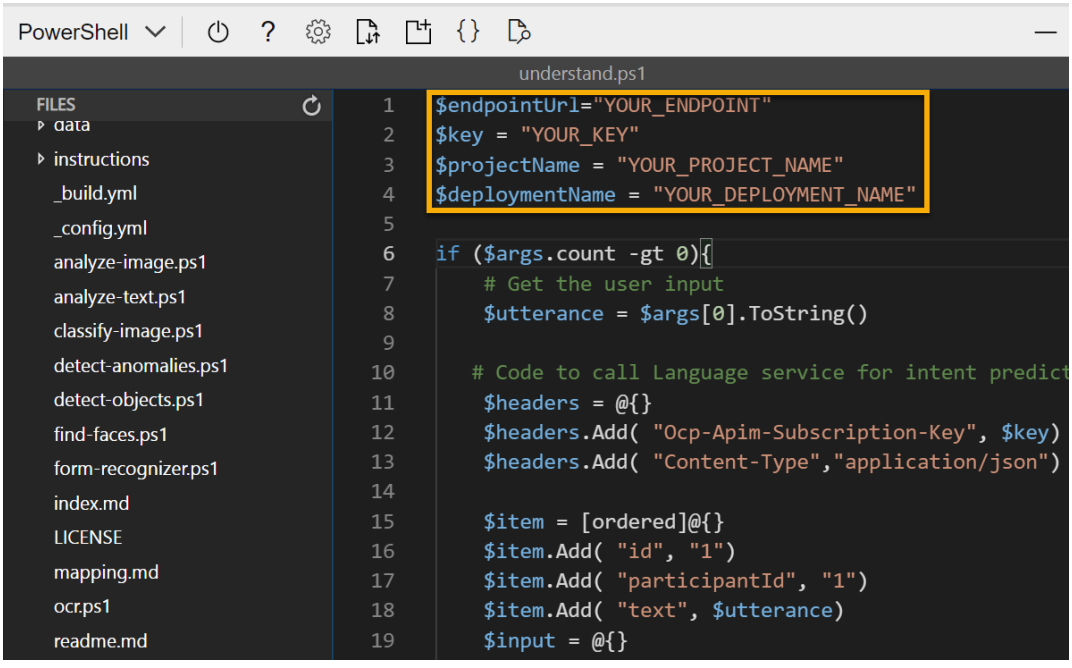
2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in this folder and work with them. Type the following commands into the shell:

   Copy

   ```
   cd ai-900
   code .
   ```

   Notice how the script opens up an editor like the one in the image below:

3. In the **Files** pane on the left, select the **understand.ps1** file in the **ai-900** folder. This file contains some code that uses your Conversational Language Understanding model.



Don't worry too much about the details of the code. The important thing is that you'll use the instructions below to modify the file to specify the language model you trained.

4. Switch back to the browser tab containing **Language Studio**. Then in Language Studio, open the **Deploying a model** page and select your model. Then click the **Get prediction URL** button. The two pieces of information you need are in this dialog box:
    - The endpoint for your model - you can copy the endpoint from the **Prediction URL** box.
    - The key for your model - the key is in the **Sample request** as the value for the **Ocp-Apim-Subscription-Key** parameter, and looks similar to ***0ab1c23de4f56gh7i8901234jkl567m8***.
5. Copy the endpoint value, then switch back to the browser tab containing the Cloud Shell and paste it into the code editor, replacing **YOUR_ENDPOINT** (within the quotation marks). The repeat that process for the key, replacing **YOUR_KEY**.
6. Next, replace **YOUR_PROJECT_NAME** with the name of your project, and replace **YOUR_DEPLOYMENT_NAME** with the name of your deployed model. The first lines of code should look similar to what you see below:

    PowerShellCopy

    ```
    $endpointUrl="https://some-name.cognitiveservices.azure.com/language/..."
    $key = "0ab1c23de4f56gh7i8901234jkl567m8"
    $projectName = "name"
    $deploymentName = "name"
    ```

7. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.
8. In the PowerShell pane, enter the following command to run the code:

    Copy

    ```
    ./understand.ps1 "Turn on the light"
    ```

9. Review the results. The app should have predicted that the intended action is to switch on the light.
10. Now try another command:

    Copy

    ```
    ./understand.ps1 "Switch the fan off"
    ```

11. Review the results from this command. The app should have predicted that the intended action is to switch off the fan.
12. Experiment with a few more commands; including commands that the model was not trained to support, such as "Hello" or "switch on the oven". The app should generally understand commands for which its language model is defined, and fail gracefully for other input.

Note

Each time you will need to start with **./understand.ps1** followed by the phrase. Include quotation marks around your phrase.

Learn more

This app shows only some of the capabilities of the Conversational Language Understanding feature of the Language service. To learn more about what you can do with this service, see the Conversational Language Understanding page.

Knowledge check

1. You need to provision an Azure resource that will be used to **author** a new Language Understanding application. What kind of resource should you create?

- Custom Language service

Incorrect. The Custom Language service is not an Azure service.

Language service

Correct. To author a Conversational Language Understanding model, you need a Language resource.

Cognitive Services

2. You are authoring a Conversational Language Understanding application to support an international clock. You want users to be able to ask for the current time in a specified city, for example "What is the time in London?". What should you do?

Define a "city" entity and a "GetTime" intent with utterances that indicate the city intent.

Correct. The intent encapsulates the task (getting the time) and the entity specifies the item to which the intent is applied (the city).

Create an intent for each city, each with an utterance that asks for the time in that city.

Incorrect. The intent is the same (get the time), the city varies and can be identified as an entity.

Add the utterance "What time is it in city" to the "None" intent.

3. You have published your Conversational Language Understanding application. What information does a client application developer need to get predictions from it?

The endpoint and key for the application's prediction resource

Correct. Client applications must connect to the endpoint of the prediction resource, specifying an associated authentication key.

The endpoint and key for the application's authoring resource

The Azure credentials of the user who published the Language Understanding application

Incorrect. You should not share individual user credentials with other users.

Part 4/6_5/5

Build a bot with the Language Service and Azure Bot Service

Bots are a popular way to provide support through multiple communication channels. This module describes how to use a knowledge base and Azure Bot Service to create a bot that answers user questions.

Introduction

In today's connected world, people use a variety of technologies to communicate. For example:

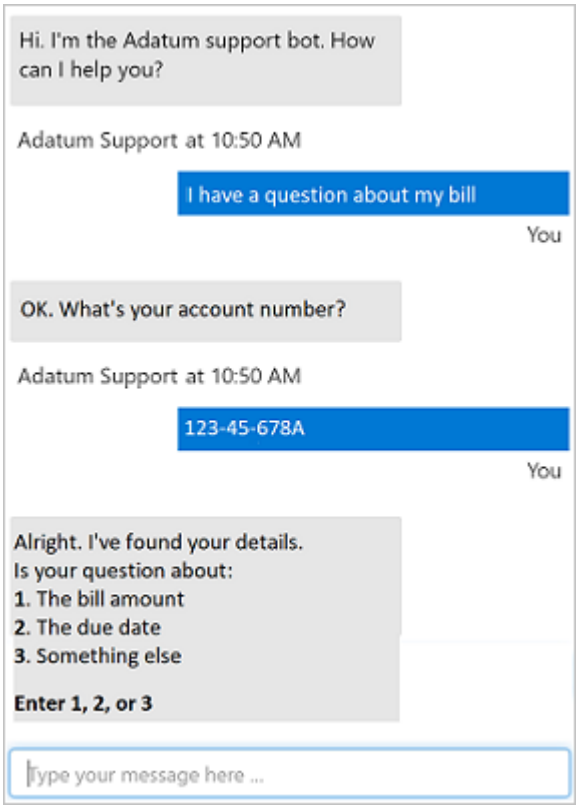
- Voice calls
- Messaging services
- Online chat applications
- Email
- Social media platforms
- Collaborative workplace tools

We've become so used to ubiquitous connectivity, that we expect the organizations we deal with to be easily contactable and immediately responsive through the channels we already use. Additionally, we expect these organizations to engage with us individually, and be able to answer complex questions at a personal level.

Conversational AI

While many organizations publish support information and answers to frequently asked questions (FAQs) that can be accessed through a web browser or dedicated app. The complexity of the systems and services they offer means that answers to specific questions are hard to find. Often, these organizations find their support personnel being overloaded with requests for help through phone calls, email, text messages, social media, and other channels.

Increasingly, organizations are turning to artificial intelligence (AI) solutions that make use of AI agents, commonly known as *bots* to provide a first-line of automated support through the full range of channels that we use to communicate. Bots are designed to interact with users in a conversational manner, as shown in this example of a chat interface:



Note

The example shown here is a chat interface, such as you might find on a web site; but bots can be designed to work across multiple channels, including email, social media platforms, and even voice calls. Regardless of the channel used, bots typically manage conversation flows using a combination of natural language and constrained option responses that guide the user to a resolution.

Conversations typically take the form of messages exchanged in turns; and one of the most common kinds of conversational exchange is a question followed by an answer. This pattern forms the basis for many user support bots, and can often be based on existing FAQ documentation. To implement this kind of solution, you need:

- A *knowledge base* of question and answer pairs - usually with some built-in natural language processing model to enable questions that can be phrased in multiple ways to be understood with the same semantic meaning.
- A *bot service* that provides an interface to the knowledge base through one or more channels.

Get started with the Language service and Azure Bot Service

You can easily create a user support bot solution on Microsoft Azure using a combination of two core services:

- **Language service**. The Language service includes a custom question answering feature that enables you to create a knowledge base of question and answer pairs that can be queried using natural language input.

  Note

  The question answering capability in the Language service is a newer version of the QnA Maker service - which is still available as a separate service.

- **Azure Bot service**. This service provides a framework for developing, publishing, and managing bots on Azure.

Creating a custom question answering knowledge base

The first challenge in creating a user support bot is to use the Language service to create a knowledge base. You can use the *Language Studio*'s custom question answering feature to create, train, publish, and manage knowledge bases.

Note

You can write code to create and manage knowledge bases using the Language service REST API or SDK. However, in most scenarios it is easier to use the Language Studio.

Provision a Language service Azure resource

To create a knowledge base, you must first provision a **Language service** resource in your Azure subscription.

Define questions and answers

After provisioning a Language service resource, you can use the Language Studio's custom question answering feature to create a knowledge base that consists of question-and-answer pairs. These questions and answers can be:

- Generated from an existing FAQ document or web page.

- Entered and edited manually.

In many cases, a knowledge base is created using a combination of all of these techniques; starting with a base dataset of questions and answers from an existing FAQ document and extending the knowledge base with additional manual entries.

Questions in the knowledge base can be assigned *alternative phrasing* to help consolidate questions with the same meaning. For example, you might include a question like:

What is your head office location?

You can anticipate different ways this question could be asked by adding an alternative phrasing such as:

Where is your head office located?

Test the knowledge base

After creating a set of question-and-answer pairs, you must save it. This process analyzes your literal questions and answers and applies a built-in natural language processing model to match appropriate answers to questions, even when they are not phrased exactly as specified in your question definitions. Then you can use the built-in test interface in the Language Studio to test your knowledge base by submitting questions and reviewing the answers that are returned.

Use the knowledge base

When you're satisfied with your knowledge base, deploy it. Then you can use it over its REST interface. To access the knowledge base, client applications require:

- The knowledge base ID
- The knowledge base endpoint
- The knowledge base authorization key

Build a bot with the Azure Bot Service

After you've created and deployed a knowledge base, you can deliver it to users through a bot.

Create a bot for your knowledge base

You can create a custom bot by using the Microsoft Bot Framework SDK to write code that controls conversation flow and integrates with your knowledge base. However, an easier approach is to use the automatic bot creation functionality, which enables you create a bot for your deployed knowledge base and publish it as an Azure Bot Service application with just a few clicks.

Extend and configure the bot

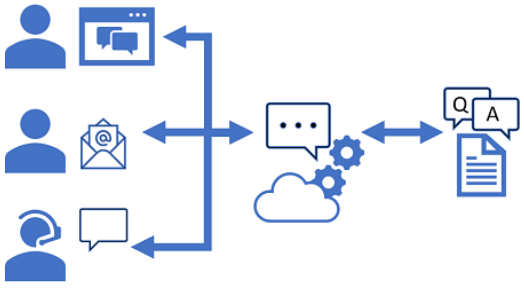
After creating your bot, you can manage it in the Azure portal, where you can:

- Extend the bot's functionality by adding custom code.
- Test the bot in an interactive test interface.
- Configure logging, analytics, and integration with other services.

For simple updates, you can edit bot code directly in the Azure portal. However, for more comprehensive customization, you can download the source code and edit it locally; republishing the bot directly to Azure when you're ready.

Connect channels

When your bot is ready to be delivered to users, you can connect it to multiple *channels*; making it possible for users to interact with it through web chat, email, Microsoft Teams, and other common communication media.



Users can submit questions to the bot through any of its channels, and receive an appropriate answer from the knowledge base on which the bot is based.

Exercise - Create a bot

For customer support scenarios, it's common to create a bot that can interpret and answer frequently asked questions through a website chat window, email, or voice interface. Underlying the bot interface is a knowledge base of questions and appropriate answers that the bot can search for suitable responses.

Create a custom question answering knowledge base

The Language service's custom question answering feature enables you to quickly create a knowledge base, either by entering question and answer pairs or from an existing document or web page. It can then use some built-in natural language processing capabilities to interpret questions and find appropriate answers.

1. Open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Language service*, and create a **Language service** resource with the following settings: **Select Additional Features**
    - **Default features**: *Keep the default features*
    - **Custom features**: *Select custom question answering* Click **Continue to create your resource**.
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Select an existing resource group or create a new one*
    - **Name**: *A unique name for your Language resource*
    - **Pricing tier**: Standard S
    - **Azure Search location**: *Any available location*
    - **Azure Search pricing tier**: Free F (*If this tier is not available, select Standard (S)*)
    - **Legal Terms**: *Agree*
    - **Responsible AI Notice**: *Agree*

    Note

    If you have already provisioned a free-tier **Azure Cognitive Search** resources, your quota may not allow you to create another one. In which case, select a tier other than **F**.
3. Click **Review and Create** and then click **Create**. Wait for the deployment of the Language service that will support your custom question answering knowledge base.
4. In a new browser tab, open the Language Studio portal at https://language.azure.com and sign in using the Microsoft account associated with your Azure subscription.
5. If prompted to choose a Language resource, select the following settings:
    - **Azure Directory**: The Azure directory containing your subscription.
    - **Azure subscription**: Your Azure subscription.
    - **Language resource**: The Language resource you created previously.
6. If you are ***not*** prompted to choose a language resource, it may be because you have multiple Language resources in your subscription; in which case:
    - On the bar at the top if the page, click the **Settings (⚙)** button.
    - On the **Settings** page, view the **Resources** tab.
    - Select the language resource you just created, and click **Switch resource**.
    - At the top of the page, click **Language Studio** to return to the Language Studio home page.
7. At the top of the Language Studio portal, in the **Create new** menu, select **Custom question answering**.
8. On the **Enter basic information** page, enter the following details and click **Next**:
    - **Language resource**: *choose your language resource*.
    - **Azure search resource**: *choose your Azure search resource*.

- **Name**: MargiesTravel
- **Description**: A simple knowledge base
- **Source language**: English
- **Default answer when no answer is returned**: No answer found

9. On the *Review and finish* page, click **Create project**.
10. You will be taken to the **Manage sources** page. Click **+Add source** and select **URLs**.
11. In the **Add URLs** box, click **+ Add URL**. Type in the following:
    - **URL name**: MargiesKB
    - **URL**: https://raw.githubusercontent.com/MicrosoftLearning/AI-900-AIFundamentals/main/data/qna/margies_faq.docx
    - **Classify file structure**: *Auto-detect* Select **Add all**.

Edit the knowledge base

Your knowledge base is based on the details in the FAQ document and some pre-defined responses. You can add custom question-and-answer pairs to supplement these.

1. Click **Edit knowledge base** on the left hand panel. Then click **+ Add question pair**.
2. In the **Questions** box, type Hello. Then click **+ Add alternative phrasing** and type Hi.
3. In the **Answer and prompts** box, type Hello. Keep the **Source**: Editorial.
4. Click **Submit**. Then at the top of the page click **Save changes**. You may need to change the size of your window to see the button.

Train and test the knowledge base

Now that you have a knowledge base, you can test it.

1. At the top of the page, click **Test** to test your knowledge base.
2. In the test pane, at the bottom enter the message *Hi*. The response **Hello** should be returned.
3. In the test pane, at the bottom enter the message *I want to book a flight*. An appropriate response from the FAQ should be returned.

   Note

   The response includes a *short answer* as well as a more verbose *answer passage* - the answer passage shows the full text in the FAQ document for the closest matched question, while the short answer is intelligently extracted from the passage. You can control whether the short answer is from the response by using the **Display short answer** checkbox at the top of the test pane.
4. Try another question, such as *How can I cancel a reservation?*
5. When you're done testing the knowledge base, click **Test** to close the test pane.

Create a bot for the knowledge base

The knowledge base provides a back-end service that client applications can use to answer questions through some sort of user interface. Commonly, these client applications are bots. To make the knowledge base available to a bot, you must publish it as a service that can be accessed over HTTP. You can then use the Azure Bot Service to create and host a bot that uses the knowledge base to answer user questions.

1. At the left of the Language Studio page, click **Deploy knowledge base**. Click **Deploy**.
2. After the service has been deployed, click **Create a Bot**. This opens the Azure portal in a new browser tab so you can create a Web App Bot in your Azure subscription.
3. In the Azure portal, create a Web App Bot with the following settings (most of these will be pre-populated for you):
   - **Bot handle**: *A unique name for your bot*
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *The resource group containing your Language resource*

- **Location**: *The same location as your Language service*.
- **Pricing tier**: Free (F0)
- **App name**: *Same as the **Bot handle** with **.azurewebsites.net** appended automatically*
- **SDK language**: *Choose either C# or Node.js*
- **Language Resource Key**: *automatically generated, if you do not see it, you need to start by creating a question answering project in the Language Studio*
- **App service plan/Location**: *Select the arrow to create a plan. Then create a unique App service plan name and choose a suitable location*
- **Application Insights**: Off
- **Microsoft App ID and password**: *Auto create App ID and password*

4. Wait for your bot to be created (the notification icon at the top right, which looks like a bell, will be animated while you wait). Then in the notification that deployment has completed, click **Go to resource** (or alternatively, on the home page, click **Resource groups**, open the resource group where you created the web app bot, and click it.)
5. In the left-hand pane of your bot look for **Settings**, click on **Test in Web Chat**, and wait until the bot displays the message **Hello and welcome!** (it may take a few seconds to initialize).
6. Use the test chat interface to ensure your bot answers questions from your knowledge base as expected. For example, try submitting *I need to cancel my hotel*.

Experiment with the bot. You'll probably find that it can answer questions from the FAQ quite accurately, but it will have limited ability to interpret questions that it has not been trained with. You can always use the Language Studio to edit the knowledge base to improve it, and republish it.

Knowledge check

1. Your organization has an existing frequently asked questions (FAQ) document. You need to create a knowledge base that includes the questions and answers from the FAQ with the least possible effort. What should you do?

- Create an empty knowledge base, and then manually copy and paste the FAQ entries into it.
- Import the existing FAQ document into a new knowledge base.

Correct. You can import question and answer pairs from an existing FAQ document into a question answering knowledge base.

- Import a pre-defined chit-chat data source.

2. You want to create a knowledge base for bots. What service would you use?

- Conversational Language Understanding
- Question Answering

That is correct.

- Azure Bot

3. You need to deliver a support bot for internal use in your organization. Some users want to be able to submit questions to the bot using Microsoft Teams, others want to use a web chat interface on an internal web site. What should you do?

- Create a knowledge base. Then create a bot for the knowledge base and connect the Web Chat and Microsoft Teams channels for your bot

Correct. The Microsoft Teams channel enables your bot to receive and respond to messages in Microsoft Teams, and the Web Chat channel enables interactions through a web chat interface.

Module complete:

Part 5/6

Microsoft Azure AI Fundamentals: Explore decision support

Learn how to automate decision support processes using Azure.

Introduction

Anomaly detection is an artificial intelligence technique used to determine whether values in a series are within expected parameters.

There are many scenarios where anomaly detection is helpful. For example, a smart HVAC system might use anomaly detection to monitor temperatures in a building and raise an alert if the temperature goes above or below the expected value for a given period of time.

Other scenarios include:

- monitoring blood pressure
- evaluating mean time between failures for hardware products
- comparing month-over-month expenses for product costs

The Azure Anomaly Detector service is a cloud-based service that helps you monitor and detect abnormalities in your historical time series and real-time data.

Learning objectives

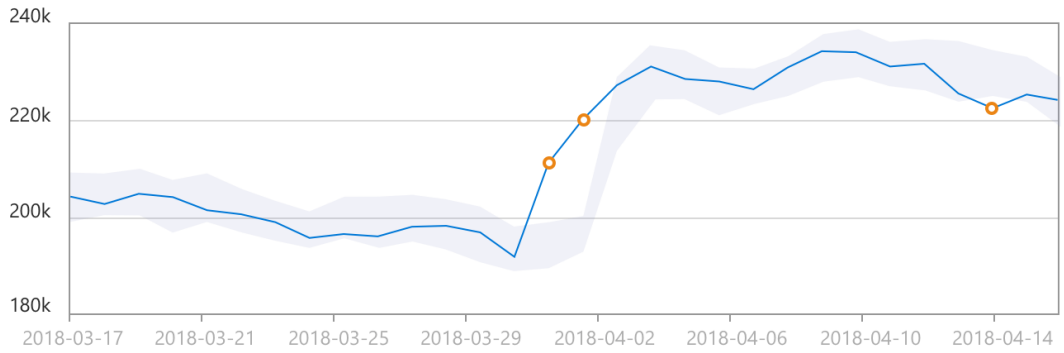
After completing this module, you'll be able to:

- Describe what anomaly detection is
- Describe how the Anomaly Detector service can evaluate time series data
- Define scenarios where anomaly detection can be applied for real-time and historical data

---

What is Anomaly Detector?

Anomalies are values that are outside the expected values or range of values.



In the graphic depicting the time series data, there is a light shaded area that indicates the boundary, or sensitivity range. The solid blue line is used to indicate the measured values. When a measured value is outside of the shaded boundary, an orange dot is used to indicate the value is considered an anomaly. The sensitivity boundary is a parameter that you can specify when calling the service. It allows you to adjust that boundary settings to tweak the results.

Anomaly detection is considered the act of identifying events, or observations, that differ in a significant way from the rest of the data being evaluated. Accurate anomaly detection leads to prompt troubleshooting, which helps to avoid revenue loss and maintain brand reputation.

Azure's Anomaly Detector service

Anomaly Detector is a part of the Decision Services category within Azure Cognitive Services. It is a cloud-based service that enables you to monitor time series data, and to detect anomalies in that data. It does not require you to know machine learning. You can use the REST API to integrate Anomaly Detector into your applications with relative ease. The service uses the concept of a "one parameter" strategy. The main parameter you need to customize is “Sensitivity”, which is from 1 to 99 to adjust the outcome to fit the scenario. The service can detect anomalies in historical time series data and also in real-time data such as streaming input from IoT devices, sensors, or other streaming input sources.

How Anomaly Detector works

The Anomaly Detector service identifies anomalies that exist outside the scope of a boundary. The boundary is set using a sensitivity value. By default, the upper and lower boundaries for anomaly detection are calculated using concepts known as **expectedValue**, **upperMargin**, and **lowerMargin**. The upper and lower boundaries are calculated using these three values. If a value exceeds either boundary, it will be identified as an anomaly. You can adjust the boundaries by applying a **marginScale** to the upper and lower margins as demonstrated by the following formula.

*upperBoundary = expectedValue + (100 - marginScale) * upperMargin*

Data format

The Anomaly Detector service accepts data in JSON format. You can use any numerical data that you have recorded over time. The key aspects of the data being sent includes the granularity, a timestamp, and a value that was recorded for that timestamp. An example of a JSON object that you might send to the API is shown in this code sample. The granularity is set as hourly and is used to represent temperatures in degrees Celsius that were recorded at the timestamps indicated.

Copy

```
{
   "granularity": "hourly",
   "series": [
    {
     "timestamp": "2021-03-01T01:00:00Z",
     "value": -10.56
    },
    {
     "timestamp": "2021-03-02T02:00:00Z",
     "value": -8.30
    },
    {
     "timestamp": "2021-03-02T03:00:00Z",
     "value": -10.30
    },
    {
     "timestamp": "2021-03-02T04:00:00Z",
     "value": 5.95
    },
   ]
}
```

The service will support a maximum of 8640 data points however, sending this many data points in the same JSON object, can result in latency for the response. You can improve the response by breaking your data points into smaller chunks (windows) and sending these in a sequence.

The same JSON object format is used in a streaming scenario. The main difference is that you will send a single value in each request. The streaming detection method will compare the current value being sent and the previous value sent.

Data consistency recommendations

If your data may have missing values in the sequence, consider the following recommendations.

- Sampling occurs every few minutes and has less than 10% of the expected number of points missing. In this case, the impact should be negligible on the detection results.
- If you have more than 10% missing, there are options to help "fill" the data set. Consider using a linear interpolation method to fill in the missing values and complete the data set. This will fill gaps with evenly distributed values.

The Anomaly Detector service will provide the best results if your time series data is evenly distributed. If the data is more randomly distributed, you can use an aggregation method to create a more even distribution data set.

When to use Anomaly Detector

The Anomaly Detector service supports batch processing of time series data and last-point anomaly detection for real-time data.

Batch detection

Batch detection involves applying the algorithm to an entire data series at one time. The concept of time series data involves evaluation of a data set as a batch. Use your time series to detect any anomalies that might exist throughout your data. This operation generates a model using your entire time series data, with each point analyzed using the same model.

Batch detection is best used when your data contains:

- Flat trend time series data with occasional spikes or dips
- Seasonal time series data with occasional anomalies
    - Seasonality is considered to be a pattern in your data, that occurs at regular intervals. Examples would be hourly, daily, or monthly patterns. Using seasonal data, and specifying a period for that pattern, can help to reduce the latency in detection.

When using the batch detection mode, Anomaly Detector creates a single statistical model based on the entire data set that you pass to the service. From this model, each data point in the data set is evaluated and anomalies are identified.

Batch detection example

Consider a pharmaceutical company that stores medications in storage facilities where the temperature in the facilities needs to remain within a specific range. To evaluate whether the medication remained stored in a safe temperature range in the past three months we need to know:

- the maximum allowable temperature
- the minimum allowable temperature
- the acceptable duration of time for temperatures to be outside the safe range

If you are interested in evaluating compliance over historical readings, you can extract the required time series data, package it into a JSON object, and send it to the Anomaly Detector service for evaluation. You will then have a historical view of the temperature readings over time.

Real-time detection

Real-time detection uses streaming data by comparing previously seen data points to the last data point to determine if your latest one is an anomaly. This operation generates a model using the data points you send, and determines if the target (current) point is an anomaly. By calling the service with each new data point you generate, you can monitor your data as it's created.

Real-time detection example

Consider a scenario in the carbonated beverage industry where real-time anomaly detection may be useful. The carbon dioxide added to soft drinks during the bottling or canning process needs to stay in a specific temperature range.

Bottling systems use a device known as a carbo-cooler to achieve the refrigeration of the product for this process. If the temperature goes too low, the product will freeze in the carbo-cooler. If the temperature is too warm, the carbon dioxide will not adhere properly. Either situation results in a product batch that cannot be sold to customers.

This carbonated beverage scenario is an example of where you could use streaming detection for real-time decision making. It could be tied into an application that controls the bottling line equipment. You may use it to feed displays that depict the system temperatures for the quality control station. A service technician may also use it to identify equipment failure potential and servicing needs.

You can use the Anomaly Detector service to create a monitoring application configured with the above criteria to perform real-time temperature monitoring. You can perform anomaly detection using both streaming and batch detection techniques. Streaming detection is most useful for monitoring critical storage requirements that must be acted on immediately. Sensors will monitor the temperature inside the compartment and send these readings to your application or an event hub on Azure. Anomaly Detector will evaluate the streaming data points and determine if a point is an anomaly.

Exercise

To test the capabilities of the Anomaly Detection service, we'll use a simple command-line application that runs in the Cloud Shell. The same principles and functionality apply in real-world solutions, such as web sites or phone apps.

Create an *Anomaly Detector* resource

Let's start by creating an **Anomaly Detector** resource in your Azure subscription:

1. In another browser tab, open the Azure portal at https://portal.azure.com, signing in with your Microsoft account.
2. Click the **+Create a resource** button, search for *Anomaly Detector*, and create an **Anomaly Detector** resource with the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Select an existing resource group or create a new one*.
    - **Region**: *Choose any available region*
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: Free F0
3. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
4. View the **Keys and Endpoint** page for your Anomaly Detector resource. You will need the endpoint and keys to connect from client applications.

Run Cloud Shell

To test the capabilities of the Anomaly Detector service, we'll use a simple command-line application that runs in the Cloud Shell on Azure.

1. In the Azure portal, select the **[>_]** (*Cloud Shell*) button at the top of the page to the right of the search box. This opens a Cloud Shell pane at the bottom of the portal.
2. The first time you open the Cloud Shell, you may be prompted to choose the type of shell you want to use (*Bash* or *PowerShell*). Select **PowerShell**. If you do not see this option, skip the step.
3. If you are prompted to create storage for your Cloud Shell, ensure your subscription is specified and select **Create storage**. Then wait a minute or so for the storage to be created.

4. Make sure the type of shell indicated on the top left of the Cloud Shell pane is switched to *PowerShell*. If it is *Bash*, switch to *PowerShell* by using the drop-down menu.
5. Wait for PowerShell to start. You should see the following screen in the Azure portal:

Configure and run a client application

Now that you have a Cloud Shell environment, you can run a simple application that uses the Anomaly Detector service to analyze data.

1. In the command shell, enter the following command to download the sample application and save it to a folder called ai-900.

   Copy

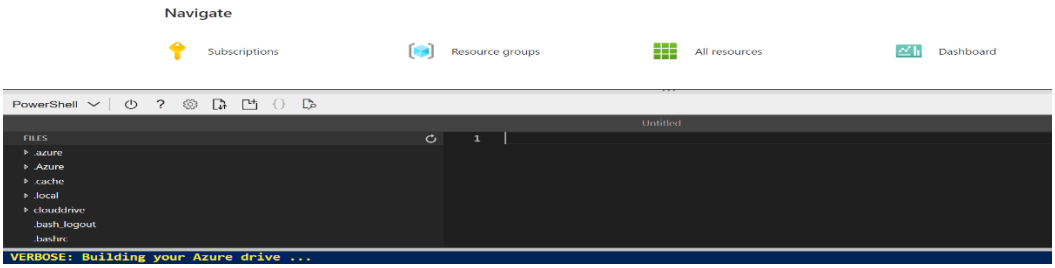
   ```
   git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
   ```

2. The files are downloaded to a folder named **ai-900**. Now we want to see all of the files in your Cloud Shell storage and work with them. Type the following command into the shell:

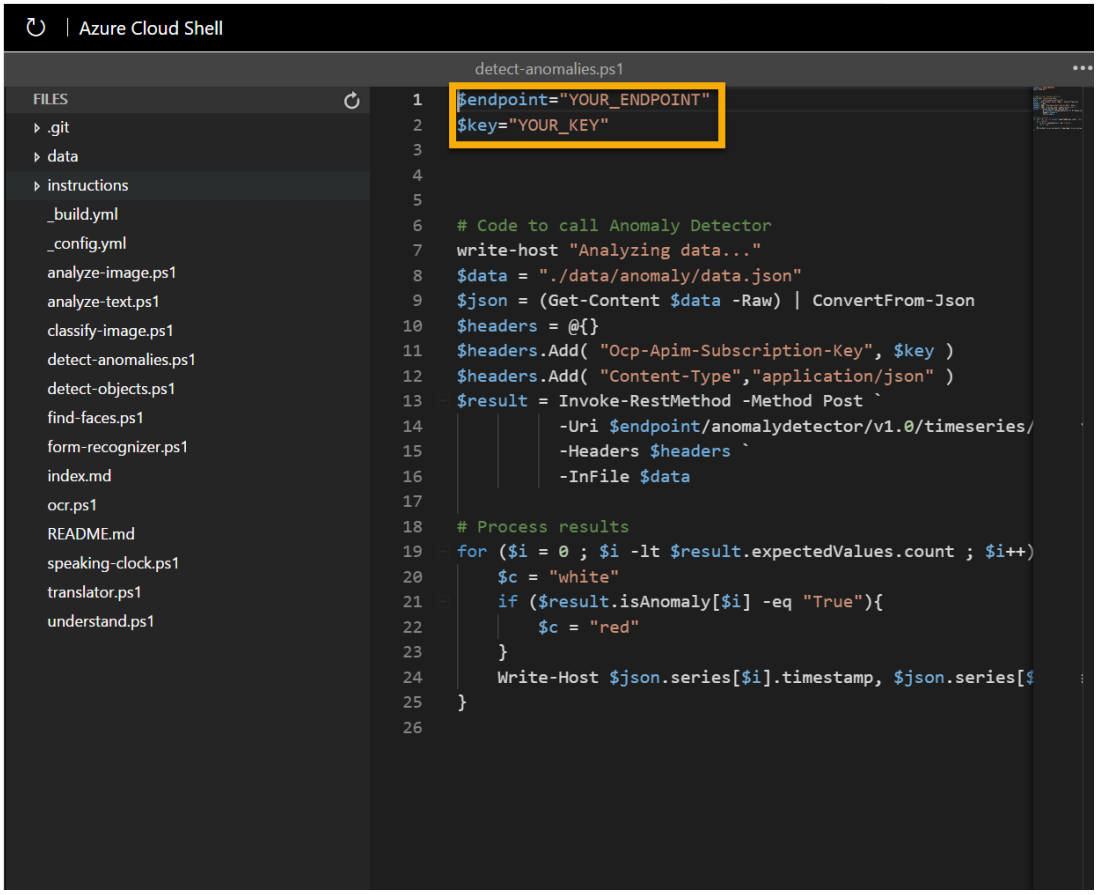
   Copy

   ```
   code .
   ```

   Notice how this opens up an editor like the one in the image below:



3. In the **Files** pane on the left, expand **ai-900** and select **detect-anomalies.ps1**. This file contains some code that uses the Anomaly Detection service, as shown here:



4. Don't worry too much about the details of the code, the important thing is that it needs the endpoint URL and either of the keys for your Anomaly Detector resource. Copy these from the **Keys and Endpoints** page for your resource (which should still be in the top area of the browser) and paste them into the code editor, replacing the **YOUR_KEY** and **YOUR_ENDPOINT** placeholder values respectively.

   Tip

   You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.

   After pasting the key and endpoint values, the first two lines of code should look similar to this:

   PowerShellCopy

   ```
   $key="1a2b3c4d5e6f7g8h9i0j...."
   $endpoint="https..."
   ```

5. At the top right of the editor pane, use the **...** button to open the menu and select **Save** to save your changes. Then open the menu again and select **Close Editor**.

   Remember, anomaly detection is an artificial intelligence technique used to determine whether values in a series are within expected parameters. The sample client application will use your Anomaly Detector service to analyze a file containing a series of date/times and numeric values. The application should return results indicating at each time point, whether the numeric value is within expected parameters.

6. In the PowerShell pane, enter the following commands to run the code:

   Copy

   ```
   cd ai-900
   .\detect-anomalies.ps1
   ```

7. Review the results, noting that the final column in the results is **True** or **False** to indicate if the value recorded at each date/time is considered an anomaly or not. Consider how we could use this information in a real-life situation. What action could the application trigger if the values were of fridge temperature or blood pressure and anomalies were detected?

Learn more

This simple app shows only some of the capabilities of the Anomaly Detector service. To learn more about what you can do with this service, see the Anomaly Detector page.

Knowledge check

1. What is meant by seasonal data?

- Data based on the time or year it was recorded.
- How far apart the values are by default for each recorded period.

Incorrect. Seasonal data is not concerned with how far apart values are.

- Data occurring at regular intervals.

Correct. Seasonal times series is considered to be a pattern in your data, that occurs at regular intervals. Examples would be hourly, daily, or monthly patterns.

2. What is the purpose of specifying granularity in your JSON data object?

- It is used to indicate the recording pattern of the data.

Correct. Granularity would be specified as hourly, daily, weekly, etc.

- It tells the service how to chunk up the results that are returned for review, independent of the time series data pattern.

Incorrect. Granularity is not used to modify the results returned by the service.

- It is used to indicate the range of acceptable values.

3. How does the Anomaly Detector service evaluate real-time data for anomalies?

- It collects all the values in a window of time and evaluates them all at once.
- It evaluates the current value against the previous value.

Correct. It evaluates previously seen data points to determine if your latest one is an anomaly.

- It uses interpolation based on the current value and the previous value to predict what the expected value should be.

Incorrect. It does not use interpolation to evaluate an expected value.

Summary

The Anomaly Detector detects anomalies automatically in time series data. It supports two basic detection models. One is for detecting a batch of data with a model trained by the time series sent to the service. The other is used for detecting the last point with the model trained by points before.

Packaging your time series data into a JSON object and passing it to the API, anomalies can be detected in the time series data. Using the returned results can help you identify issues with industrial processes or recorded events. Batch series data is best used to evaluate recorded events that represent seasonal patterns and don't require immediate action. Streaming data points into the API can offer real-time awareness of anomalies that may require immediate action.

The API can be integrated into your applications by using REST calls or by incorporating the appropriate SDK into your code. Using the Anomaly Detector service does not require you to devise, or to be knowledgeable in, machine learning algorithms.

Learn more

To learn more about Anomaly Detector, consider reviewing the Identify abnormal time-series data with Anomaly Detector module on Microsoft Learn.

Module complete:

Part 6/6

Microsoft Azure AI Fundamentals: Explore knowledge mining

Knowledge mining is a discipline in artificial intelligence (AI) that uses a combination of intelligent services to quickly search and learn from vast amounts of information.

Introduction

Searching for information online has never been easier. However, it's still a challenge to find information from documents that aren't in a search index. For example, every day, people deal with unstructured, typed, image-based, or hand-written documents. Often, people must manually read through these documents to extract and record their insights in order to persist the found data. Now we have solutions that can automate information extraction.

Knowledge mining is the term used to describe solutions that involve extracting information from large volumes of often unstructured data. One of these knowledge mining solutions is Azure Cognitive Search, a cloud search service that has tools for building user-managed indexes. The indexes can be used for internal only use, or to enable searchable content on public-facing internet assets.

Importantly, Azure Cognitive Search can utilize the built-in AI capabilities of Azure Cognitive Services such as image processing, content extraction, and natural language processing to perform knowledge mining of documents. The product's AI capabilities makes it possible to index previously unsearchable documents and to extract and surface insights from large amounts of data quickly.

Learning objectives

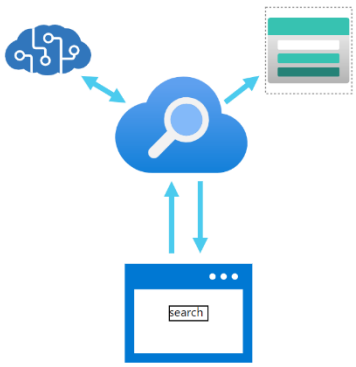
In this module, you will:

- Understand how Azure Cognitive Search uses cognitive skills
- Learn how indexers automate data ingestion steps, including JSON serialization
- Describe the purpose of a knowledge store
- Build and query a search index

What is Azure Cognitive Search?

Azure Cognitive Search provides the infrastructure and tools to create search solutions that extract data from various structured, semi-structured, and non-structured documents.

Azure Cognitive Search results contain only your data, which can include text inferred or extracted from images, or new entities and key phrases detection through text analytics. It's a Platform as a Service (PaaS) solution. Microsoft manages the infrastructure and availability, allowing your organization to benefit without the need to purchase or manage dedicated hardware resources.



Azure Cognitive Search features

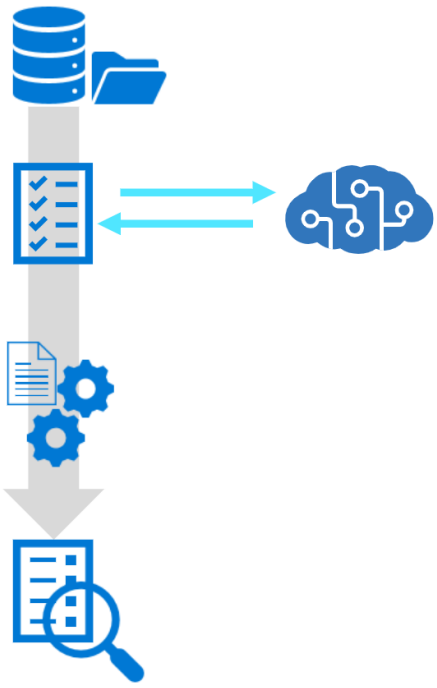
Azure Cognitive Search exists to complement existing technologies and provides a programmable search engine built on Apache Lucene, an open-source software library. It's a highly available platform offering a 99.9% uptime SLA available for cloud and on-premises assets.

Azure Cognitive Search comes with the following features:

- **Data from any source**: Azure Cognitive Search accepts data from any source provided in JSON format, with auto crawling support for selected data sources in Azure.
- **Full text search and analysis**: Azure Cognitive Search offers full text search capabilities supporting both simple query and full Lucene query syntax.
- **AI powered search**: Azure Cognitive Search has Cognitive AI capabilities built in for image and text analysis from raw content.

- **Multi-lingual**: Azure Cognitive Search offers linguistic analysis for 56 languages to intelligently handle phonetic matching or language-specific linguistics. Natural language processors available in Azure Cognitive Search are also used by Bing and Office.
- **Geo-enabled**: Azure Cognitive Search supports geo-search filtering based on proximity to a physical location.
- **Configurable user experience**: Azure Cognitive Search has several features to improve the user experience including autocomplete, autosuggest, pagination, and hit highlighting.

Identify elements of a search solution



A typical Azure Cognitive Search solution starts with a data source that contains the data artifacts you want to search. This could be a hierarchy of folders and files in Azure Storage, or text in a database such as Azure SQL Database or Azure Cosmos DB. The data format that Cognitive Search supports is JSON. Regardless of where your data originates, if you can provide it as a JSON document, the search engine can index it.

If your data resides in supported data source, you can use an indexer to automate data ingestion, including JSON serialization of source data in native formats. An indexer connects to a data source, serializes the data, and passes to the search engine for indexing. Most indexers support change detection, which makes data refresh a simpler exercise.

Besides automating data ingestion, indexers also support AI enrichment. You can attach a skillset that applies a sequence of AI skills to enrich the data, making it more searchable. A comprehensive set of built-in skills, based on Cognitive Services APIs, can help you derive new fields – for example by recognizing entities in text, translating text, evaluating sentiment, or predicting appropriate captions for images. Optionally, enriched content can be sent to a knowledge store, which stores output from an AI enrichment pipeline in tables and blobs in Azure Storage for independent analysis or downstream processing.

Whether you write application code that pushes data to an index - or use an indexer that automates data ingestion and adds AI enrichment - the fields containing your content are persisted in an index, which can be searched by client applications. The fields are used for searching, filtering, and sorting to generate a set of results that can be displayed or otherwise used by the client application.

Use a skillset to define an enrichment pipeline

AI enrichment refers to embedded image and natural language processing in a pipeline that extracts text and information from content that can't otherwise be indexed for full text search.

AI processing is achieved by adding and combining skills in a skillset. A skillset defines the operations that extract and enrich data to make it searchable. These AI skills can be either built-in skills, such as text translation or Optical Character Recognition (OCR), or custom skills that you provide.

Built in skills

Built-in skills are based on pre-trained models from Microsoft, which means you can't train the model using your own training data. Skills that call the Cognitive Resources APIs have a dependency on those services and are billed at the Cognitive Services pay-as-you-go price when you attach a resource. Other skills are metered by Azure Cognitive Search, or are utility skills that are available at no charge.

Built-in skills fall into these categories:

Natural language processing skills: with these skills, unstructured text is mapped as searchable and filterable fields in an index.

Some examples include:

- Key Phrase Extraction: uses a pre-trained model to detect important phrases based on term placement, linguistic rules, proximity to other terms, and how unusual the term is within the source data.
- Text Translation Skill: uses a pre-trained model to translate the input text into various languages for normalization or localization use cases.

Image processing skills: creates text representations of image content, making it searchable using the query capabilities of Azure Cognitive Search.

Some examples include:

- Image Analysis Skill: uses an image detection algorithm to identify the content of an image and generate a text description.
- Optical Character Recognition Skill: allows you to extract printed or handwritten text from images, such as photos of street signs and products, as well as from documents—invoices, bills, financial reports, articles, and more.

Understand indexes

An Azure Cognitive Search index can be thought of as a container of searchable documents. Conceptually you can think of an index as a table and each row in the table represents a document. Tables have columns, and the columns can be thought of as equivalent to the fields in a document. Columns have data types, just as the fields do on the documents.

Index schema

In Azure Cognitive Search, an index is a persistent collection of JSON documents and other content used to enable search functionality. The documents within an index can be thought of as rows in a table, each document is a single unit of searchable data in the index.

The index includes a definition of the structure of the data in these documents, called its schema. An example of an index schema with AI-extracted fields *keyphrases* and *imageTags* is below:

JSONCopy

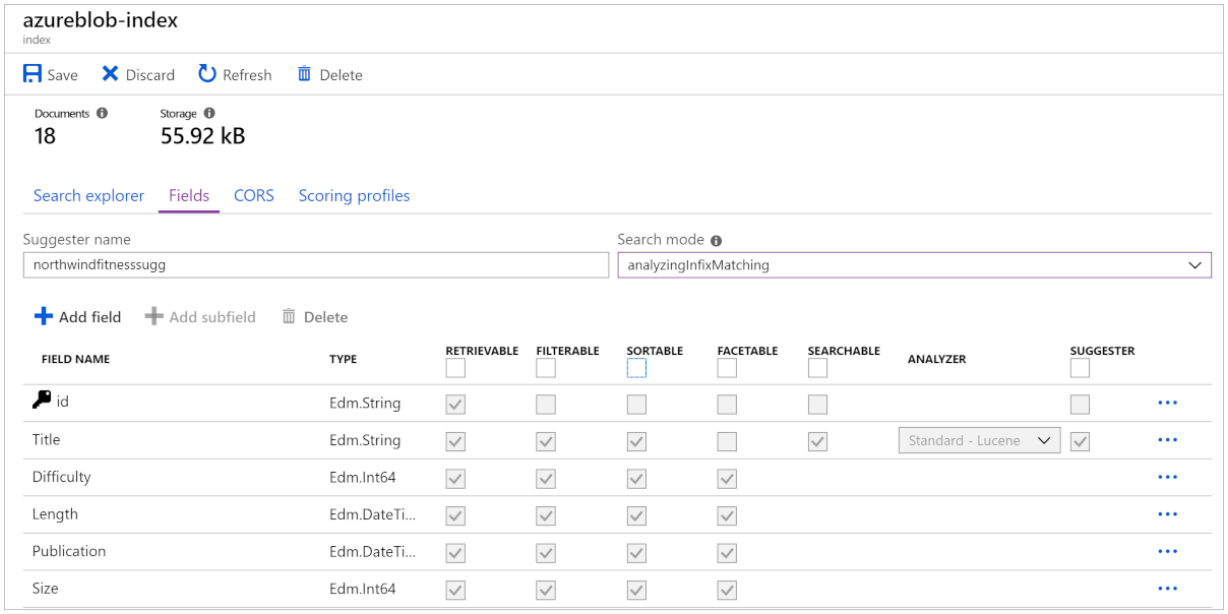
```
{
 "name": "index",
 "fields": [
  {
   "name": "content", "type": "Edm.String", "analyzer": "standard.lucene", "fields": []
  }
  {
   "name": "keyphrases", "type": "Collection(Edm.String)", "analyzer": "standard.lucene", "fields": []
  },
  {
   "name": "imageTags", "type": "Collection(Edm.String)", "analyzer": "standard.lucene", "fields": []
  },
 ]
}
```

Index attributes

Azure Cognitive Search needs to know how you would like to search and display the fields in the documents. You specify that by assigning attributes, or behaviors, to these fields. For each field in the document, the index stores its name, the data type, and supported behaviors for the field such as, is the field searchable, can the field be sorted?

The most efficient indexes use only the behaviors that are needed. If you forget to set a required behavior on a field when designing, the only way to get that feature is to rebuild the index.

The following image depicts the fields when designing an index in Azure:



Use an indexer to build an index

In order to index the documents in Azure Storage, they need to be exported from their original file type to JSON. In order to export data in any format to JSON, and load it into an index, we use an indexer.

To create search documents, you can either generate JSON documents with application code or you can use Azure's indexer to export incoming documents into JSON.

Azure Cognitive Search lets you create and load JSON documents into an index with two approaches:

- **Push method**: JSON data is pushed into a search index via either the REST API or the .NET SDK. Pushing data has the most flexibility as it has no restrictions on the data source type, location, or frequency of execution.
- **Pull method**: Search service indexers can pull data from popular Azure data sources, and if necessary, export that data into JSON if it isn't already in that format.

Use the pull method to load data with an indexer

Azure Cognitive Search's indexer is a crawler that extracts searchable text and metadata from an external Azure data source and populates a search index using field-to-field mappings between source data and your index. Using the indexer is sometimes referred to as a 'pull model' approach because the service pulls data in without you having to write any code that adds data to an index. An indexer maps source fields to their matching fields in the index.

Data import monitoring and verification

The search services overview page has a dashboard that lets you quickly see the health of the search service. On the dashboard, you can see how many documents are in the search service, how may indexes have been used, and how much storage is in use.

When loading new documents into an index, the progress can be monitored by clicking on the index's associated indexer. The document count will grow as documents are loaded into the index. In some instances, the portal page can take a few minutes to display up-to-date document counts. Once the index is ready for querying, you can then use Search explorer to verify the results. An index is ready when the first document is successfully loaded.

Indexers only import new or updated documents, so it is normal to see zero documents indexed.

The Search explorer can perform quick searches to check the contents of an index, and ensure that you are getting expected search results. Having this tool available in the portal enables you to easily check the index by reviewing the results that are returned as JSON documents.

Making changes to an index

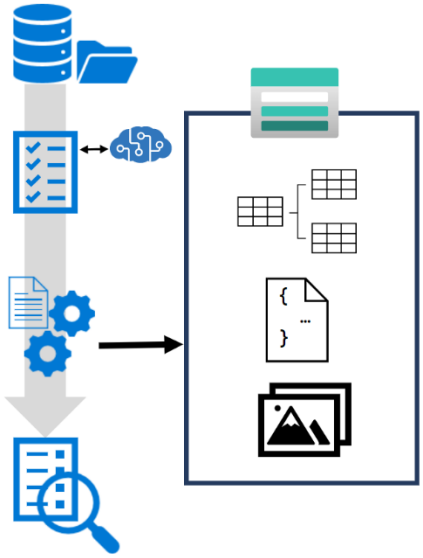
You have to drop and recreate indexes if you need to make changes to field definitions. Adding new fields is supported, with all existing documents having null values. You'll find it faster using a code-based approach to

iterate your designs, as working in the portal requires the index to be deleted, recreated, and the schema details to be manually filled out.

An approach to updating an index without affecting your users is to create a new index under a different name. You can use the same indexer and data source. After importing data, you can switch your app to use the new index.

Persist enriched data in a knowledge store

A knowledge store is persistent storage of enriched content. The purpose of a knowledge store is to store the data generated from AI enrichment in a container. For example, you may want to save the results of an AI skillset that generates captions from images.



Recall that skillsets move a document through a sequence of enrichments that invoke transformations, such as recognizing entities or translating text. The outcome can be a search index, or projections in a knowledge store. The two outputs, search index and knowledge store, are mutually exclusive products of the same pipeline; derived from the same inputs, but resulting in output that is structured, stored, and used in different applications.

While the focus of an Azure Cognitive Search solution is usually to create a searchable index, you can also take advantage of its data extraction and enrichment capabilities to persist the enriched data in a knowledge store for further analysis or processing.

A knowledge store can contain one or more of three types of projection of the extracted data:

- Table projections are used to structure the extracted data in a relational schema for querying and visualization
- Object projections are JSON documents that represent each data entity
- File projections are used to store extracted images in JPG format

---

Create an index in the Azure portal

Before using an indexer to create an index, you'll first need to make your data available in a supported data source. Supported data sources include:

- Cosmos DB (SQL API)
- Azure SQL (database, managed instance, and SQL Server on an Azure VM)
- Azure Storage (Blob Storage, Table Storage, ADLS Gen2)

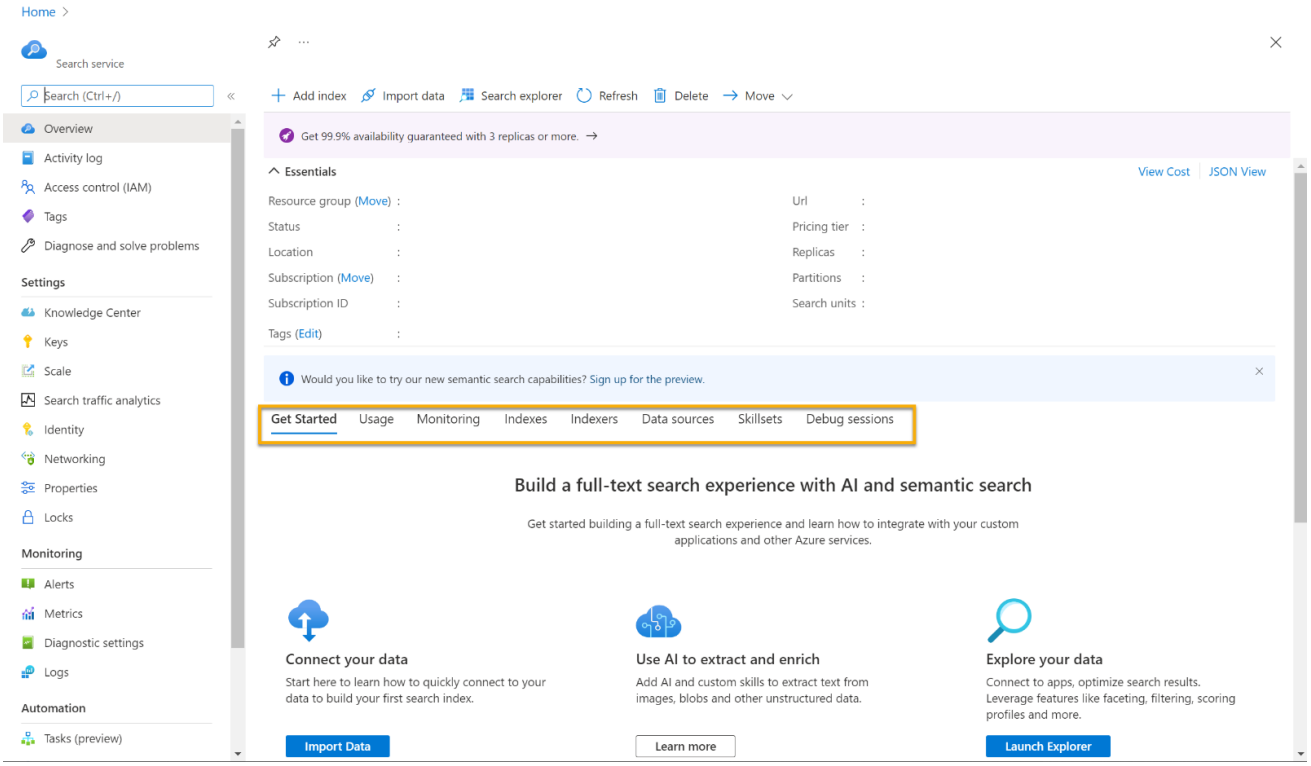
Using the Azure portal's Import data wizard

Once your data is in an Azure data source, you can begin using Azure Cognitive Search. Contained within the Azure Cognitive Search service in Azure portal is the Import data wizard, which automates processes in the Azure portal to create various objects needed for the search engine. You can see it in action when creating any of the following objects using the Azure portal:

- **Data Source**: Persists connection information to source data, including credentials. A data source object is used exclusively with indexers.
- **Index**: Physical data structure used for full text search and other queries.
- **Indexer**: A configuration object specifying a data source, target index, an optional AI skillset, optional schedule, and optional configuration settings for error handling and base-64 encoding.
- **Skillset**: A complete set of instructions for manipulating, transforming, and shaping content, including analyzing and extracting information from image files. Except for very simple and limited structures, it includes a reference to a Cognitive Services resource that provides enrichment.

- **Knowledge store**: Stores output from an AI enrichment pipeline in tables and blobs in Azure Storage for independent analysis or downstream processing.

To use Azure Cognitive Search, you'll need an Azure Cognitive Search resource. You can create a resource in the Azure portal. Once the resource is created, you can manage components of your service from the resource *Overview* page in the portal.



You can build Azure search indexes using the Azure portal or programmatically with the REST API or software development kits (SDKs).

Query data in an Azure Cognitive Search index

Index and query design are closely linked. After we build the index, we can perform queries. A crucial component to understand is that the schema of the index determines what queries can be answered.

Azure Cognitive Search queries can be submitted as an HTTP or REST API request, with the response coming back as JSON. Queries can specify what fields are searched and returned, how search results are shaped, and how the results should be filtered or sorted. A query that doesn't specify the field to search will execute against all the searchable fields within the index.

Azure Cognitive Search supports two types of syntax: simple and full Lucene. Simple syntax covers all of the common query scenarios, while full Lucene is useful for advanced scenarios.

Simple query requests

A query request is a list or words (search terms) and query operators (simple or full) of what you would like to see returned in a result set. Let's look what components make up a search query. Consider this simple search example:

Copy

```
coffee (-"busy" + "wifi")
```

This query is trying to find content about coffee, excluding busy and including wifi.

Breaking the query into components, it's made up of search terms, (coffee), plus two verbatim phrases, "busy" and "wifi", and operators (-, +, and ( )). The search terms can be matched in the search index in any order or location in the content. The two phrases will only match with exactly what is specified, so wi-fi would not be a match. Finally, a query can contain a number of operators. In this example, the - operator tells the search engine that these phrases should *NOT* be in the results. The parenthesis group terms together, and set their precedence.

By default, the search engine will match any of the terms in the query. Content containing just `coffee` would be a match. In this example, using `-"busy"` would lead to the search results including all content that doesn't have the exact string "busy" in it.

The simple query syntax in Azure Cognitive Search excludes some of the more complex features of the full Lucene query syntax, and it's the default search syntax for queries.

You can learn more about query syntax in the documentation.

Exercise-Create an Azure Cognitive Search solution

Let's imagine you work for Fourth Coffee, a national coffee chain. You're asked to help build a knowledge mining solution that makes it easy to search for insights about customer experiences. You decide to build an Azure Cognitive Search index using data extracted from customer reviews.

In this lab you'll:

- Create Azure resources
- Extract data from a data source
- Enrich data with AI skills
- Use Azure's indexer in the Azure portal
- Query your search index
- Review results saved to a Knowledge Store

Azure resources needed

The solution you'll create for Fourth Coffee requires the following resources in your Azure subscription:

- An **Azure Cognitive Search** resource, which will manage indexing and querying.
- A **Cognitive Services** resource, which provides AI services for skills that your search solution can use to enrich the data in the data source with AI-generated insights.

  Note

  Your Azure Cognitive Search and Cognitive Services resources must be in the same location!
- A **Storage account** with blob containers, which will store raw documents and other collections of tables, objects, or files.

Create an *Azure Cognitive Search* resource

1. Sign into the Azure portal.
2. Click the **+ Create a resource** button, search for *Azure Cognitive Search*, and create a **Azure Cognitive Search** resource with the following settings:
   - **Subscription**: *Your Azure subscription*.
   - **Resource group**: *Select or create a resource group with a unique name*.
   - **Service name**: *A unique name*
   - **Location**: *Choose any available region*
   - **Pricing tier**: Basic
3. Select **Review + create**, and after you see the response **Validation Passed**, select **Create**.
4. After deployment completes, select **Go to resource**. On the Azure Cognitive Search overview page, you can add indexes, import data, and search created indexes.

Create a Cognitive Services resource

You'll need to provision a **Cognitive Services** resource that's in the same location as your Azure Cognitive Search resource. Your search solution will use this resource to enrich the data in the datastore with AI-generated insights.

1. Return to the home page of the Azure portal, and then select the **+ Create a resource** button, search for *Cognitive Services*, and create a **Cognitive Services** resource with the following settings:

- **Subscription**: *Your Azure subscription*
- **Resource group**: *The same resource group as your Azure Cognitive Search resource*
- **Region**: *The same location as your Azure Cognitive Search resource*
- **Name**: *A unique name*
- **Pricing tier**: Standard S0

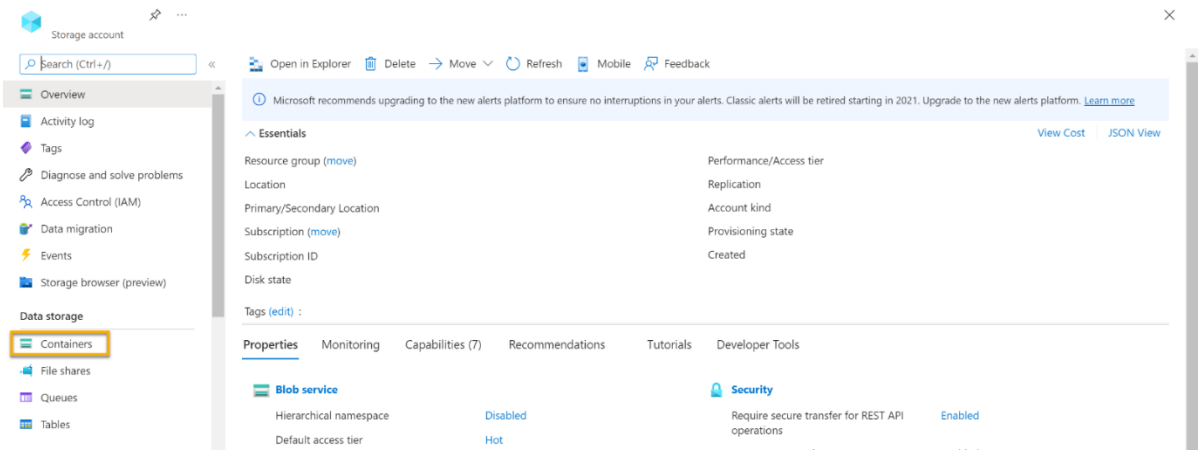
2. Select the required checkboxes, and then select **Review + create**.
3. After you see the response **Validation Passed**, select **Create**.
4. Wait for deployment to complete, then view the deployment details.

Create a storage account

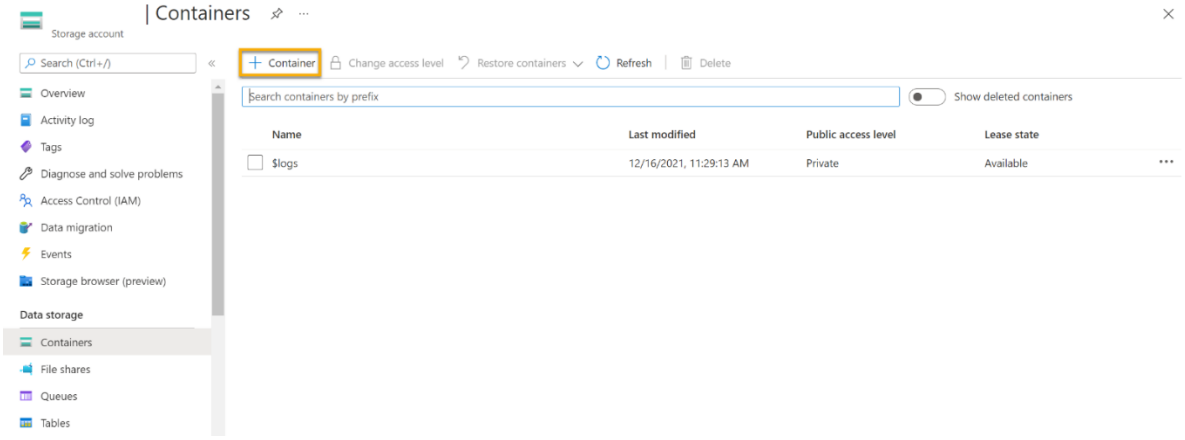
1. Return to the home page of the Azure portal, and then select the **+ Create a resource** button.
2. Search for *storage account*, and create a **Storage account** resource with the following settings:
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *The same resource group as your Azure Cognitive Search and Cognitive Services resources*
   - **Storage account name**: *A unique name*
   - **Location**: *Choose any available location*
   - **Performance**: Standard
   - **Redundancy**: Locally redundant storage (LRS)
3. Click **Review + Create** and then click **Create**. Wait for deployment to complete, and then go to the deployed resource.

Upload Documents to Azure Storage

1. Select **Go to resource** to access the Azure Storage account you created. In the left-hand menu pane, select **Containers**.



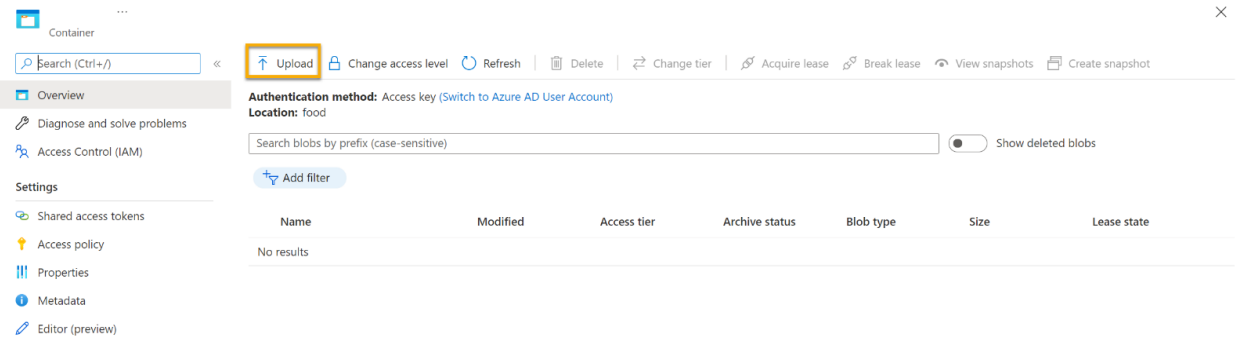
2. Select **+ Container**. A pane on your right-hand side opens.



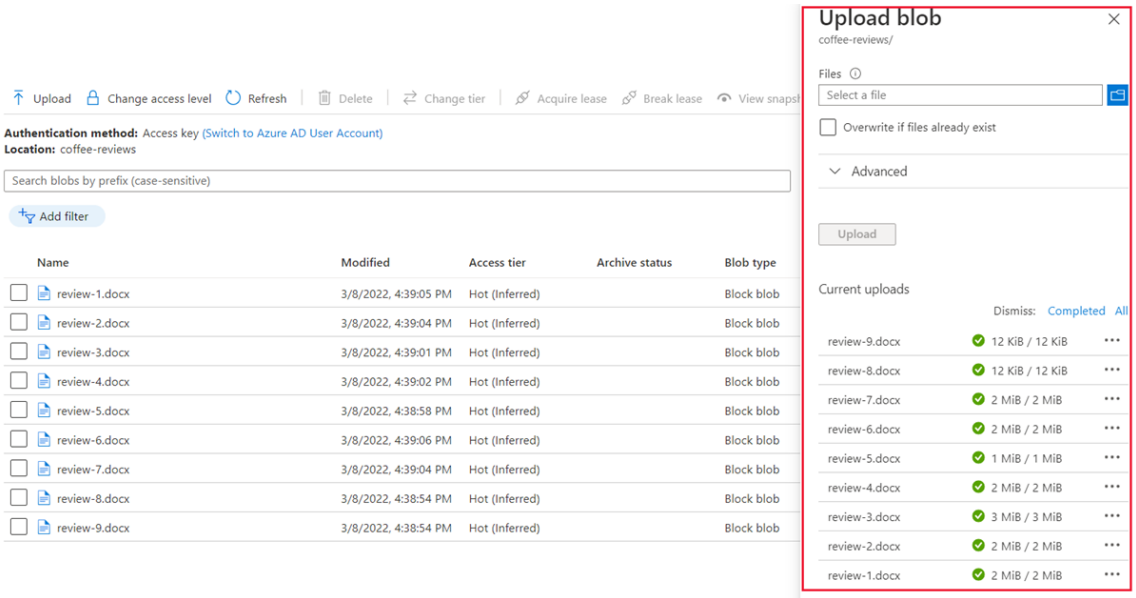
Enter the following settings:

- **Name**: coffee-reviews
- **Public access level**: Container (anonymous read access for containers and blobs)
- **Advanced**: *no changes*
- Click **Create**.

3. Download the zipped documents from https://aka.ms/km-documents. Then extract the files from *reviews* folder.
4. Select your *coffee-reviews* container. In the container, Select **Upload**.



5. In the Explorer window, select **all** the files in the *reviews* folder, select **Open**, and then select **Upload**.

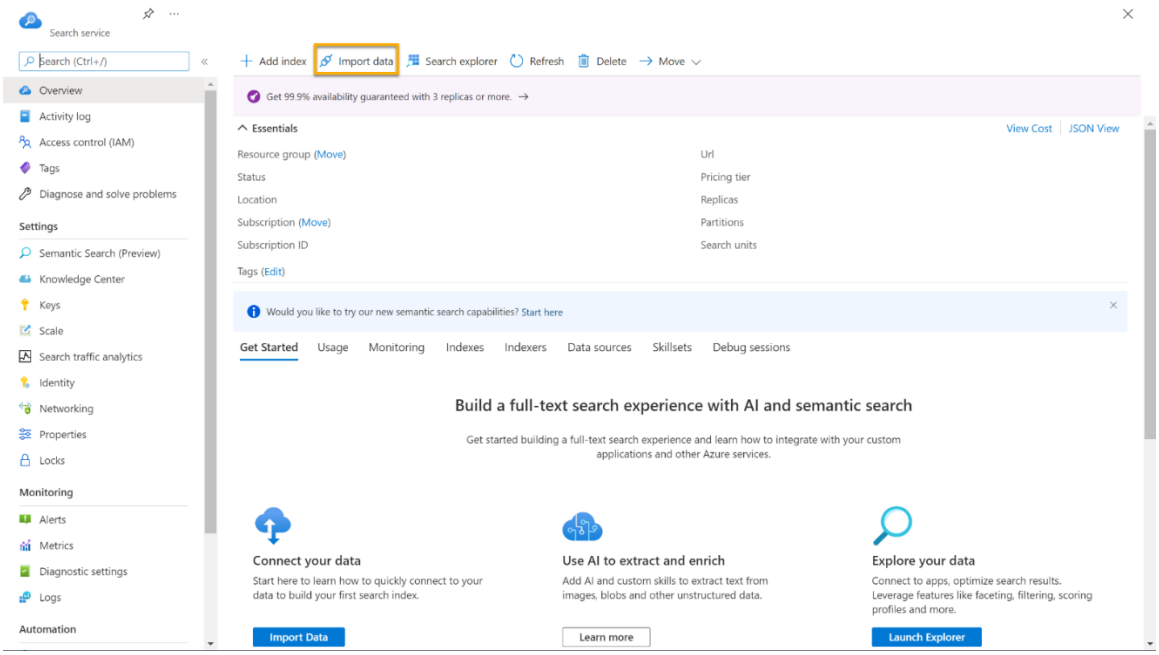


6. After the upload is complete, you can close the **Upload blob** pane. Your documents are now in your *coffee-reviews* storage container.

Index the documents

Once you have the documents in storage, you can use Azure Cognitive Search to extract insights from the documents. The Azure portal provides an *Import data wizard*. With this wizard, you can automatically create an index and indexer for supported data sources. You'll use the wizard to create an index, and import your search documents from storage into the Azure Cognitive Search index.

1. In the Azure portal, browse to your Azure Cognitive Search resource. On the **Overview** page, select **Import data**.



2. On the **Connect to your data** page, in the **Data Source** list, select **Azure Blob Storage**. Complete the data store details with the following values:
    - **Data Source**: *Choose Azure Blob Storage*
    - **Data source name**: coffee-customer-data
    - **Data to extract**: Content and metadata
    - **Parsing mode**: Default
    - **Connection string**: *Select **Choose an existing connection**. Select your storage account, select the **coffee-reviews** container, and then click **Select**.
    - **Managed identity authentication**: None
    - **Container name**: *this setting is auto-populated after you choose an existing connection*
    - **Blob folder**: *Leave this blank*
    - **Description**: Reviews for Fourth Coffee shops.
3. Select **Next: Add cognitive skills (Optional)**.
4. In the **Attach Cognitive Services** section, select your Cognitive Services resource.
5. In the **Add enrichments** section:
    - Change the **Skillset name** to **coffee-skillset**.
    - Select the checkbox **Enable OCR and merge all text into merged_content field**.

      Note

      It's important to select **Enable OCR** to see all of the enriched field options.
    - Ensure that the **Source data field** is set to **merged_content**.
    - Change the **Enrichment granularity level** to **Pages (5000 character chunks)**.
    - Don't select *Enable incremental enrichment*
    - Select the following enriched fields:

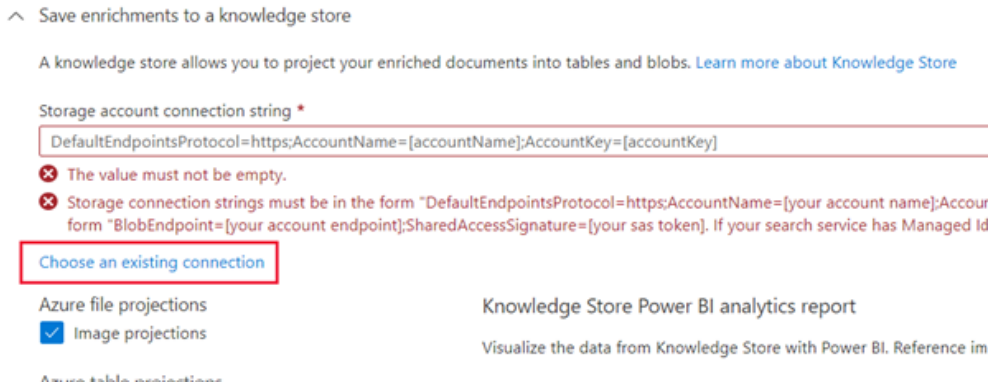
Cognitive Skill	Parameter	Field name
Extract location names		locations
Extract key phrases		keyphrases
Detect Sentiment		sentiment
Generate tags from images		imageTags

Cognitive Skill	Parameter	Field name
Generate captions from images		imageCaption

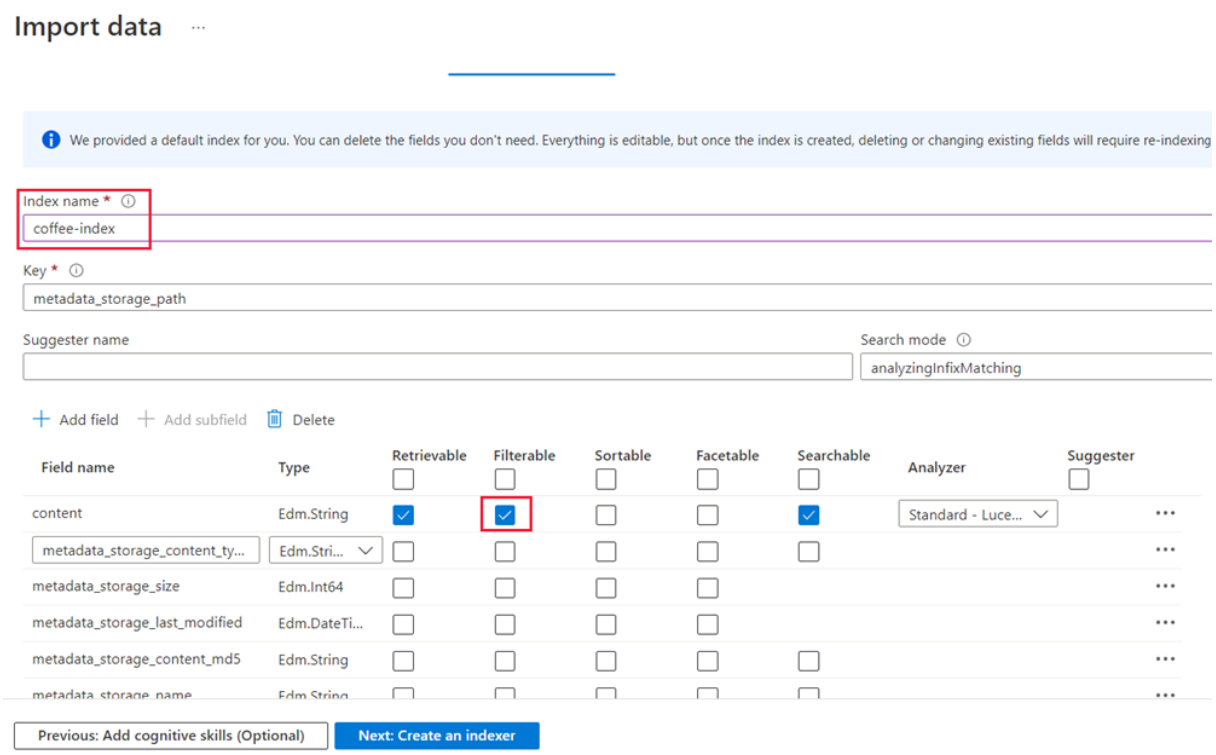
6. Under **Save Enrichments to a Knowledge Store**, select:
    - Image projections
    - Documents
    - Pages
    - Key phrases
    - Entities
    - Image details
    - Image references

    Note

    A warning asking for a **Storage Account Connection String** appears.



7. Select **Choose an existing connection**. Choose the storage account you created earlier.
8. Click on **+ Container** to create a new container called **knowledge-store** with the privacy level set to private.
9. Select **Create**.
10. Select the **knowledge-store** container, and then click **Select** at the bottom of the screen.
11. Select **Azure blob projects: Document**. A setting for *Container name* with the *knowledge-store* container auto-populated displays. Don't change the container name.
12. Select **Next: Customize Target Index**. Change the **Index name** to **coffee-index**.
13. Ensure that the **Key** is set to **metadata_storage_path**. Leave **Suggester name** blank and **Search mode** autopopulated.
14. Review the index fields' default settings. Select **filterable** for all the fields that are already selected by default.

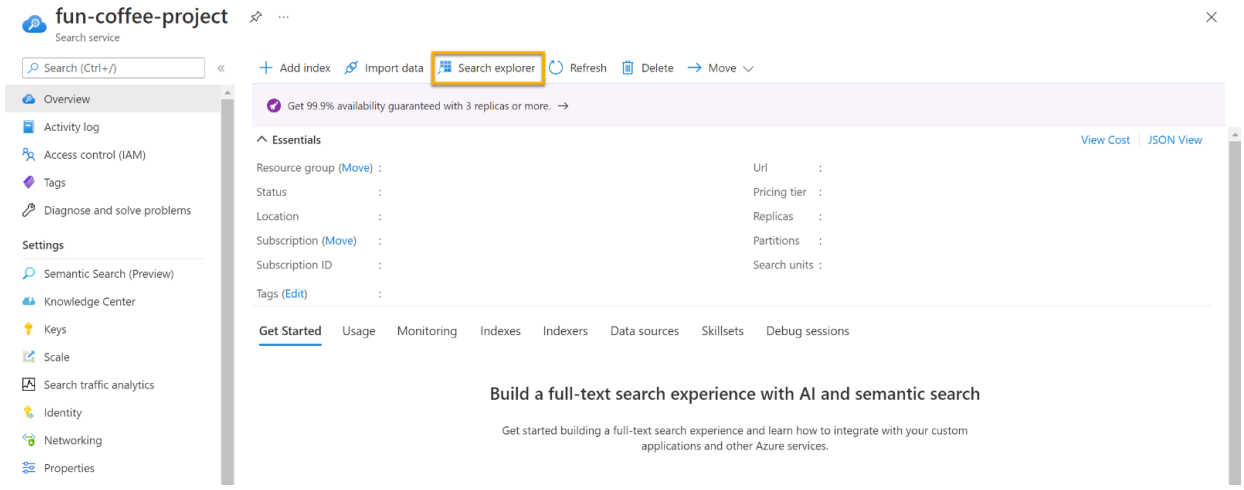


15. Select **Next: Create an indexer**.
16. Change the **Indexer name** to **coffee-indexer**.
17. Leave the **Schedule** set to **Once**.
18. Expand the **Advanced** options. Ensure that the **Base-64 Encode Keys** option is selected, as encoding keys can make the index more efficient.
19. Select **Submit** to create the data source, skillset, index, and indexer. The indexer is run automatically and runs the indexing pipeline, which:
    - Extracts the document metadata fields and content from the data source.
    - Runs the skillset of cognitive skills to generate more enriched fields.
    - Maps the extracted fields to the index.
20. In the bottom half of the **Overview** page for your Azure Cognitive Search resource, select the **Indexers** tab.This tab shows the newly created **coffee-indexer**. Wait a minute, and select ↻ **Refresh** until the **Status** indicates success.
21. Select the indexer name to see more details.

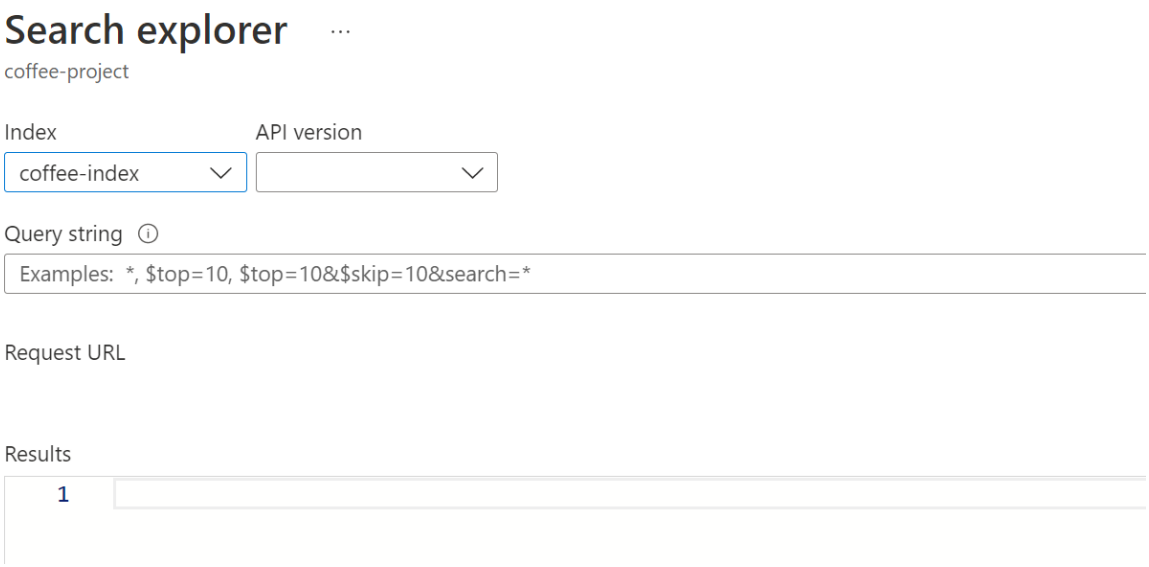
Query the index

Use the Search explorer to write and test queries. Search explorer is a tool built into the Azure portal that gives you an easy way to validate the quality of your search index. You can use Search explorer to write queries and review results in JSON.

1. In your Search service's *Overview* page, select **Search explorer** at the top of the screen.



2. Notice how the index selected is the *coffee-index* you created.



   In the **Query string** field, enter search=*&$count=true, and then select **Search**. The search query returns all the documents in the search index, including a count of all the documents in the **@odata.count** field. The search index should return a JSON document containing your search results.

3. Now let's filter by location. Enter search=$filter=locations eq 'Chicago'. The query searches all the documents in the index and filters for reviews with a Chicago location.

4. Now let's filter by sentiment. Enter search=$filter=sentiment eq 'negative' in the **Query string** field, and then select **Search**. The query searches all the documents in the index and filters for reviews with a negative sentiment.

   Note

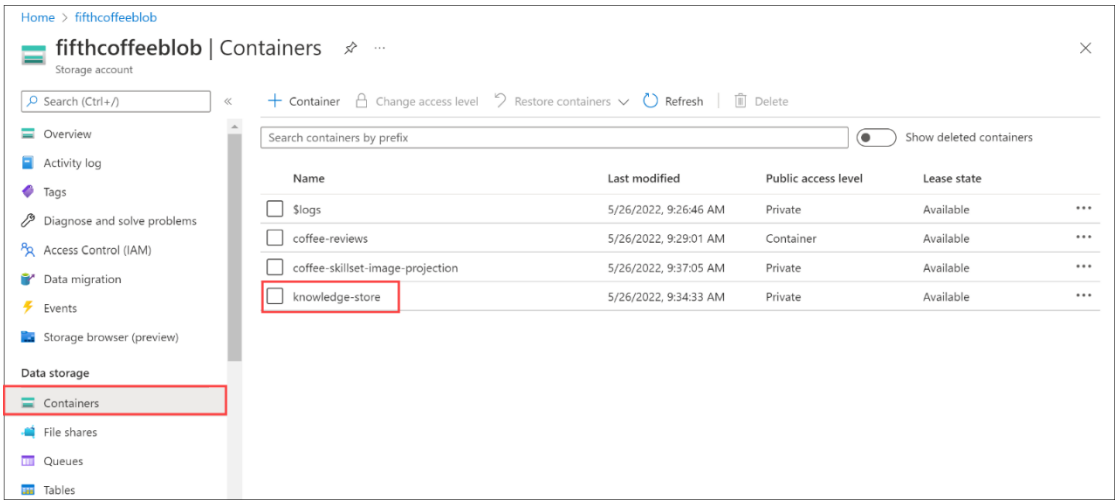
   See how the results are sorted by @search.score. This is the score assigned by the search engine to show how closely the results match the given query.

5. One of the problems we might want to solve for is why there might be certain reviews. Let's take a look at the key phrases associated with the negative review. What do you think might be the cause of the review?

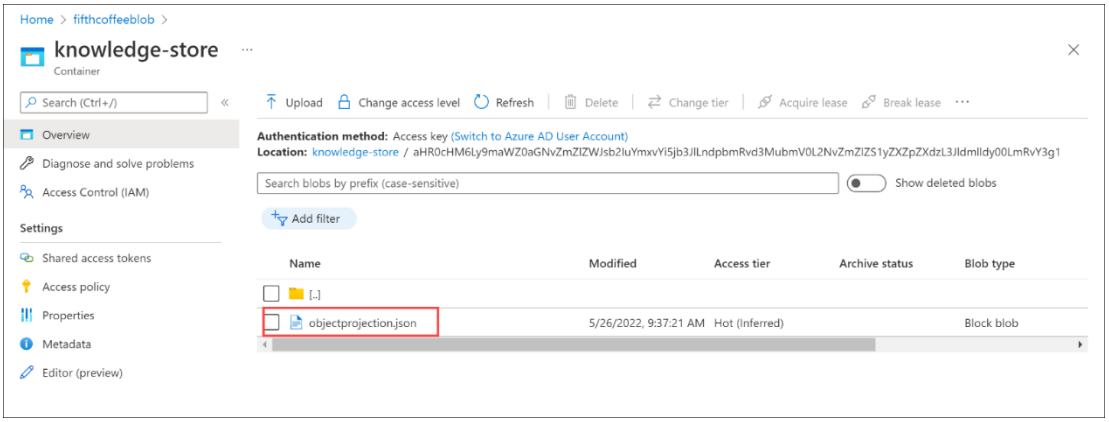
Review the knowledge store

Let's see the power of the knowledge store in action. When you ran the *Import data wizard*, you also created a knowledge store. Inside the knowledge store, you'll find the enriched data extracted by AI skills persists in the form of projections and tables.

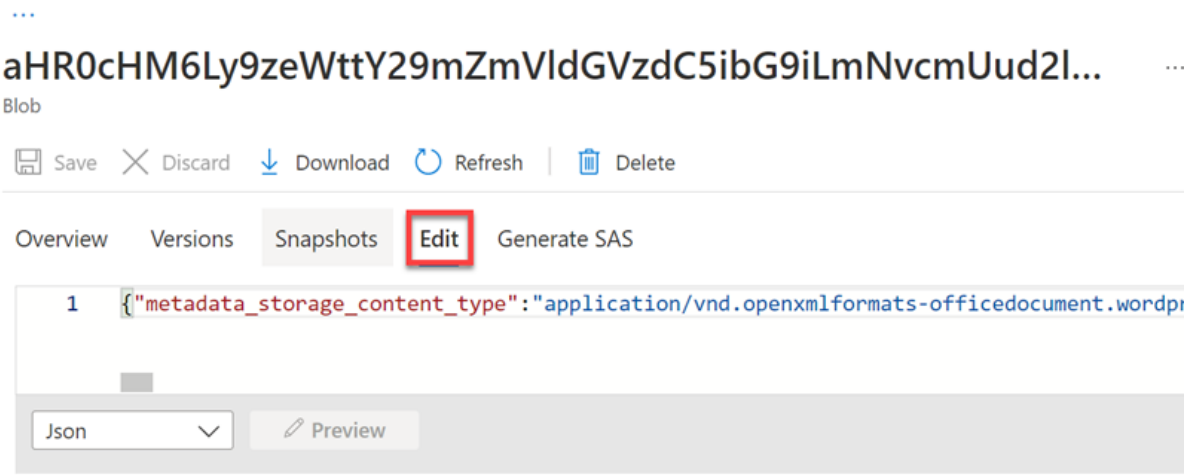
1. In the Azure portal, navigate back to your Azure storage account.
2. In the left-hand menu pane, select **Containers**. Select the **knowledge-store** container.



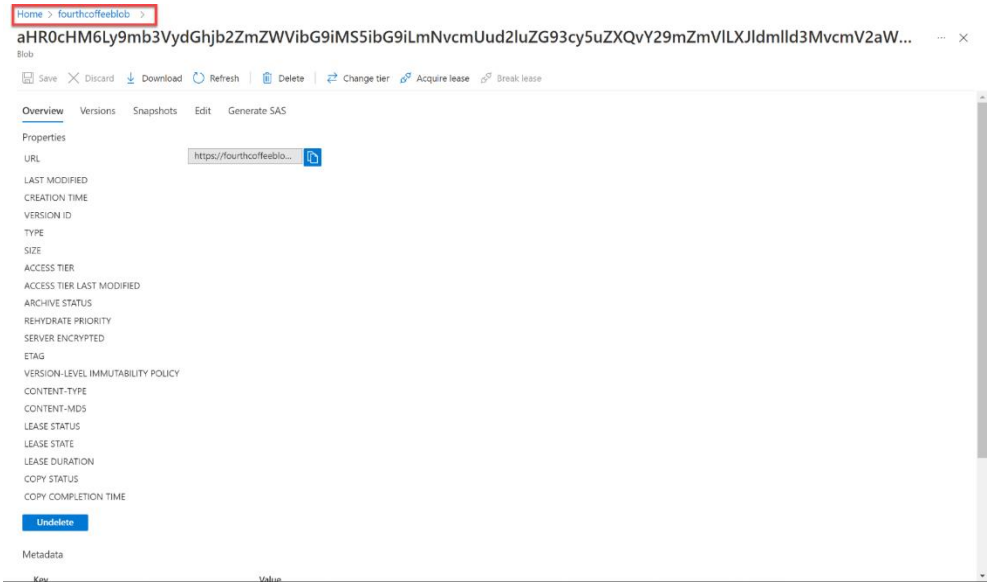
3. Select any of the items, and then click the **objectprojection.json** file.



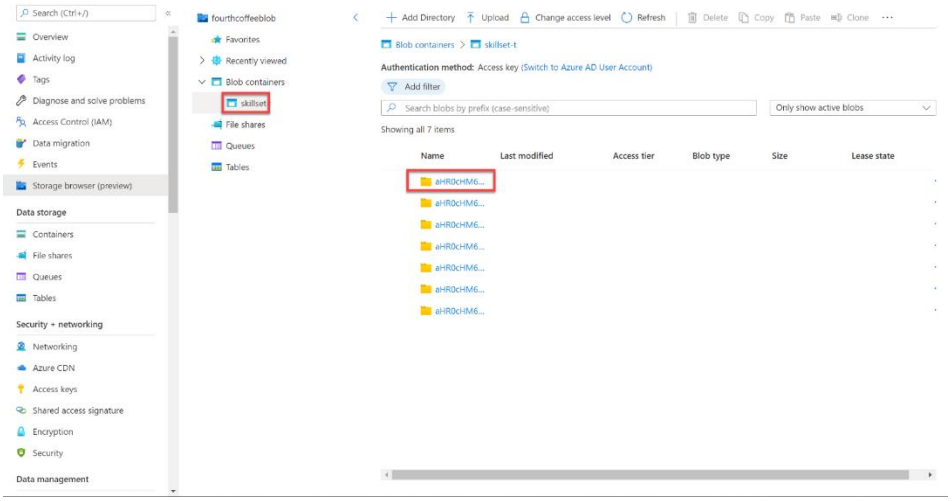
4. Select **Edit** to see the JSON produced for one of the documents from your Azure data store.



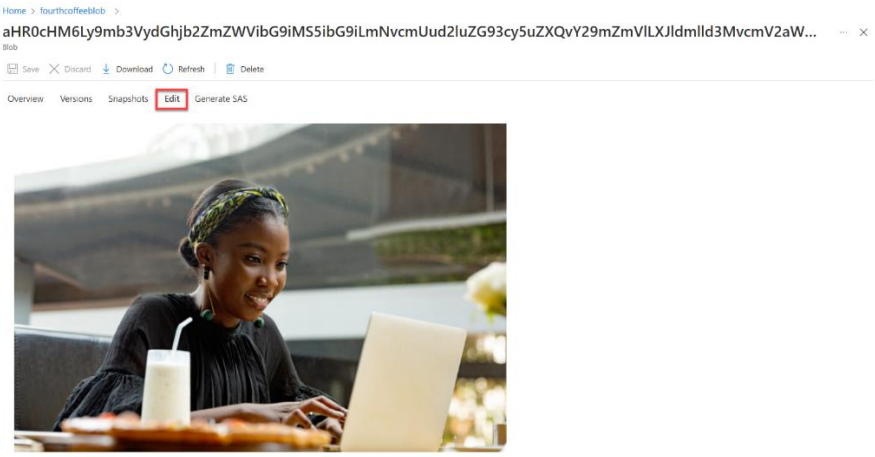
5. Select the storage blob breadcrumb at the top left of the screen to return to the Storage browser's tree.



6. Select **Blob containers** on the left-hand panel again. Select the container *coffee-skillset-image-projection*. Select any of the items.



7. Select any of the *.jpg* files. Select **Edit** to see the image stored from the document. Notice how all the images from the documents are stored in this manner.



8. Select the storage blob breadcrumb at the top left of the screen to return to the Storage browser's tree.
9. Select **Tables** on the left-hand panel. There's a table for each entity in the index. Select the table *coffeeSkillsetKeyPhrases*.

Look at the key phrases the knowledge store was able to capture from the content in the reviews. Many of the fields are keys, so you can link the tables like a relational database. The last field shows the key phrases that were extracted by the skillset.

Learn more

This simple search index only some of the capabilities of the Azure Cognitive Search service. To learn more about what you can do with this service, see the Azure Cognitive Search service page.

Check your knowledge

1. Which data format is accepted by Azure Cognitive Search when you're pushing data to the index?

- CSV.
- SQL.
- JSON.

Correct. Cognitive Search can index JSON documents. JSON is also used to define index schemas, indexers, and data source objects.

2. Which explanation best describes an indexer and an index?

- An indexer converts documents into JSON and forwards them to a search engine for indexing.

Correct. An indexer serializes a source document into JSON before passing it to a search engine for indexing. An indexer automates several steps of data ingestion, reducing the amount of code you need to write.

- An indexer can be used instead of an index if the files are already in the proper format.
- An indexer is only used for AI enrichment and skillset execution.

3. If you set up a search index without including a skillset, which would you still be able to query?

- Sentiment.
- Text content.

Correct. Cognitive Search is used for full text search over indexes containing alphanumeric content.

- Image captions.

Summary

In this module, you explored the benefits of using Azure Cognitive Search to create a scalable and rich search experience.

Module complete: