DISCLAIMER: THE CONTENT WAS TAKEN FROM THE MS LEARNING PATH FOR THE CERTIFICATE ON THE MS PAGE. THEREAFTER IT WAS EDITED AND FORMED TO THIS DOCUMENT.

# Microsoft Azure AI Fundamentals

# Lucia Harcekova

2022

Table of Contents

[Get started with artificial intelligence 9](#_Toc93671930)

[What is AI? 9](#_Toc93671931)

[Understand machine learning 9](#_Toc93671932)

[How machine learning works 9](#_Toc93671933)

[Machine learning in Microsoft Azure 9](#_Toc93671934)

[Understand computer vision 10](#_Toc93671935)

[Computer Vision models and capabilities 10](#_Toc93671936)

[Computer vision services in Microsoft Azure 10](#_Toc93671937)

[Understand natural language processing 11](#_Toc93671938)

[Natural language processing in Microsoft Azure 11](#_Toc93671939)

[Understand conversational AI 11](#_Toc93671940)

[Conversational AI in Microsoft Azure 11](#_Toc93671941)

[Try this 12](#_Toc93671942)

[Challenges and risks with AI 12](#_Toc93671943)

[Understand responsible AI 12](#_Toc93671944)

[Explore visual tools for machine learning 13](#_Toc93671945)

[Use automated machine learning in Azure Machine Learning 13](#_Toc93671946)

[What is machine learning? 13](#_Toc93671947)

[Create an Azure Machine Learning workspace 13](#_Toc93671948)

[Create an Azure Machine Learning workspace 14](#_Toc93671949)

[Create compute resources 14](#_Toc93671950)

[Create compute targets 15](#_Toc93671951)

[Explore data 15](#_Toc93671952)

[Create a dataset 15](#_Toc93671953)

[Train a machine learning model 16](#_Toc93671954)

[Run an automated machine learning experiment 16](#_Toc93671955)

[Review the best model 18](#_Toc93671956)

[Deploy a model as a service 20](#_Toc93671957)

[Deploy a predictive service 21](#_Toc93671958)

[Test the deployed service 21](#_Toc93671959)

[Create a Regression Model with Azure Machine Learning designer 23](#_Toc93671960)

[Create an Azure Machine Learning workspace 23](#_Toc93671961)

[Create an Azure Machine Learning workspace 23](#_Toc93671962)

[Create compute resources 23](#_Toc93671963)

[Explore data 23](#_Toc93671964)

[Create a pipeline 23](#_Toc93671965)

[Add and explore a dataset 24](#_Toc93671966)

[Add data transformations 24](#_Toc93671967)

[Run the pipeline 25](#_Toc93671968)

[View the transformed data 26](#_Toc93671969)

[Create and run a training pipeline 26](#_Toc93671970)

[Add training modules 26](#_Toc93671971)

[Run the training pipeline 27](#_Toc93671972)

[Evaluate a regression model 28](#_Toc93671973)

[Add an Evaluate Model module 28](#_Toc93671974)

[Create an inference pipeline 29](#_Toc93671975)

[Create and run an inference pipeline 29](#_Toc93671976)

[Deploy a predictive service 31](#_Toc93671977)

[Deploy a service 31](#_Toc93671978)

[Test the service 31](#_Toc93671979)

[Create a classification model with Azure Machine Learning designer 33](#_Toc93671980)

[Create an Azure Machine Learning workspace 33](#_Toc93671981)

[Create compute resources 33](#_Toc93671982)

[Explore data 33](#_Toc93671983)

[Create a dataset 33](#_Toc93671984)

[Create a pipeline 34](#_Toc93671985)

[Add Transformations 34](#_Toc93671986)

[Run the pipeline 35](#_Toc93671987)

[View the transformed data 35](#_Toc93671988)

[Create and run a training pipeline 35](#_Toc93671989)

[Add training modules 36](#_Toc93671990)

[Run the training pipeline 37](#_Toc93671991)

[Evaluate a classification model 37](#_Toc93671992)

[Add an Evaluate Model module 37](#_Toc93671993)

[Create an inference pipeline 39](#_Toc93671994)

[Deploy a predictive service 41](#_Toc93671995)

[Deploy a service 41](#_Toc93671996)

[Test the service 41](#_Toc93671997)

[Create a Clustering Model with Azure Machine Learning designer 43](#_Toc93671998)

[Create an Azure Machine Learning workspace 43](#_Toc93671999)

[Create compute resources 43](#_Toc93672000)

[Explore data 43](#_Toc93672001)

[Create a dataset 43](#_Toc93672002)

[Create a pipeline 44](#_Toc93672003)

[Apply transformations 45](#_Toc93672004)

[Run the pipeline 46](#_Toc93672005)

[View the transformed data 47](#_Toc93672006)

[Create and run a training pipeline 47](#_Toc93672007)

[Add training modules 47](#_Toc93672008)

[Run the training pipeline 49](#_Toc93672009)

[Evaluate a clustering model 49](#_Toc93672010)

[Add an Evaluate Model module 49](#_Toc93672011)

[Create an inference pipeline 50](#_Toc93672012)

[Deploy a predictive service 52](#_Toc93672013)

[Deploy a service 52](#_Toc93672014)

[Test the service 52](#_Toc93672015)

[Explore computer vision 53](#_Toc93672016)

[Analyse images with the Computer Vision service 53](#_Toc93672017)

[Get started with image analysis on Azure 54](#_Toc93672018)

[Azure resources for Computer Vision 54](#_Toc93672019)

[Analyzing images with the Computer Vision service 54](#_Toc93672020)

[Describing an image 55](#_Toc93672021)

[Tagging visual features 55](#_Toc93672022)

[Detecting objects 55](#_Toc93672023)

[Detecting brands 55](#_Toc93672024)

[Detecting faces 55](#_Toc93672025)

[Categorizing an image 56](#_Toc93672026)

[Detecting domain-specific content 56](#_Toc93672027)

[Optical character recognition 56](#_Toc93672028)

[Additional capabilities 56](#_Toc93672029)

[Exercise - Analyze images with the Computer Vision service 56](#_Toc93672030)

[Create a Cognitive Services resource 56](#_Toc93672031)

[Run Cloud Shell 57](#_Toc93672032)

[Configure and run a client application 57](#_Toc93672033)

[Classify images with the Custom Vision service 59](#_Toc93672034)

[Uses of image classification 59](#_Toc93672035)

[Understand classification 59](#_Toc93672036)

[Understand image classification 59](#_Toc93672037)

[Azure's Custom Vision service 59](#_Toc93672038)

[Get started with image classification on Azure 60](#_Toc93672039)

[Azure resources for Custom Vision 60](#_Toc93672040)

[Model training 60](#_Toc93672041)

[Model evaluation 61](#_Toc93672042)

[Using the model for prediction 61](#_Toc93672043)

[Exercise - Create an image classification solution 61](#_Toc93672044)

[Create a Cognitive Services resource 61](#_Toc93672045)

[Create a Custom Vision project 62](#_Toc93672046)

[Test the model 63](#_Toc93672047)

[Publish the image classification model 63](#_Toc93672048)

[Run Cloud Shell 63](#_Toc93672049)

[Configure and run a client application 63](#_Toc93672050)

[Detect objects in images with the Custom Vision service 64](#_Toc93672051)

[Uses of object detection 65](#_Toc93672052)

[What is object detection? 65](#_Toc93672053)

[Get started with object detection on Azure 65](#_Toc93672054)

[Azure resources for Custom Vision 65](#_Toc93672055)

[Image tagging 66](#_Toc93672056)

[Model training and evaluation 66](#_Toc93672057)

[Using the model for prediction 66](#_Toc93672058)

[Exercise - Create an object detection solution 67](#_Toc93672059)

[Create a Cognitive Services resource 67](#_Toc93672060)

[Create a Custom Vision project 67](#_Toc93672061)

[Add and tag images 68](#_Toc93672062)

[Train and test a model 69](#_Toc93672063)

[Publish the object detection model 69](#_Toc93672064)

[Run Cloud Shell 70](#_Toc93672065)

[Configure and run a client application 70](#_Toc93672066)

[Detect and analyse faces with the Face service 70](#_Toc93672067)

[Get started with Face analysis on Azure 70](#_Toc93672068)

[Face 71](#_Toc93672069)

[Azure resources for Face 71](#_Toc93672070)

[Tips for more accurate results 72](#_Toc93672071)

[Exercise - Detect and analyze faces with the Face service 72](#_Toc93672072)

[Create a Cognitive Services resource 72](#_Toc93672073)

[Run Cloud Shell 72](#_Toc93672074)

[Configure and run a client application 72](#_Toc93672075)

[Read text with the Computer Vision service 73](#_Toc93672076)

[Uses of OCR 73](#_Toc93672077)

[Get started with OCR on Azure 73](#_Toc93672078)

[Azure resources for Computer Vision 74](#_Toc93672079)

[Use the Computer Vision service to read text 74](#_Toc93672080)

[The OCR API 74](#_Toc93672081)

[The Read API 74](#_Toc93672082)

[Exercise - Read text with the Computer Vision service 75](#_Toc93672083)

[Use the Computer Vision Service to Read Text in an Image 75](#_Toc93672084)

[Create a Cognitive Services resource 75](#_Toc93672085)

[Run Cloud Shell 75](#_Toc93672086)

[Configure and run a client application 75](#_Toc93672087)

[Analyse receipts with the Form Recognizer service 76](#_Toc93672088)

[Get started with receipt analysis on Azure 76](#_Toc93672089)

[Azure resources to access Form Recognizer services 77](#_Toc93672090)

[Using the pre-built receipt model 77](#_Toc93672091)

[Exercise - Analyze receipts with Form Recognizer 77](#_Toc93672092)

[Create a Cognitive Services resource 78](#_Toc93672093)

[Run Cloud Shell 78](#_Toc93672094)

[Configure and run a client application 78](#_Toc93672095)

[Explore natural language processing 79](#_Toc93672096)

[Analyse text with the Language service 79](#_Toc93672097)

[Text Analytics Techniques 79](#_Toc93672098)

[Get started with text analysis 80](#_Toc93672099)

[Azure resources for the Language service 80](#_Toc93672100)

[Language detection 80](#_Toc93672101)

[Ambiguous or mixed language content 81](#_Toc93672102)

[Sentiment analysis 81](#_Toc93672103)

[Key phrase extraction 81](#_Toc93672104)

[Entity recognition 82](#_Toc93672105)

[Exercise - Analyze text 83](#_Toc93672106)

[Create a Cognitive Services resource 83](#_Toc93672107)

[Get the Key and Location for your Cognitive Services resource 83](#_Toc93672108)

[Run Cloud Shell 83](#_Toc93672109)

[Configure and run a client application 84](#_Toc93672110)

[Recognize and synthesize speech 85](#_Toc93672111)

[Speech recognition 85](#_Toc93672112)

[Speech synthesis 85](#_Toc93672113)

[Get started with speech on Azure 86](#_Toc93672114)

[Azure resources for the Speech service 86](#_Toc93672115)

[The speech-to-text API 86](#_Toc93672116)

[Real-time transcription 86](#_Toc93672117)

[Batch transcription 86](#_Toc93672118)

[The text-to-speech API 86](#_Toc93672119)

[Speech synthesis voices 86](#_Toc93672120)

[Supported Languages 87](#_Toc93672121)

[Exercise - Use the Speech service 87](#_Toc93672122)

[Create a Cognitive Services resource 87](#_Toc93672123)

[Get the Key and Location for your Cognitive Services resource 87](#_Toc93672124)

[Run Cloud Shell 87](#_Toc93672125)

[Configure and run a client application 88](#_Toc93672126)

[Translate text and speech 88](#_Toc93672127)

[Literal and semantic translation 88](#_Toc93672128)

[Text and speech translation 89](#_Toc93672129)

[Get started translation in Azure 89](#_Toc93672130)

[Azure resources for Translator Text and Speech 89](#_Toc93672131)

[Text translation with the Translator Text service 89](#_Toc93672132)

[Optional Configurations 89](#_Toc93672133)

[Speech translation with the Speech service 90](#_Toc93672134)

[Speech service language support 90](#_Toc93672135)

[Exercise - Translate text and speech 90](#_Toc93672136)

[Create a Cognitive Services resource 90](#_Toc93672137)

[Get the Key and Location for your Cognitive Services resource 91](#_Toc93672138)

[Run Cloud Shell 91](#_Toc93672139)

[Configure and run a client application 91](#_Toc93672140)

[Create a language model with Conversational Language Understanding 92](#_Toc93672141)

[Utterances 92](#_Toc93672142)

[Entities 92](#_Toc93672143)

[Intents 92](#_Toc93672144)

[Getting started with Conversational Language Understanding 93](#_Toc93672145)

[Azure resources for Conversational Language Understanding 93](#_Toc93672146)

[Authoring 94](#_Toc93672147)

[Creating intents 94](#_Toc93672148)

[Creating entities 94](#_Toc93672149)

[Training the model 94](#_Toc93672150)

[Predicting 94](#_Toc93672151)

[Exercise - Create a Conversational Language Understanding application 95](#_Toc93672152)

[Create a Language service resource 95](#_Toc93672153)

[Create a Conversational Language Understanding App 95](#_Toc93672154)

[Create intents and entities 96](#_Toc93672155)

[Train the model 97](#_Toc93672156)

[Deploy and test the model 97](#_Toc93672157)

[Run Cloud Shell 98](#_Toc93672158)

[Configure and run a client application 98](#_Toc93672159)

[Explore conversational AI 99](#_Toc93672160)

[Conversational AI 99](#_Toc93672161)

[Get started with the Language service and Azure Bot Service 100](#_Toc93672162)

[Creating a custom question answering knowledge base 100](#_Toc93672163)

[Provision a Language service Azure resource 100](#_Toc93672164)

[Define questions and answers 100](#_Toc93672165)

[Test the knowledge base 100](#_Toc93672166)

[Use the knowledge base 101](#_Toc93672167)

[Build a bot with the Azure Bot Service 101](#_Toc93672168)

[Create a bot for your knowledge base 101](#_Toc93672169)

[Extend and configure the bot 101](#_Toc93672170)

[Connect channels 101](#_Toc93672171)

[Exercise - Create a bot 101](#_Toc93672172)

[Create a custom question answering knowledge base 102](#_Toc93672173)

[Edit the knowledge base 103](#_Toc93672174)

[Train and test the knowledge base 103](#_Toc93672175)

[Create a bot for the knowledge base 103](#_Toc93672176)

[Explore decision support 104](#_Toc93672177)

[Introduction to Anomaly Detector 104](#_Toc93672178)

[What is Anomaly Detector? 104](#_Toc93672179)

[Azure's Anomaly Detector service 105](#_Toc93672180)

[How Anomaly Detector works 105](#_Toc93672181)

[Data format 105](#_Toc93672182)

[Data consistency recommendations 106](#_Toc93672183)

[When to use Anomaly Detector 106](#_Toc93672184)

[Batch detection 106](#_Toc93672185)

[Real-time detection 106](#_Toc93672186)

[Exercise 107](#_Toc93672187)

[Create an Anomaly Detector resource 107](#_Toc93672188)

[Run Cloud Shell 107](#_Toc93672189)

[Configure and run a client application 107](#_Toc93672190)

[Explore knowledge mining 108](#_Toc93672191)

[Introduction to Azure Cognitive Search 108](#_Toc93672192)

[What is Azure Cognitive Search? 108](#_Toc93672193)

[What can Azure Cognitive Search do? 109](#_Toc93672194)

[Importing content to storage 109](#_Toc93672195)

[Creating an Azure Cognitive Search resource 109](#_Toc93672196)

[Use the Azure portal to create an index 109](#_Toc93672197)

[Design an index 110](#_Toc93672198)

[Index schema 110](#_Toc93672199)

[Index attributes 111](#_Toc93672200)

[Field behaviors 111](#_Toc93672201)

[Use indexer to build index 111](#_Toc93672202)

[Use the pull method to load data with an indexer 112](#_Toc93672203)

[Data import monitoring and verification 112](#_Toc93672204)

[Index data with Azure Cognitive Search 112](#_Toc93672205)

[Making changes to an index 113](#_Toc93672206)

[Enhance Azure Cognitive Search indexes with skills 113](#_Toc93672207)

[Query data in an Azure Cognitive Search index 113](#_Toc93672208)

[Search content using Azure Cognitive Search 113](#_Toc93672209)

[Query request Elements and Parsing 113](#_Toc93672210)

[Simple query syntax 114](#_Toc93672211)

[Handle results 115](#_Toc93672212)

[Exercise 115](#_Toc93672213)

[Create an Azure Cognitive Search resource 115](#_Toc93672214)

[Run Cloud Shell 115](#_Toc93672215)

[Review content 116](#_Toc93672216)

[Load content into Azure blob storage 116](#_Toc93672217)

[Create an Azure Cognitive Search index 117](#_Toc93672218)

[Query the index 119](#_Toc93672219)

[Sources 120](#_Toc93672220)

# 

# Get started with artificial intelligence

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!

What is AI?

AI is the creation of software that imitates human behaviours and capabilities. Key elements 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.
* **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.
* **Conversational AI** - The capability of a software "agent" to participate in a conversation.

Understand machine learning

Machine Learning is the foundation for most AI solutions. 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.

## How machine learning works

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 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.

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

MS Azure provides the **Azure ML** service - a cloud-based platform for creating, managing, and publishing machine learning models. Azure ML 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 computer vision

Computer Vision is an area of AI that deals with visual processing. The **Seeing AI** app is a great example of the power of pc 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.

## 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 | It involves training a machine learning model to classify images based on their contents. For example, in a traffic monitoring you use an image classification model to classify images based on the type of vehicle they contain, such as taxis, buses, ... |
| Object detection | An image of a street with buses, cars, and cyclists identified and highlighted with a bounding boxObject 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 | An image of a street with the pixels belonging to buses, cars, and cyclists identifiedSemantic 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. |
| Image analysis | An image of a person with a dog on a street and the caption "A person with a dog on a street"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 | An image of multiple people on a city street with their faces highlightedFace 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. |
| 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 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** | Use service to analyse images and video,extract descriptions, tags, objects, 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. |

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.

Ex. Starship, is a VR game, that takes place in a fiction world. It uses natural language processing to enable players to control the narrative and interact with in-game characters and starship systems.

## Natural language processing in Microsoft Azure

You can use the following cognitive services to build natural language processing solutions:

| Service | Capabilities |
| --- | --- |
| **Text Analytics** | Use this service to analyze text documents and extract key phrases, detect entities (such as places, dates, and people), and evaluate sentiment (how positive or negative a document is). |
| **Translator Text** | Use this service to translate text between more than 60 languages. |
| **Speech** | To recognize and synthesize speech, and to translate spoken languages. |
| **Language Understanding Intelligent Service (LUIS**) | Use this service to train a language model that can understand spoken or text-based commands. |

Understand conversational AI

Conversational AI is the term used to describe solutions where AI agents participate in conversations with humans. Most commonly, conversational AI solutions use bots to manage dialogs with users. These dialogs can take place through web site interfaces, email, social media platforms, ....

Bots can be the basis of AI solutions for:

* Customer support for products or services.
* Reservation systems for restaurants, airlines, cinemas, and other appointment based businesses.
* Health care consultations and self-diagnosis.
* Home automation and personal digital assistants.

## Conversational AI in Microsoft Azure

To create conversational AI solutions on Microsoft Azure, you can use the following services:

| Service | Capabilities |
| --- | --- |
| **QnA Maker** | This cognitive service enables you to quickly build a knowledge base of questions and answers that can form the basis of a dialog between a human and an AI agent. |
| **Azure Bot Service** | This service provides a platform for creating, publishing, and managing bots. Developers can use the Bot Framework to create a bot and manage it with Azure Bot Service - integrating back-end services like QnA Maker and LUIS, and connecting to channels for web chat, email, Microsoft Teams, and others. |

## Try this

The Microsoft healthcare bot is built on Azure Bot Service and enables developers to quickly create conversational AI solutions for health care.

Challenges and risks with AI

AI is a powerful tool that can be used to greatly benefit the world. However, like any tool, it must be used responsibly. The table shows some of the potential challenges risks facing an AI app 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 MS, AI software development is guided by a set of six principles, designed to ensure that AI apps provide amazing solutions to difficult problems without any unintended negative consequences.

* **Fairness:** AI systems should treat all people fairly. Ex. 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.
* **Reliability and safety:** AI systems should perform reliably and safely. Ex. consider an AI-based sw system for an autonomous vehicle; or ML model that diagnoses patient symptoms and recommends prescriptions. Unreliability in these kinds of system can result in substantial risk to human life. AI-based software app development must be subjected to rigorous testing and deployment management processes to ensure that they work as expected before release.
* **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.
* **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.
* **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.
* **Accountability:** People should be accountable for AI systems. Designers and developers of AI-based solution should work within a framework of governance and organizational principles that ensure the solution meets ethical and legal standards that are clearly defined. The principles of responsible AI can help you understand some of the challenges facing developers as they try to create ethical AI solutions.

Explore visual tools for machine learning

Use automated machine learning in Azure Machine Learning

*Machine Learning* is the foundation for most AI solutions, and the creation of an intelligent solution often begins with the use of ML 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 ML models.

What is machine learning?

ML is a technique that uses mathematics and statistics to create a model that can predict unknown values. Ex. suppose Cycles is a business that rents cycles. 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, *Cycles* could create a ML model that takes information about a specific day (the day of week, weather conditions, …) 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 ML 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.

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.

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 ML, 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 ML workloads.

This module is one of many that make use of an Azure Machine Learning workspace, including the other modules in the [**Create no-code predictive models with Azure Machine Learning**](https://docs.microsoft.com/en-us/learn/paths/create-no-code-predictive-models-azure-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](https://portal.azure.com/) using the MS credentials associated with your Azure subscription.
2. Select **＋Create a resource**, search for Machine Learning, and create a new **Machine Learning** resource 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 geographical region closest to you
   * **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. 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](https://ml.azure.com/)), and sign into Azure Machine Learning studio using your Microsoft account. If prompted, select your Azure directory and subscription, and your Azure Machine Learning workspace.
5. In Azure Machine Learning studio, toggle the ☰ 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

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

## 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](https://ml.azure.com/), 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.

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. On the **Compute Instances** tab, add a new compute instance with the following settings. You'll use this as a workstation from which to test your model:
   * **Compute name**: enter a unique name
   * **Virtual Machine type**: CPU
   * **Virtual Machine size**:
     + Choose **Select from all options**
     + Search for and select **Standard\_DS11\_v2**
2. While the compute instance is being created, switch to the **Compute Clusters** tab, and add a new compute cluster with the following settings. You'll use this to train a ML model:
   * **Location**: Select the same as your workspace. If that location is not listed, choose the one closest to you
   * **Virtual Machine priority**: Dedicated
   * **Virtual Machine type**: CPU
   * **Virtual Machine size**:
     + Choose **Select from all options**
     + Search for and select **Standard\_DS11\_v2**
   * **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

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](https://ml.azure.com/), view the **Datasets** page. Datasets represent specific data files or tables that you plan to work with in Azure ML.
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*](https://www.capitalbikeshare.com/system-data) and is used in accordance with the published data [*license agreement*](https://www.capitalbikeshare.com/data-license-agreement).

Train a machine learning model

Azure Machine Learning includes an automated machine learning capability that leverages the scalability of cloud compute to automatically try multiple pre-processing techniques and model-training algorithms in parallel to find the best performing supervised machine learning model for your data.

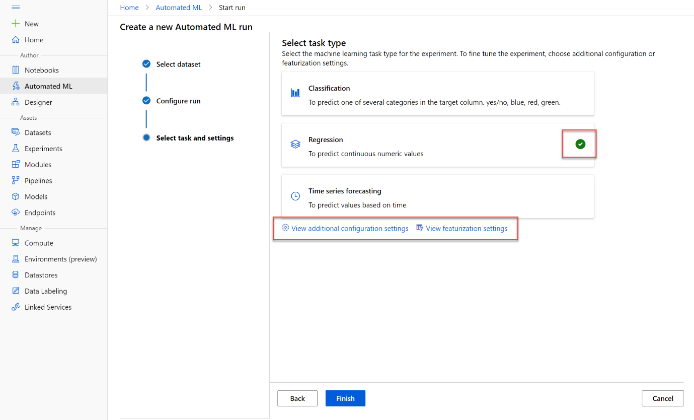
The automated machine learning capability in Azure Machine Learning supports supervised machine learning models - in other words, models for which the training data includes known label values. You can use automated machine learning to train models for:

* **Classification** (predicting categories or classes)
* **Regression** (predicting numeric values)
* **Time series forecasting** (regression with a time-series element, enabling you to predict numeric values at a future point in time)

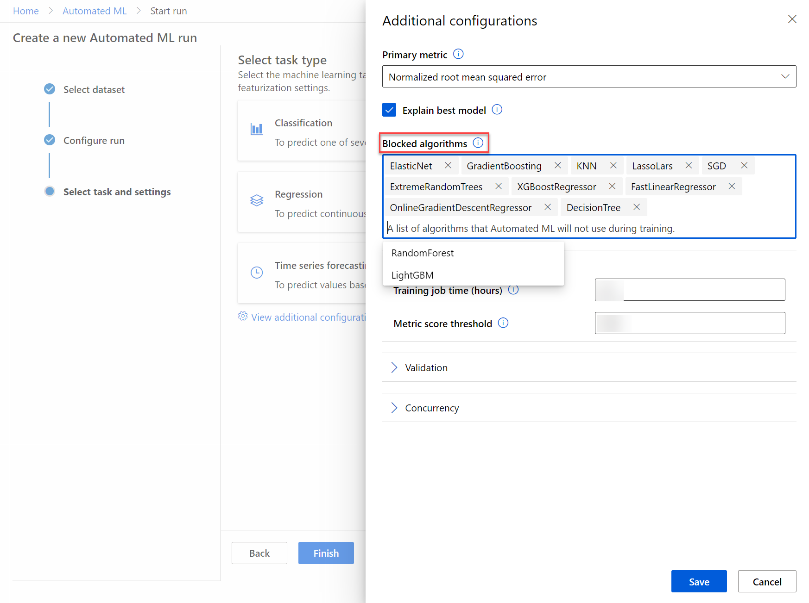
## Run an automated machine learning experiment

In Azure ML, operations that you run are called experiments. Follow the steps below to run an experiment that uses automated ML to train a regression model that predicts bicycle rentals.

1. In [Azure Machine Learning studio](https://ml.azure.com/), view the **Automated ML** page (under **Author**).
2. Create a new 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 the model will be trained to predict)
     + **Select compute cluster**: the compute cluster you created previously
   * **Select task and settings**:
     + **Task type**: Regression (the model will predict a numeric value)

**

* + **Additional configuration settings:**
    - **Primary metric**: Select **Normalized root mean squared error**
    - **Explain best model**: Selected - this option causes automated machine learning to calculate feature importance for the best model; making it possible to determine the influence of each feature on the predicted label.
    - **Blocked algorithms**: Block ***all*** other than ***RandomForest*** and ***LightGBM*** - normally you'd want to try as many as possible, but doing so can take a long time!
    - **Exit criterion**:
      * **Training job time (hours)**: 0.5 - this causes the experiment to end after a maximum of 30 minutes.
      * **Metric score threshold**: 0.08 - this causes the experiment to end if a model achieves a normalized root mean squared error metric score of 0.08 or less.

**

* + **Featurization settings:**
    - **Enable featurization**: Selected - this causes Azure Machine Learning to automatically preprocess the features before training.

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

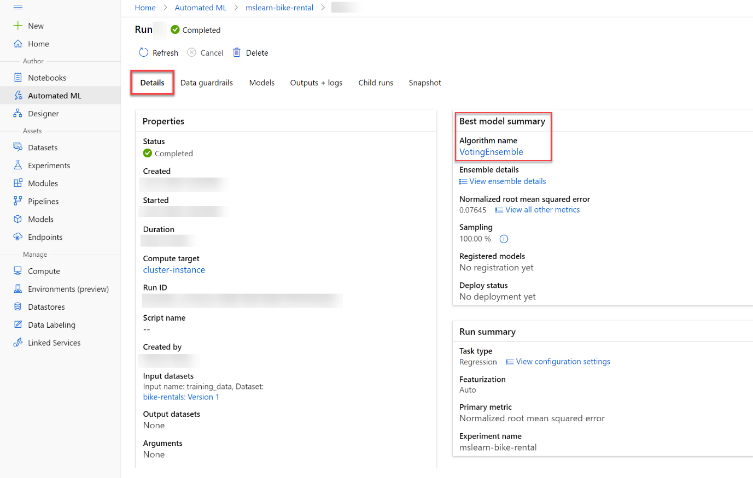
* + **[Optional] Select the validation and test type**
    - **Validation type**: Auto
    - **Test dataset (preview)**: No test dataset required

1. When you finish submitting the automated ML run details, it will start automatically. Wait for the run status to change from Preparing to Running.
2. 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 will automatically refresh periodically, but you can also select **↻ Refresh**. It may take ten minutes or so before models start to appear, as the cluster nodes need to be initialized before training can begin.
3. Wait for the experiment to finish.

## Review the best model

After the experiment has finished; you can review the best performing model that was generated (note that in this case, we used exit criteria to stop the experiment - so the "best" model found by the experiment may not be the best possible model,).

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

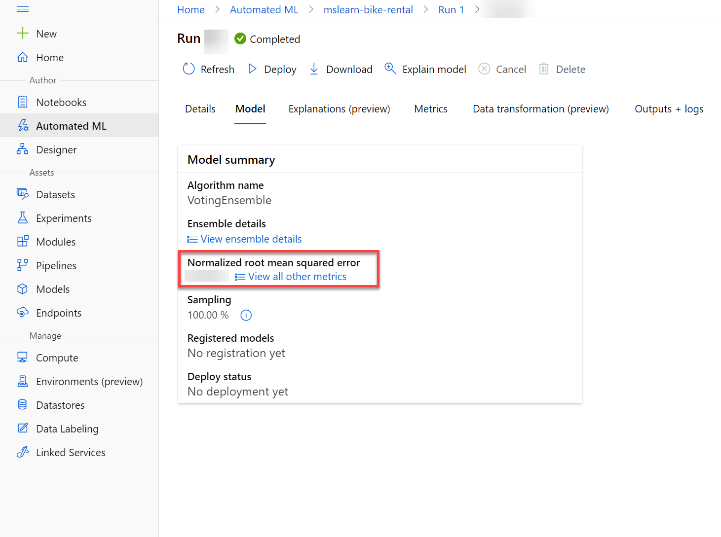


1. Select the **Algorithm name** for the best model to view its details.

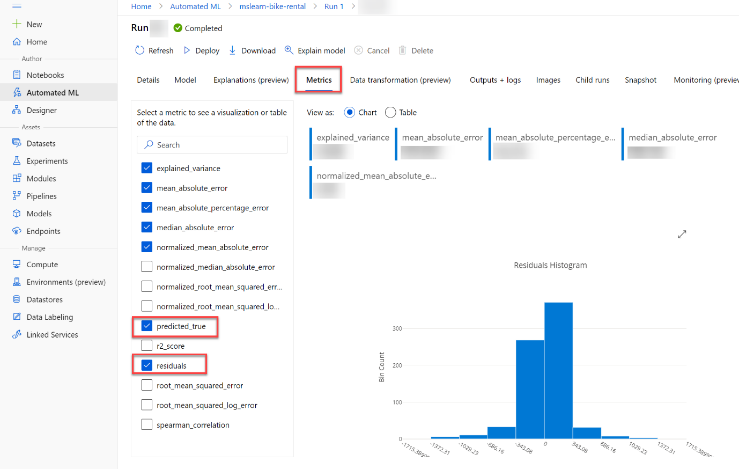
The best model is identified based on the evaluation metric you specified (Normalized root mean squared error). To calculate this metric, the training process used some of the data to train the model, and applied a technique called cross-validation to iteratively test the trained model with data it wasn't trained with and compare the predicted value with the actual known value.

The difference between the predicted and actual value (known as the residuals) indicates the amount of error in the model, and this particular performance metric 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 accurately the model is predicting.

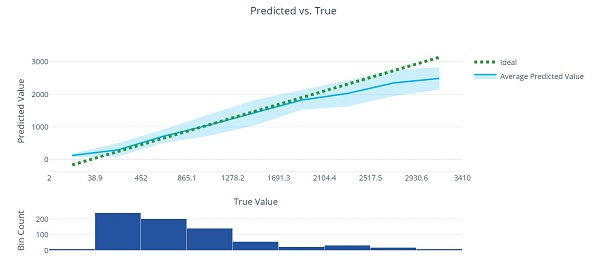
1. Next to the Normalized root mean squared error value, select **View all other metrics** to see values of other possible evaluation metrics for a regression model.



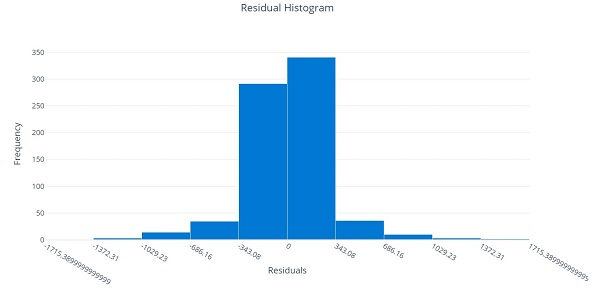
1. Select the **Metrics** tab and select the **residuals** and **predicted\_true** charts.



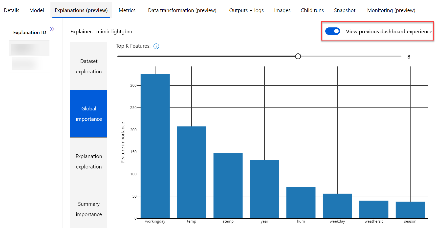
Then review the charts, which show the performance of the model by comparing the predicted values against the true values, and by showing the residuals (differences between predicted and actual values) as a histogram. The **Predicted vs. True** chart should show a diagonal trend in which the predicted value correlates closely to the true value. A dotted line shows how a perfect model should perform, and the closer the line for your model's average predicted value is to this, the better its performance. A histogram the line chart shows the distribution of 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; so what you should hope to see is that the most frequently occurring residual values are clustered around 0 (in other words, most of the errors are small), with fewer errors at the extreme ends of the scale.



1. Select the **Explanations** tab. Click on the arrows >> next to **Explanation ID** to expand the explanations list. Select an explanation ID, select **View previous dashboard experience** on the right-hand side. Then select **Global Importance**. This chart shows how much each feature in the dataset influences the label prediction, like this:



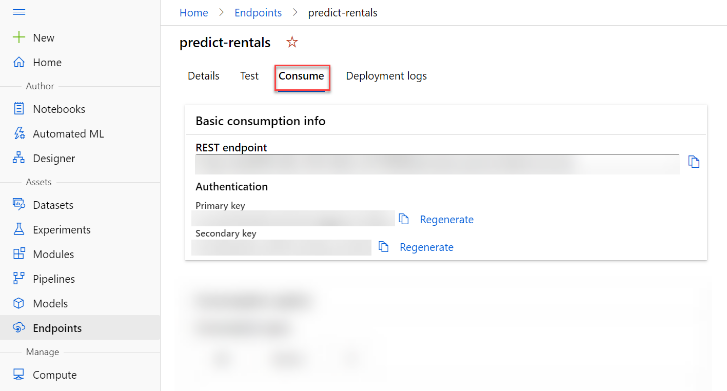
Deploy a model as a service

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](https://ml.azure.com/), 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**. You may need to select **↻ Refresh** periodically.
5. In Azure Machine Learning studio, view the **Endpoints** page and select the **predict-rentals** real-time endpoint. Then select the **Consume** tab and note the following information there. You need this information to connect to your deployed service from a client application.
   * The REST endpoint for your service
   * the Primary Key for your service



Note that you can use the ⧉ link next to these values to copy them to the clipboard.

## Test the deployed service

Now that you've deployed a service, you can test it using some simple code.

1. With the **Consume** page for the **predict-rentals** service page open in your browser, open a new browser tab and open a second instance of [Azure Machine Learning studio](https://ml.azure.com/). Then in the new tab, view the **Notebooks** page (under **Author**).
2. In the **Notebooks** page, under **My files**, use the **🗋** button to create a new file with the following settings:
   * **File location**: Users/your user name
   * **File name**: Test-Bikes.ipynb
   * **File type**: Notebook
   * **Overwrite if already exists**: Selected
3. When the new notebook has been created, ensure that the compute instance you created previously is selected in the **Compute** box, and that it has a status of **Running**.
4. Use the **≪** button to collapse the file explorer pane and give you more room to focus on the **Test-Bikes.ipynb** notebook tab.
5. In the rectangular cell that has been created in the notebook, paste the following code:

endpoint = 'YOUR\_ENDPOINT' #Replace with your endpoint

key = 'YOUR\_KEY' #Replace with your key

import json

import requests

#An array of features based on five-day weather forecast

x = [[1,1,2022,1,0,6,0,2,0.344167,0.363625,0.805833,0.160446],

[2,1,2022,1,0,0,0,2,0.363478,0.353739,0.696087,0.248539],

[3,1,2022,1,0,1,1,1,0.196364,0.189405,0.437273,0.248309],

[4,1,2022,1,0,2,1,1,0.2,0.212122,0.590435,0.160296],

[5,1,2022,1,0,3,1,1,0.226957,0.22927,0.436957,0.1869]]

#Convert the array to JSON format

input\_json = json.dumps({"data": x})

#Set the content type and authentication for the request

headers = {"Content-Type":"application/json",

"Authorization":"Bearer " + key}

#Send the request

response = requests.post(endpoint, input\_json, headers=headers)

#If we got a valid response, display the predictions

if response.status\_code == 200:

y = json.loads(response.json())

print("Predictions:")

for i in range(len(x)):

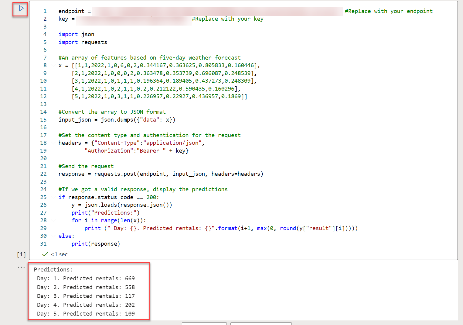
print (" Day: {}. Predicted rentals: {}".format(i+1, max(0, round(y["result"][i]))))

else:

print(response)

Don't worry too much about the details of the code. It just defines features for a five day period using hypothetical weather forecast data, and uses the **predict-rentals** service you created to predict cycle rentals for those five days.

1. Switch to the browser tab containing the **Consume** page for the **predict-rentals** service, and copy the REST endpoint for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_ENDPOINT.
2. Switch to the browser tab containing the **Consume** page for the **predict-rentals** service, and copy the Primary Key for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_KEY.
3. Save the notebook, Then use the **▷** button next to the cell to run the code. You will get predictions for the number of bicycle rentals expected per day.



1. Verify that predicted number of rentals for each day in the five day period are returned.

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.

Create a Regression Model with Azure Machine Learning designer

*Regression* is a form of machine learning that is used to predict a numeric *label* based on an item's *features*. Ex. automobile sales company might use the characteristics of a car 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.

Create an Azure Machine Learning workspace

Azure ML is a cloud-based platform for building and operating ML 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 ML, 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 ML workloads.

Same as in previous unit.

Create compute resources

To train and deploy models using Azure Machine Learning designer, you need compute on which to run the training process, and to test the trained model after deploying it.

Same as in previous unit.

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) and known label values (the numeric value that you want to train a model to predict).

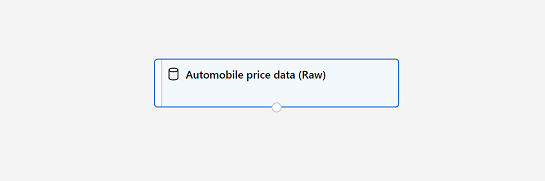
## Create a pipeline

To use the Azure Machine Learning designer, you create a pipeline that you will 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](https://ml.azure.com/), view the **Designer** page (under **Author**), and select **+** to create a new pipeline.
2. At the top left-hand side of the screen, click on the default pipeline name (**Pipeline-Created-on-date**) and change it to **Auto Price Training**.
3. You need to specify a compute target on which to run the pipeline. In the **Settings** pane, click on **Select compute target** to select the compute cluster you created previously (if the **Settings** pane is not visible, select the **⚙** icon next to the pipeline name at the top).

## Add and explore a dataset

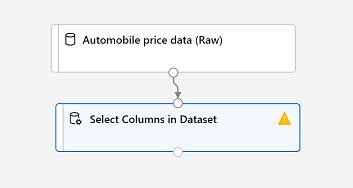
In this module, you'll train a regression model that predicts the price of an automobile based on its characteristics. Azure Machine Learning includes a sample dataset that you can use for this model.

1. Next to the pipeline name on the left, select the button **>>** to expand the panel. Find the **Sample datasets** section, and drag the **Automobile price data (Raw)** dataset from the **Samples** section onto the canvas.
2. Right-click (Ctrl+click on a Mac) the **Automobile price data (Raw)** dataset on the canvas, and on the **Outputs** menu, select **Dataset output** by clicking on the Preview data graph icon.
3. Review the schema of the data, noting that you can see the distributions of the various columns as histograms.
4. Scroll to the right of the dataset until you see the **Price** column. This is the label your model will predict.
5. Select the column header for the **price** column and view the details that are displayed in the pane to the right. These include various statistics for the column values, and a histogram showing the distribution of the column values.
6. Scroll back to the left and select the **normalized-losses** column header. Then review the statistics for this column noting, there are quite a few missing values in this column. This will limit its usefulness in predicting the **price** label; so you might want to exclude it from training.
7. View the statistics for the **bore**, **stroke**, and **horsepower** columns, noting the number of missing values. These columns have significantly fewer missing values than **normalized-losses**, so they may still be useful in predicting **price** if you exclude the rows where the values are missing from training.
8. Compare the values in the **stroke**, **peak-rpm**, and **city-mpg** columns. These are all measured in different scales, and its 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.
9. Close the **Automobile price data (Raw) result visualization** window so that you can see the dataset on the canvas.

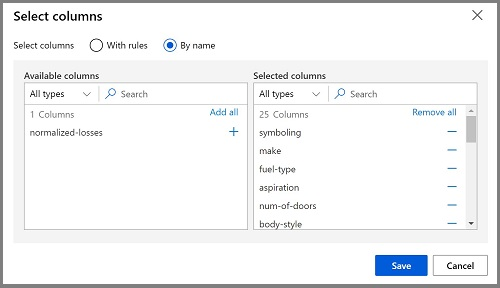
## Add data transformations

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

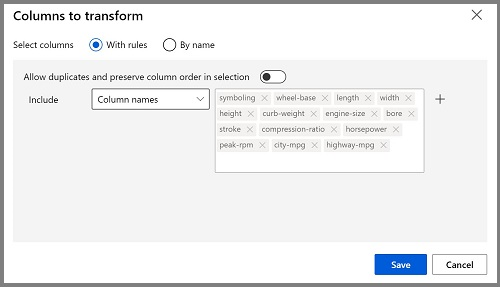
1. In the pane on the left, expand the **Data Transformation** section, which contains a wide range of modules you can use to transform data before model training.
2. Drag a **Select Columns in Dataset** module to the canvas, below the **Automobile price data (Raw)** module. Then connect the output at the bottom of the **Automobile price data (Raw)** module to the input at the top of the **Select Columns in Dataset** module, like this:



1. Select the **Select Columns in Dataset** module, and in its **Settings** pane on the right, select **Edit column**. Then in the **Select columns** window, select **By name** and use the **+** links to add all columns other than **normalized-losses**, like this:



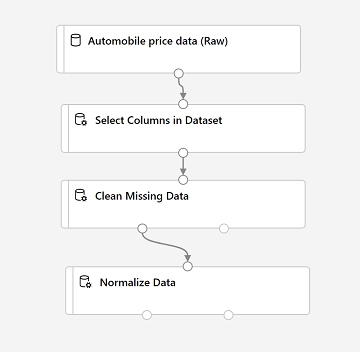
1. Drag a **Clean Missing Data** module from the **Data Transformations** section, and place it under the **Select Columns in Dataset** module. Then connect the output from the **Select Columns in Dataset** module to the input of the **Clean Missing Data** module.
2. Select the **Clean Missing Data** module, and in the settings pane on the right, click **Edit column**. Then in the **Select columns** window, select **With rules**, in the **Include** list select **Column names**, in the box of column names enter **bore**, **stroke**, and **horsepower** (making sure you match the spelling and capitalization exactly).
3. 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
4. Drag a **Normalize Data** module 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.
5. Select the **Normalize Data** module and view its settings, noting that it requires you to specify the transformation method and the columns to be transformed. Then, set the transformation to **MinMax** and edit the columns by applying a rule to include the following **Column names** (ensuring you match the spelling, capitalization, and hyphenation exactly): **symbolling, wheel-base, Length, width, Height, curb-weight, engine-size, bore, stroke, compression-ratio, horsepower, peak-rpm, city-mpg, highway-mpg**

****

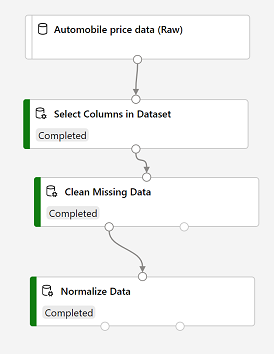
## Run the pipeline

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

1. Ensure your pipeline looks similar to this:



1. Select **Submit**, and run the pipeline as a new experiment named **mslearn-auto-training** on your compute cluster.
2. Wait for the run to finish. This may take 5 minutes or more. When the run has completed, the modules should look like this:



## View the transformed data

The dataset is now prepared for model training.

1. Select the completed **Normalize Data** module, and in its **Settings** pane on the right, on the **Outputs + logs** tab, select the **Visualize** icon for the **Transformed dataset**.
2. View the data, noting that the **normalized-losses** column has been removed, all rows contain data for **bore**, **stroke**, and **horsepower**, and the numeric columns you selected have been normalized to a common scale.
3. Close the normalized data result visualization.

Create and run a training pipeline

After you've used data transformations to prepare the data, you can use it to train a ML 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 for reference as you add and configure the required modules.

1. Open the **Auto Price Training** pipeline you created in the previous unit.
2. In the pane on the left, in the **Data Transformations** section, drag 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.
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
   * **Random seed**: 123
   * **Stratified split**: False
4. Expand the **Model Training** section in the pane on the left, and drag 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. Expand the **Machine Learning Algorithms** section, and under **Regression**, drag 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. 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**](https://aka.ms/mlcheatsheet).
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. Expand the **Model Scoring & Evaluation** section and drag 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:



## 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.
3. When the experiment run has completed, select the **Score Model** module and in the settings pane, on the **Outputs + logs** tab, under **Data outputs** in the **Scored dataset** section, use the **Preview Data** icon 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.

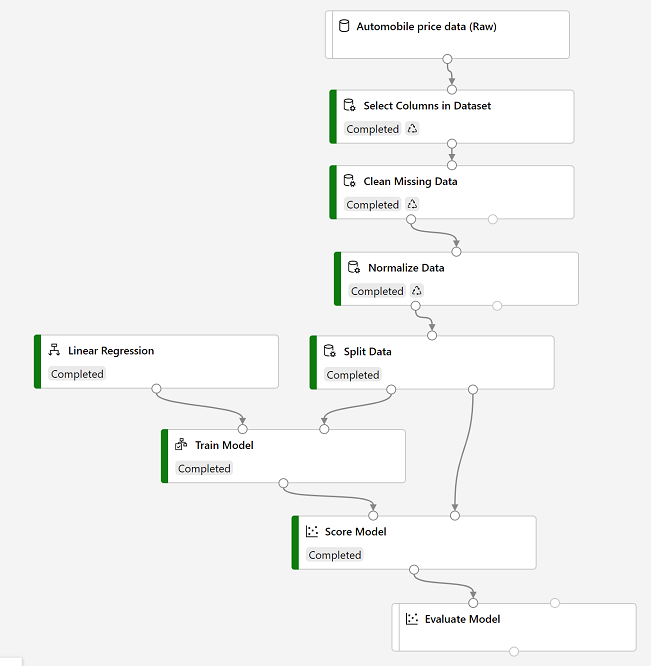
The model is predicting values for the **price** label, but how reliable are its predictions? To assess that, you need to evaluate the model.

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 pane on the left, in the **Model Scoring & Evaluation** section, drag 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 **Evaluate Model** module.
3. Ensure your pipeline looks like this:



1. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-auto-training**.
2. Wait for the experiment run to complete. When the experiment run has completed, select the **Evaluate Model** module and in the settings pane, on the **Outputs + logs** tab, under **Data outputs** in the **Evaluation results** section, use the **Preview Data** icon to view the results. These include the following regression performance 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 (R2)**: 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.
3. Close the **Evaluate Model result visualization** window.

You can try a different regression algorithm and compare the results by connecting the same outputs from the **Split Data** module to a second **Train model** module (with a different algorithm) and a second **Score Model** module; and then connecting the outputs of both **Score Model** modules to the same **Evaluate Model** module for a side-by-side comparison.

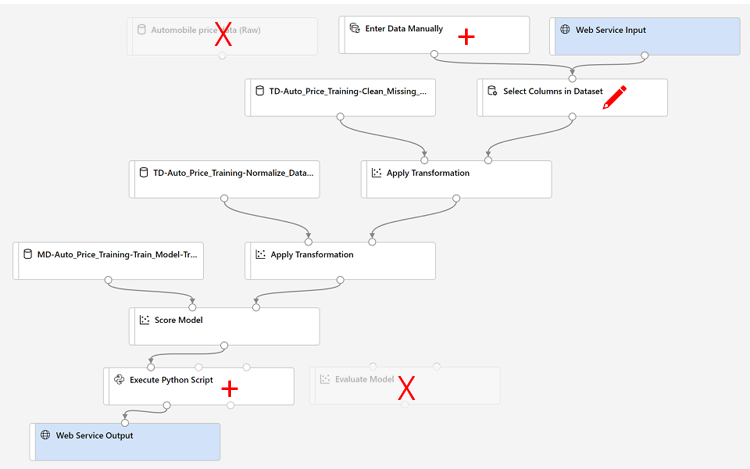
Create an inference pipeline

After creating and running a pipeline to train the model, you need a second pipeline that performs the same data transformations for new data, and then uses the trained model to infer (in other words, predict) label values based on its features. This 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, click the **Designer** page to view all of the pipelines you have created. Then open the **Auto Price Training** pipeline you created previously.
2. Navigate to the the **Create inference pipeline** drop-down list, located on the top right hand corner of the screen. If you do not see it, you may need to expand your screen or click on the **...** three dots which represent **More Actions** on the top right hand corner. Then 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 does not include ***Web Service Input*** and ***Web Service Output*** modules, go back to the ***Designer*** page and then re-open the ***Auto Price Training-real time inference*** pipeline.
3. 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 have been encapsulated in this pipeline so that the statistics from your training data will be used to normalize any new data values, and the trained model will be used to score the new data.

You are going to make the following changes to the inference pipeline in the next steps:



1. 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 has not 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):

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

1. Connect the new **Enter Data Manually** module to the same **dataset** input of the **Select Columns in Dataset** module as the **Web Service Input**.
2. 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.
3. The inference pipeline includes the **Evaluate Model** module, which is not useful when predicting from new data, so delete this module.
4. The output from the **Score Model** module includes all of the input features as well as 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 the default python script with the following code (which selects only the **Scored Labels** column and renames it to **predicted\_price**):

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**.

1. Submit the pipeline as a new experiment named **mslearn-auto-inference** on your compute cluster. This may take a while!
2. When the pipeline has completed, select the **Execute Python Script** module, and in the settings pane, on the **Output + logs** tab, visualize the **Result dataset** to see the predicted prices for the three cars in the input data.
3. 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.

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. At the top right, select **Deploy**, and deploy a new real-time endpoint, using the following settings:
   * **Name**: predict-auto-price
   * **Description**: Auto price regression.
   * **Compute type**: Azure Container Instance
3. 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

Now you can test your deployed service from a client application - in this case, you'll use the code in the cell below to simulate a client application.

1. On the **Endpoints** page, open the **predict-auto-price** real-time endpoint.
2. When the **predict-auto-price** endpoint opens, view the **Consume** tab and note the following information there. You need this to connect to your deployed service from a client application.
   * The REST endpoint for your service
   * The Primary Key for your service
3. Observe that you can use the ⧉ link next to these values to copy them to the clipboard.
4. With the **Consume** page for the **predict-auto-price** service page open in your browser, open a new browser tab and open a second instance of [Azure Machine Learning studio](https://ml.azure.com/). Then in the new tab, view the **Notebooks** page (under **Author**).
5. In the **Notebooks** page, under **My files**, use the **🗋** button to create a new file with the following settings:
   * **File location**: Users/your user name
   * **File name**: Test-Autos
   * **File type**: Notebook
   * **Overwrite if already exists**: Selected
6. When the new notebook has been created, ensure that the compute instance you created previously is selected in the **Compute** box, and that it has a status of **Running**.
7. Use the **≪** button to collapse the file explorer pane and give you more room to focus on the **Test-Autos.ipynb** notebook tab.
8. In the rectangular cell that has been created in the notebook, paste the following code:

endpoint = 'YOUR\_ENDPOINT' #Replace with your endpoint

key = 'YOUR\_KEY' #Replace with your key

import urllib.request

import json

import os

# Prepare the input data

data = {

"Inputs": {

"WebServiceInput0":

[

{

'symboling': 3,

'normalized-losses': None,

'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"

},

],

},

"GlobalParameters": {

}

}

body = str.encode(json.dumps(data))

headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ key)}

req = urllib.request.Request(endpoint, body, headers)

try:

response = urllib.request.urlopen(req)

result = response.read()

json\_result = json.loads(result)

y = json\_result["Results"]["WebServiceOutput0"][0]

print(y)

except urllib.error.HTTPError as error:

print("The request failed with status code: " + str(error.code))

# Print the headers to help debug the error

print(error.info())

print(json.loads(error.read().decode("utf8", 'ignore')))

Don't worry too much about the details of the code. It just submits details of a car and uses the **predict-auto-price** service you created to get a predicted price.

1. Switch to the browser tab containing the **Consume** page for the **predict-auto-price** service, and copy the REST endpoint for your service. Then switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_ENDPOINT.
2. Switch to the browser tab containing the **Consume** page for the **predict-auto-price** service, and copy the Primary Key for your service. Then switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_KEY.
3. Save the notebook. Then use the **▷** button next to the cell to run the code.
4. Verify that predicted price is returned.

Create a classification model with Azure Machine Learning designer

*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.

Create an Azure Machine Learning workspace

Same as in the previous unit.

Create compute resources

Same as in the previous unit.

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).

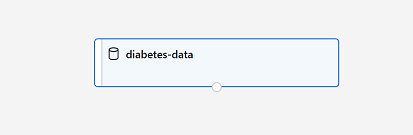
## Create a dataset

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](https://ml.azure.com/), view the **Datasets** page. Datasets represent specific data files or tables that you plan to work with in Azure ML.
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
   * **Schema**:
     + Include all columns other than **Path**
     + Review the automatically detected types
   * **Confirm details**:
     + Do not profile the dataset after creation
3. 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

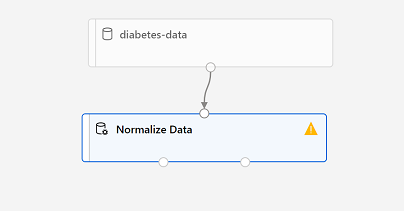
To get started with Azure Machine Learning designer, first you must create a pipeline and add the dataset you want to work with.

1. In [Azure Machine Learning studio](https://ml.azure.com/) for your workspace, view the **Designer** page and select **+** to create a new pipeline.
2. At the top left-hand side of the screen, click on the default pipeline name (**Pipeline-Created-on-date**) and change it to **Diabetes Training**.
3. You need to specify a compute target on which to run the pipeline. In the **Settings** pane, click on **Select compute target** to select the compute cluster you created previously (if the **Settings** pane is not visible, select the **⚙** icon next to the pipeline name at the top).
4. Next to the pipeline name on the left, select the button **>>** to expand the panel. Drag the **diabetes-data** dataset you created in the previous exercise onto the canvas.
5. Right-click (Ctrl+click on a Mac) the **diabetes-data** dataset on the canvas, and on the **Outputs** menu, select **Dataset output** by clicking on the Preview data graph icon.
6. Review the schema of the data, noting that you can see the distributions of the various columns as histograms.
7. Scroll to the right 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.
8. Scroll back to the left 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.
9. Close the **diabetes-data result visualization** window so that you can see the dataset on the canvas like this:

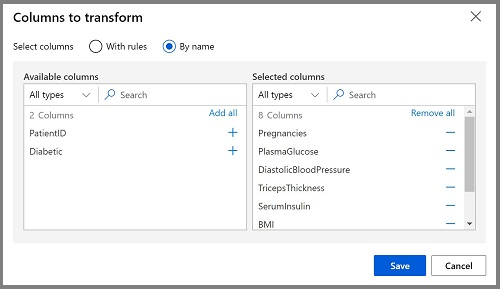
## Add Transformations

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

1. In the pane on the left, expand the **Data Transformation** section, which contains a wide range of modules you can use to transform data before model training.
2. Drag a **Normalize Data** module to 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:



1. Select the **Normalize Data** module and view its settings, noting that it requires you to specify the transformation method and the columns to be transformed.
   * 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**

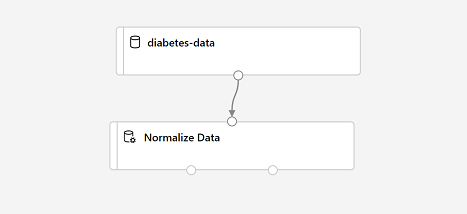
****

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. Ensure your pipeline looks similar to this:



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.

## View the transformed data

The dataset is now prepared for model training.

1. Select the completed **Normalize Data** module, and in its **Settings** pane on the right, on the **Outputs + logs** tab, select the **Visualize** icon for the **Transformed dataset**.
2. View the data, noting that the numeric columns you selected have been normalized to a common scale.
3. Close the normalized data result visualization.

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

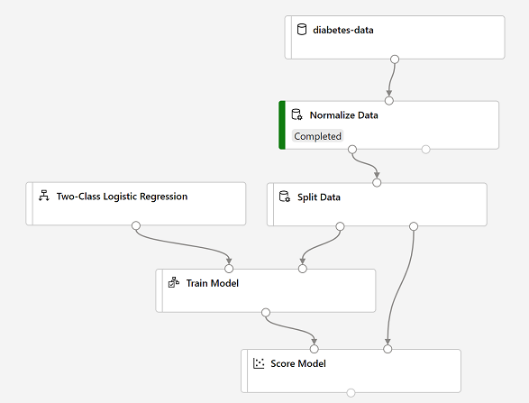
It's common practice to train the model using a subset of the data, while holding back some data with which to test the trained model. This enables you to compare the labels that the model predicts with the actual known labels in the original dataset.

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 pane on the left, in the **Data Transformations** section, drag 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.
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
   * **Random seed**: 123
   * **Stratified split**: False
4. Expand the **Model Training** section in the pane on the left, and drag 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** (matching the case and spelling exactly!)
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. Expand the **Machine Learning Algorithms** section, and under **Classification**, drag 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.

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**](https://aka.ms/mlcheatsheet).

1. 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. Expand the **Model Scoring & Evaluation** section and drag 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.
2. Ensure your pipeline looks like this:



## 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 the **Score Model** module and in the settings pane, on the **Outputs + Logs** tab, under **Data outputs** in the **Scored dataset** section, use the **Visualize** icon to view the results.
4. 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).
5. Close the **Score Model result visualization** window.

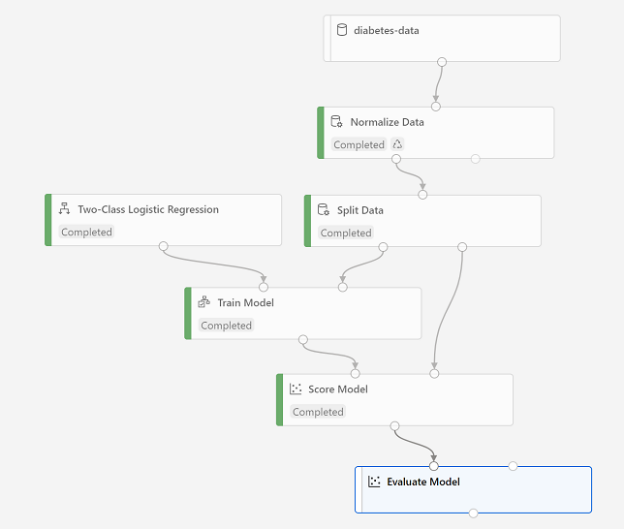
The model is predicting values for the **Diabetic** label, but how reliable are its predictions? To assess that, you need to evaluate the model.

Evaluate a classification model

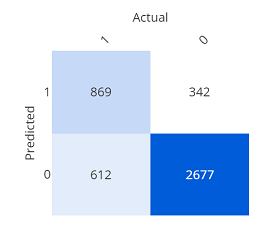
The validation data you held back and used to score the model includes the known values for the label. So to validate the model, you can compare the true values for the label to the label values that were predicted when you scored the validation dataset. Based on this comparison, you can calculate various 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 pane on the left, in the **Model Scoring & Evaluation** section, drag 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.



1. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-diabetes-training**.
2. Wait for the experiment run to finish.
3. When the experiment run has finished, select the **Evaluate Model** module and in the settings pane, on the **Outputs + Logs** tab, under **Data outputs** in the **Evaluation results** section, use the **Preview Data** icon to view the performance metrics. These metrics can help data scientists assess how well the model predicts based on the validation data.
4. 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**.

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.

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.

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.

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.

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 that performs the same data transformations for new data, and then uses the trained model to infer (in other words, predict) label values based on its features. This pipeline will form the basis for a predictive service that you can publish for applications to use.

1. In Azure Machine Learning Studio, click the **Designer** page to view all of the pipelines you have created. Then open the **Diabetes Training** pipeline you created previously.
2. Navigate to the the **Create inference pipeline** drop-down list, located on the top right hand corner of the screen. If you do not see it, you may need to expand your screen or click on the **...** three dots which represent **More Actions** on the top right hand corner. Then 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 does not include ***Web Service Input*** and ***Web Service Output*** modules, go back to the ***Designer*** page and then re-open the ***Diabetes Training-real time inference*** pipeline.
3. 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 have been encapsulated in this pipeline so that the statistics from your training data will be used to normalize any new data values, and the trained model will be used to score the new data.

* Replace the **diabetes-data** dataset with an **Enter Data Manually** module that does not 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.

1. 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 unintuitive to include in new patient data for which a diabetes prediction has not 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 new patient observations:

PatientID,Pregnancies,PlasmaGlucose,DiastolicBloodPressure,TricepsThickness,SerumInsulin,BMI,DiabetesPedigree,Age

1882185,9,104,51,7,24,27.36983156,1.350472047,43

1662484,6,73,61,35,24,18.74367404,1.074147566,75

1228510,4,115,50,29,243,34.69215364,0.741159926,59

1. Connect the new **Enter Data Manually** module to the same **Dataset** input of the **Apply Transformation** module as the **Web Service Input**.
2. The inference pipeline includes the **Evaluate Model** module, which is not useful when predicting from new data, so delete this module.
3. The output from the **Score Model** module includes all of the input features as well as the predicted label and probability score. To limit the output to only the prediction and probability:
4. Delete the connection between the **Score Model** module and the **Web Service Output**.
5. 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 **PatientID**, **Scored Labels** and **Scored Probabilities** columns and renames them appropriately):

import pandas as pd

def azureml\_main(dataframe1 = None, dataframe2 = None):

scored\_results = dataframe1[['PatientID', 'Scored Labels', 'Scored Probabilities']]

scored\_results.rename(columns={'Scored Labels':'DiabetesPrediction',

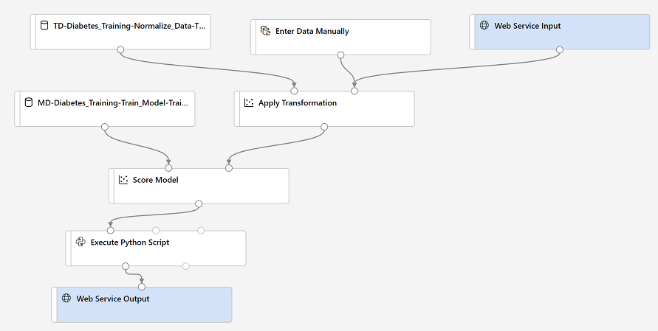
'Scored Probabilities':'Probability'},

inplace=True)

return scored\_results

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

1. Verify that your pipeline looks similar to the following:



1. Run the pipeline as a new experiment named **mslearn-diabetes-inference** on your compute cluster. This may take a while!
2. When the pipeline has finished, select the **Execute Python Script** module, and in the settings pane, on the **Output + Logs** tab, visualize the **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.

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. 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

1. 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

Now you can test your deployed service from a client application - in this case, you'll use the code in the cell below to simulate a client application.

1. On the **Endpoints** page, open the **predict-diabetes** real-time endpoint.
2. When the **predict-diabetes** endpoint opens, view the **Consume** tab and note the following information there. You need this to connect to your deployed service from a client application.
   * The REST endpoint for your service
   * the Primary Key for your service
3. Note that you can use the ⧉ link next to these values to copy them to the clipboard.
4. With the **Consume** page for the **predict-diabetes** service page open in your browser, open a new browser tab and open a second instance of [Azure Machine Learning studio](https://ml.azure.com/). Then in the new tab, view the **Notebooks** page (under **Author**).
5. In the **Notebooks** page, under **My files**, use the **🗋** button to create a new file with the following settings:
   * **File location**: Users/your user name
   * **File name**: Test-Diabetes
   * **File type**: Notebook
   * **Overwrite if already exists**: Selected
6. When the new notebook has been created, ensure that the compute instance you created previously is selected in the **Compute** box, and that it has a status of **Running**.
7. Use the **≪** button to collapse the file explorer pane and give you more room to focus on the **Test-Diabetes.ipynb** notebook tab.
8. In the rectangular cell that has been created in the notebook, paste the following code:

endpoint = 'YOUR\_ENDPOINT' #Replace with your endpoint

key = 'YOUR\_KEY' #Replace with your key

import urllib.request

import json

import os

data = {

"Inputs": {

"WebServiceInput0": [{

'PatientID': 1882185,

'Pregnancies': 9,

'PlasmaGlucose': 104,

'DiastolicBloodPressure': 51,

'TricepsThickness': 7,

'SerumInsulin': 24,

'BMI': 27.36983156,

'DiabetesPedigree': 1.3504720469999998,

'Age': 43,

},], },

"GlobalParameters": {

}

}

body = str.encode(json.dumps(data))

headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ key)}

req = urllib.request.Request(endpoint, body, headers)

try:

response = urllib.request.urlopen(req)

result = response.read()

json\_result = json.loads(result)

output = json\_result["Results"]["WebServiceOutput0"][0]

print('Patient: {}\nPrediction: {}\nProbability: {:.2f}'.format(output["PatientID"],

output["DiabetesPrediction"], output["Probability"]))

except urllib.error.HTTPError as error:

print("The request failed with status code: " + str(error.code))

# Print the headers to help debug

print(error.info())

print(json.loads(error.read().decode("utf8", 'ignore')))

1. Switch to the browser tab containing the **Consume** page for the **predict-diabetes** service, and copy the REST endpoint for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_ENDPOINT.
2. Switch to the browser tab containing the **Consume** page for the **predict-diabetes** service, and copy the Primary Key for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_KEY.
3. Save the notebook. Then use the **▷** button next to the cell to run the code.
4. Verify that predicted diabetes diagnosis is returned.

Create a Clustering Model with Azure Machine Learning designer

*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.

Create an Azure Machine Learning workspace

Same as in the previous unit.

Create compute resources

Same as in the previous unit.

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

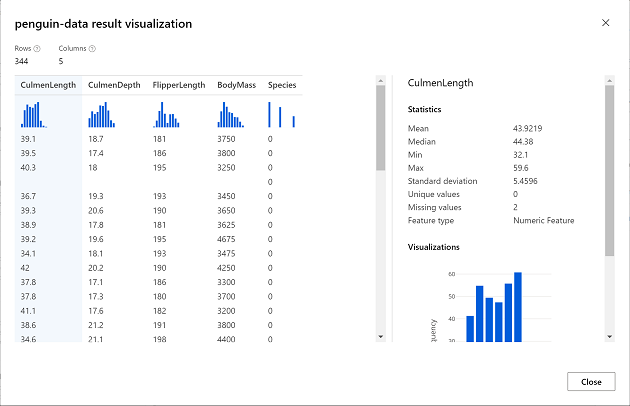
In Azure ML, data for model training and other operations is usually encapsulated in an object called a dataset. In this module, you'll use a dataset that includes observations of three species of penguin.

1. In [Azure Machine Learning studio](https://ml.azure.com/), view the **Datasets** page. Datasets represent specific data files or tables that you plan to work with in Azure ML.
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
     + **Settings and preview**:
     + **File format**: Delimited
     + **Delimiter**: Comma
     + **Encoding**: UTF-8
     + **Column headers**: Only first file has headers
     + **Skip rows**: None
   * **Schema**:
     + Include all columns other than **Path**
     + Review the automatically detected types
   * **Confirm details**:
     + Do not profile the dataset after creation
3. 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.

## 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](https://ml.azure.com/) for your workspace, view the **Designer** page and create a new pipeline.
2. At the top left-hand side of the screen, click on the default pipeline name (**Pipeline-Created-on-date**) to **Train Penguin Clustering**.
3. You need to specify a compute target on which to run the pipeline. In the **Settings** pane, click on **Select compute target** to select the compute cluster you created previously (if the **Settings** pane is not visible, select the **⚙** icon next to the pipeline name at the top).
4. Next to the pipeline name on the left, select the button **>>** to expand the panel. Drag the **penguin-data** dataset you created in the previous exercise onto the canvas.
5. Right-click (Ctrl+click on a Mac) the **penguin-data** dataset on the canvas, and on the **Outputs** menu, select **Dataset output** by clicking on the Preview data graph icon.
6. Review the 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:

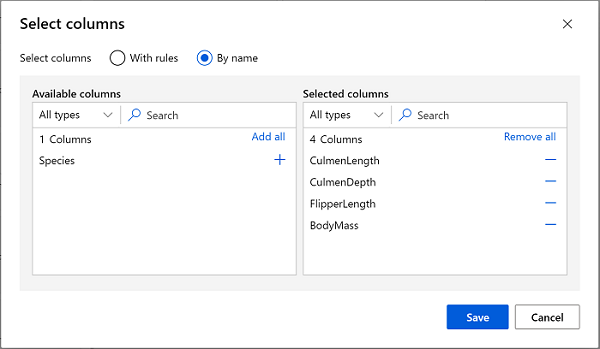


1. Note the following characteristics of the dataset:
   * The dataset includes the following columns:
     + **CulmenLength**: Length of the penguin's bill in millimeters.
     + **CulmenDepth**: Depth of the penguin's bill in millimeters.
     + **FlipperLength**: Length of the penguin's flipper in 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).
2. Close the dataset visualization so you can see the dataset on the pipeline canvas.

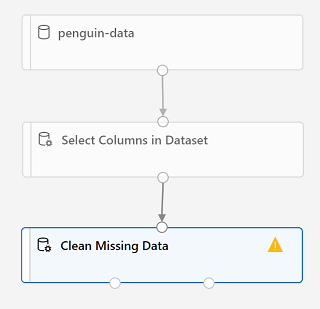
## Apply transformations

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.

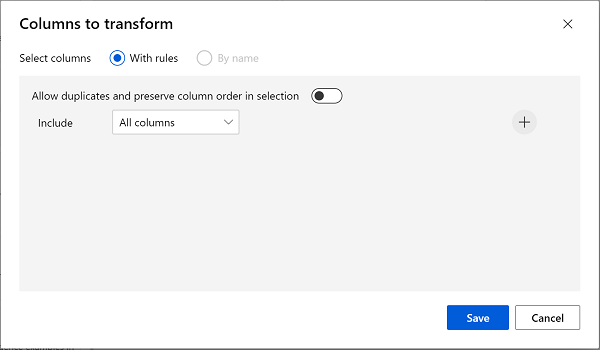
1. In the pane on the left, expand the **Data Transformation** section, which contains a wide range of modules you can use to transform data before model training.
2. To cluster the penguin observations, we're going to use only the measurements - we'll ignore the species column. So, drag a **Select Columns in Dataset** module to 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.
3. Select the **Select Columns in Dataset** module, and in its **Settings** 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:



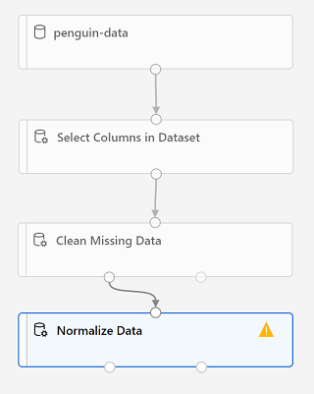
1. Save the **Select Columns in a Dataset** module settings to return to the designer canvas.
2. Drag a **Clean Missing Data** module to the canvas, below the **Select columns in a dataset** module and connect them like this:



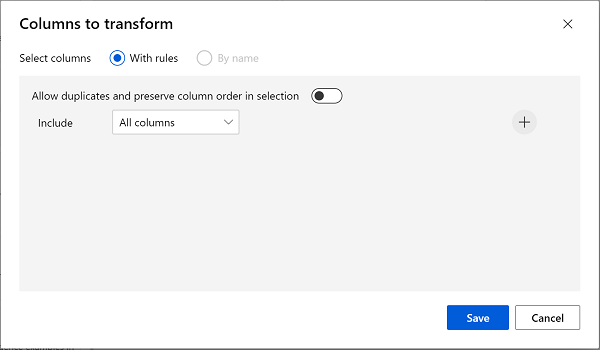
1. Select the **Clean Missing Data** module, and in the settings pane on the right, click **Edit column**. Then in the **Select columns** window, select **With rules** and include **All columns**; like this:



1. 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
2. Drag a **Normalize Data** module 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.



1. Select the **Normalize Data** module, and in its **Settings** 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:

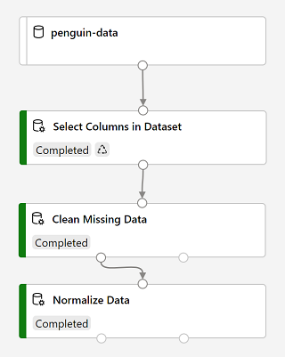


1. Save the **Normalize Data** module settings to return to the designer canvas.

## 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. When the run has completed, the modules should look like this:



## View the transformed data

The dataset is now prepared for model training.

1. Select the completed **Normalize Data** module, and in its **Settings** pane on the right, on the **Outputs + logs** tab, select the **Visualize** icon for the **Transformed dataset**.
2. 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.
3. Close the normalized data result visualization.

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.

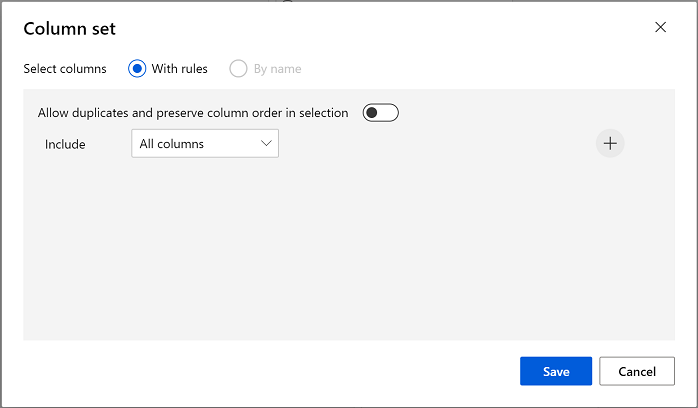
Create and run a training pipeline

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

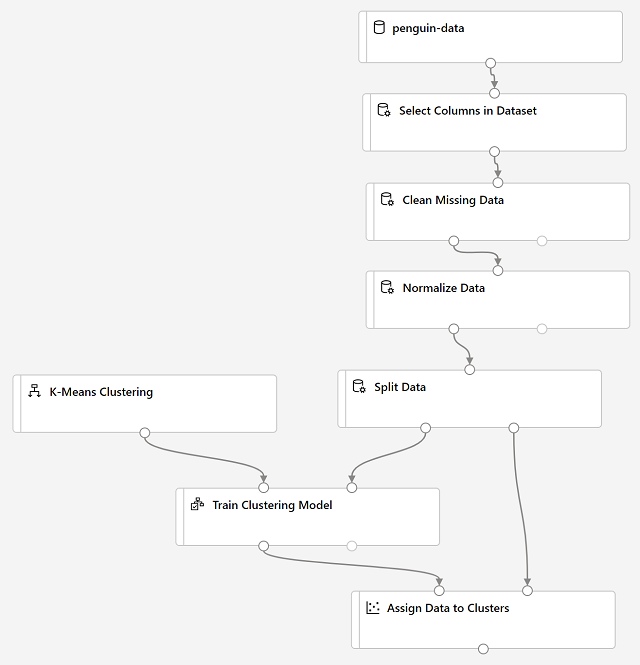
## Add training modules

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.

1. Open the **Train Penguin Clustering** pipeline, if it's not already open.
2. In the pane on the left, in the **Data Transformations** section, drag 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.
3. Select the **Split Data** module, and configure its settings as follows:
   1. **Splitting mode**: Split Rows
   2. **Fraction of rows in the first output dataset**: 0.7
   3. **Random seed**: 123
   4. **Stratified split**: False
4. Expand the **Model Training** section in the pane on the left, and drag 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. Select the **Train Clustering Model** module and in its settings pane, on the **Parameters** tab, select **Edit Columns** and use the **With rules** option to include all columns; like this:



1. 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. Expand the **Machine Learning Algorithms** section, and under **Clustering**, drag 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.
2. 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 its settings pane, on the **Parameters** tab, set the **Number of centroids** parameter to **3**. You can think of data observations, like the penguin measurements, as being multidimensional vectors. The K-Means algorithm works by:
   1. initializing K coordinates as randomly selected points called centroids in n-dimensional space (where n is the number of dimensions in the feature vectors).
   2. Plotting the feature vectors as points in the same space, and assigning each point to its closest centroid.
   3. Moving the centroids to the middle of the points allocated to it (based on the mean distance).
   4. Reassigning the points to their closest centroid after the move.
   5. Repeating steps 3 and 4 until the cluster allocations stabilize or the specified number of iterations has completed.
3. 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. Expand the **Model Scoring & Evaluation** section and drag 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

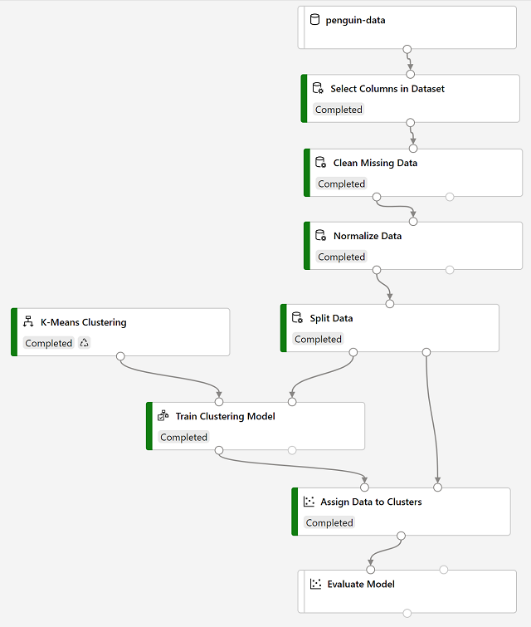
1. Select **Submit**, and run the pipeline using the existing experiment named **mslearn-penguin-training** on your compute cluster.
2. Wait for the experiment run to finish. This may take 5 minutes or more.
3. When the experiment run has finished, select the **Assign Data to Clusters** module and in its settings pane, on the **Outputs + Logs** tab, under **Data outputs** in the **Results dataset** section, use the **Visualize** icon to view the results.
4. Scroll to the right, 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.
5. Close the **Assign Data to Clusters** visualization.

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 pane on the left, in the **Model Scoring & Evaluation** section, drag an **Evaluate Model** module to the canvas, under the **Assign Data to Clusters** module, and 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:



1. Select **Submit**, and run the pipeline using the existing **mslearn-penguin-training** experiment.
2. Wait for the experiment run to finish. When the experiment run has finished, select the **Evaluate Model** module and in the settings pane, on the **Outputs + Logs** tab, under **Data outputs** in the **Evaluation results** section, use the **Preview Data** icon to view the performance metrics. 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:
   1. **Average Distance to Other Center**: This indicates how close, on average, each point in the cluster is to the centroids of all other clusters.
   2. **Average Distance to Cluster Center**: This indicates how close, on average, each point in the cluster is to the centroid of the cluster.
   3. **Number of Points**: The number of points assigned to the cluster.
   4. **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.
3. 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 that uses the model to assign new data observations to clusters. This will form the basis for a predictive service that you can publish for applications to use.

1. In Azure Machine Learning Studio, open the **Train Penguin Clustering** pipeline you created previously.
2. Navigate to the the **Create inference pipeline** drop-down list, located on the top right hand corner of the screen. If you do not see it, you may need to expand your screen or click on the **...** three dots which represent **More Actions** on the top right hand corner. Then 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 does not include ***Web Service Input*** and ***Web Service Output*** modules, go back to the ***Designer*** page and then re-open the ***Train Penguin Clustering-real time inference*** pipeline.
3. 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 encapsulated in this pipeline based on the statistics from your training data, and will be used to transform and score the new data.
   1. Replace the **penguin-data** dataset with an **Enter Data Manually** module that does not include the **Species** column.
   2. Remove the **Select Columns in Dataset** module, which is now redundant.
   3. Connect the **Web Service Input** and **Enter Data Manually** modules (which represent inputs for data to be clustered) to the first **Apply Transformation** module.
   4. Remove the **Evaluate Model** module.
4. 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 **Data Input and Output** section. 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):

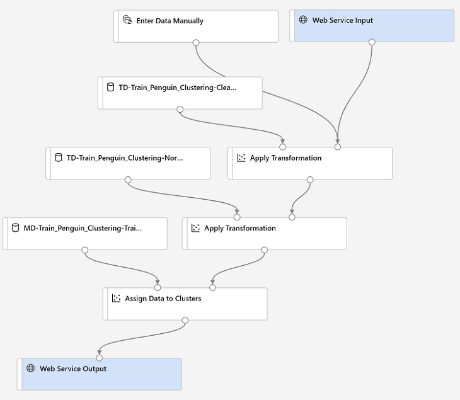
CulmenLength,CulmenDepth,FlipperLength,BodyMass

39.1,18.7,181,3750

49.1,14.8,220,5150

46.6,17.8,193,3800

1. 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.
2. Delete the **Evaluate Model** module.
3. Verify that your pipeline looks similar to the following:



1. Submit the pipeline as a new experiment named **mslearn-penguin-inference** on your compute cluster. This may take a while!
2. When the pipeline has finished, visualize the **Results dataset** output of the **Assign Data to Clusters** module 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. 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
3. 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

Now you can test your deployed service from a client application - in this case, you'll use the code in the cell below to simulate a client application.

1. On the **Endpoints** page, open the **predict-penguin-clusters** real-time endpoint.
2. When the **predict-penguin-clusters** endpoint opens, view the **Consume** tab and note the following information there. You need this to connect to your deployed service from a client application.
   * The REST endpoint for your service
   * the Primary Key for your service
3. Note that you can use the ⧉ link next to these values to copy them to the clipboard.
4. With the **Consume** page for the **predict-penguin-clusters** service page open in your browser, open a new browser tab and open a second instance of [Azure Machine Learning studio](https://ml.azure.com/). Then in the new tab, view the **Notebooks** page (under **Author**).
5. In the **Notebooks** page, under **My files**, use the **🗋** button to create a new file with the following settings:
   * **File location**: Users/your user name
   * **File name**: Test-Penguins
   * **File type**: Notebook
   * **Overwrite if already exists**: Selected
6. When the new notebook has been created, ensure that the compute instance you created previously is selected in the **Compute** box, and that it has a status of **Running**.
7. Use the **≪** button to collapse the file explorer pane and give you more room to focus on the **Test-Penguins.ipynb** notebook tab.
8. In the rectangular cell that has been created in the notebook, paste the following code:

endpoint = 'YOUR\_ENDPOINT' #Replace with your endpoint

key = 'YOUR\_KEY' #Replace with your key

import urllib.request

import json

import os

data = {

"Inputs": {

"WebServiceInput0":

[{

'CulmenLength': 49.1,

'CulmenDepth': 4.8,

'FlipperLength': 1220,

'BodyMass': 5150,

},},

"GlobalParameters": { }

}

body = str.encode(json.dumps(data))

headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ key)}

req = urllib.request.Request(endpoint, body, headers)

try:

response = urllib.request.urlopen(req)

result = response.read()

json\_result = json.loads(result)

output = json\_result["Results"]["WebServiceOutput0"][0]

print('Cluster: {}'.format(output["Assignments"]))

except urllib.error.HTTPError as error:

print("The request failed with status code: " + str(error.code))

Don't worry too much about the details of the code. It just defines features for a penguin, and uses the **predict-penguin-clusters** service you created to predict a cluster assignment.

1. Switch to the browser tab containing the **Consume** page for the **predict-penguin-clusters** service, and copy the REST endpoint for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_ENDPOINT.
2. Switch to the browser tab containing the **Consume** page for the **predict-penguin-clusters** service, and copy the Primary Key for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_KEY.
3. Save the notebook, Then use the **▷** button next to the cell to run the code.
4. Verify that predicted cluster is returned.

Explore computer vision

Analyse images with the Computer Vision service

*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 ML 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.

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.

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

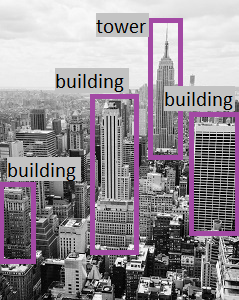
Computer Vision has the ability to analyze an image, evaluate the objects that are detected, and generate a human-readable phrase or sentence that can describe what was detected in the image. Depending on the image contents, the service may return multiple results, or phrases. Each returned phrase will 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.

### 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.

### 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](https://docs.microsoft.com/en-us/azure/cognitive-services/face/). 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.

### 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\_**.

### 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. Inf: [Read text with the Computer Vision service](https://docs.microsoft.com/en-us/learn/modules/read-text-computer-vision/)

### 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.

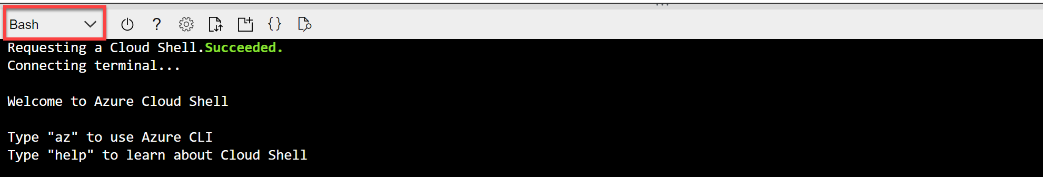
If you haven't already done so, create a **Cognitive Services** resource in your Azure subscription.

1. Open the Azure portal at [https://portal.azure.com](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.

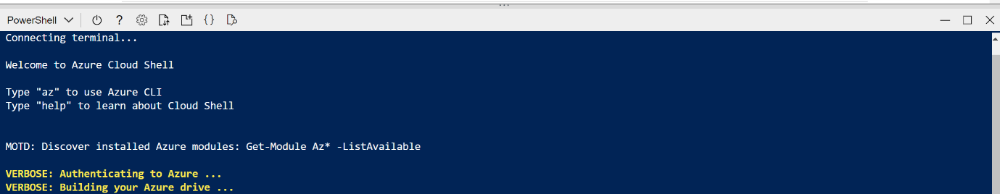
## Run Cloud Shell

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.



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



## Configure and run a client application

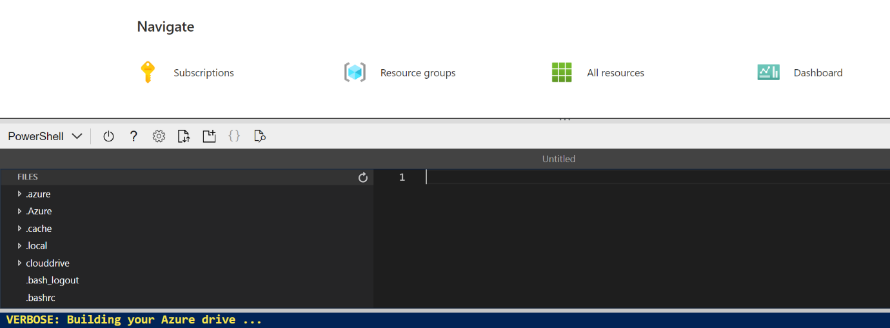
Now that you have a Cloud Shell environment, you can run a simple application that uses the Computer Vision service to analyze an image.

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

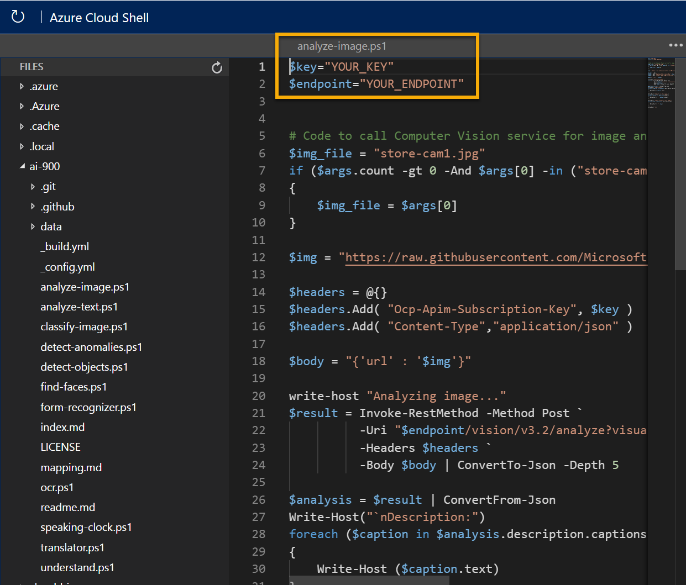
git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900

1. 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: code .

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



1. 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:



1. 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.

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:

$key="1a2b3c4d5e6f7g8h9i0j...."

$endpoint="https..."

1. 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:



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

cd ai-900

./analyze-image.ps1 store-camera-1.jpg

1. 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.

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.

## 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.

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 "IterationX", 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.

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](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
3. **I confirm I have read and understood the notices**: Selected.
4. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
5. View the **Keys and Endpoint** page for your Cognitive Services resource. You will need the endpoint and keys to connect from client applications.

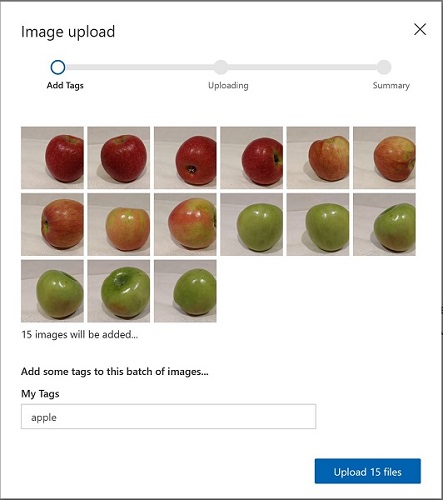
## Create a Custom Vision project

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.

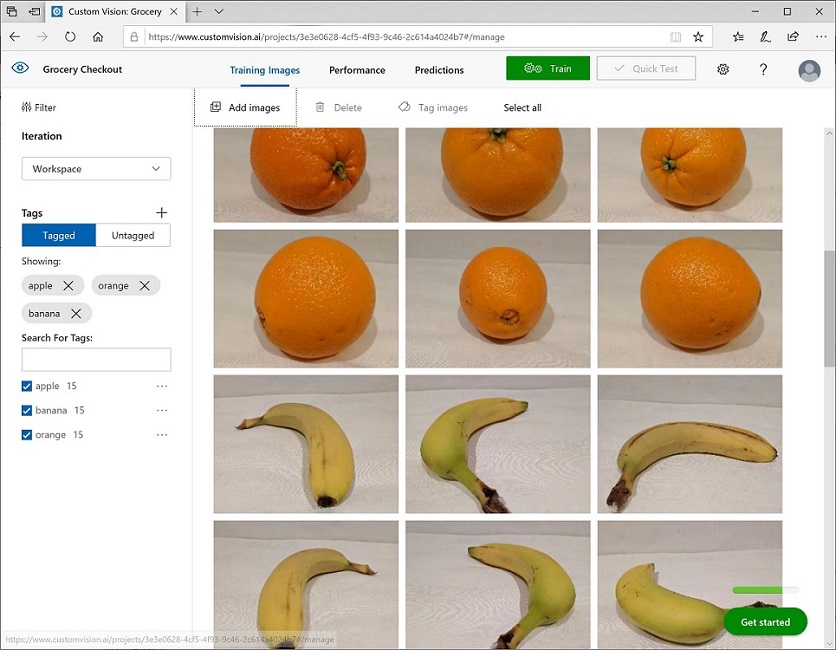
1. Download and extract the training images from <https://aka.ms/fruit-images>. These images are provided in a zipped folder, which when extracted contains subfolders called **apple**, **banana**, and **orange**.
2. In another browser tab, open the Custom Vision portal at [https://customvision.ai](https://customvision.ai/). 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

1. 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:



1. 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.
2. Explore the images you have uploaded in the Custom Vision project - there should be 15 images of each class, like this:

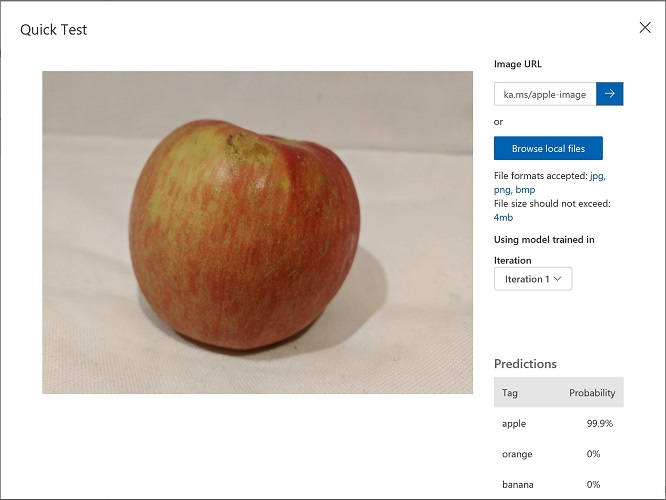


1. In the Custom Vision project, above the images, click **Train** to train a classification model using the tagged images. Select the **Quick Training** option, and then wait for the training iteration to complete (this may take a minute or so).
2. When the model iteration has been trained, review the 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:



1. 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

Same as in previous unit.

## 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.

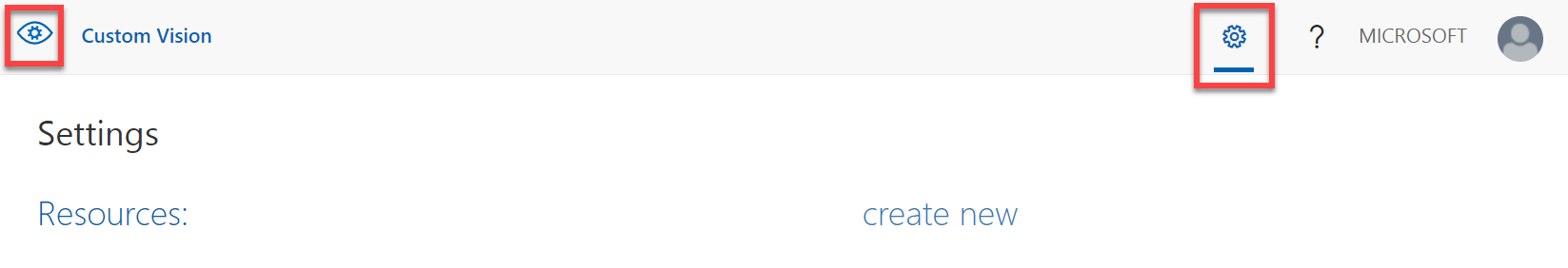
git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900

1. 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: code .
2. 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.
3. 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 (you reviewed it after you published the image classification model).

Paste it into the code editor, replacing the **YOUR\_PREDICTION\_URL**.

Get the prediction key. Click on the project gallery page\* icon on the top left hand side of the custom vision portal. Then click on the settings icon on the top right hand side of the custom vision portal. Look for your prediction resource and click on it.



Copy the prediction key. 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:

$predictionUrl="https..."

$predictionKey ="1a2b3c4d5e6f7g8h9i0j...."

1. 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.

1. We will classify img, In the PowerShell pane, enter the following commands to run the code:

cd ai-900

./classify-image.ps1 1

1. Review the prediction, which should be **apple**.

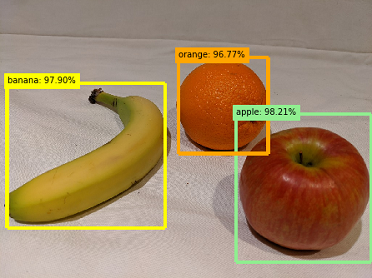
Detect objects in images with the Custom Vision service

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.

What is object detection?

Let's look at example of object detection. Consider the following image (image with fruits). 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.

**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 "IterationX", 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 apps.

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.

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

1. Open the Azure portal at [https://portal.azure.com](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.

## Create a Custom Vision project

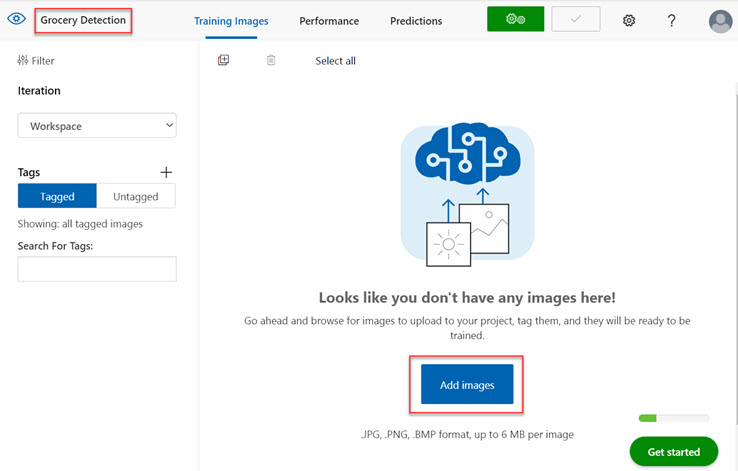
To train an object detection model, you need to create a Custom Vision project based on your training resource. To do this, you'll use the Custom Vision portal.

1. In a new browser tab, open the Custom Vision portal at [https://customvision.ai](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.

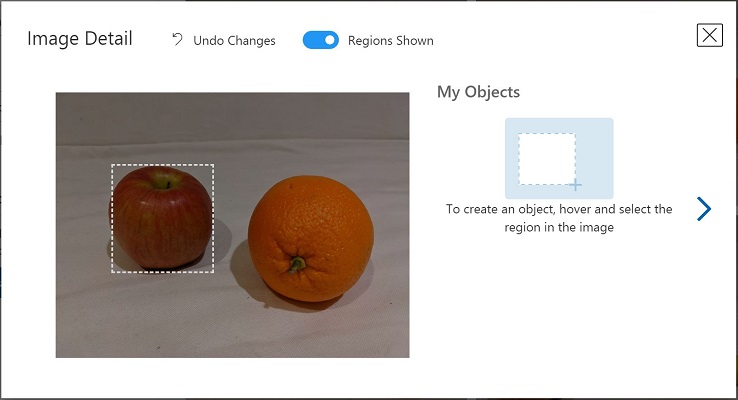
## Add and tag images

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](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.

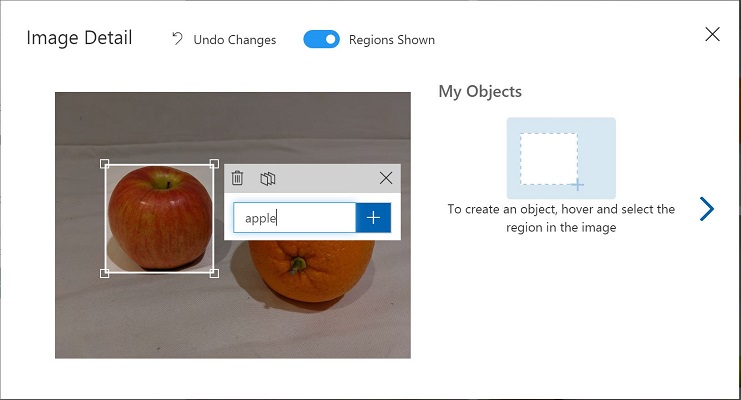


1. After the images have been uploaded, select the first one to open it.
2. Hold the mouse over any object in the image until an automatically detected region is displayed like the image below. Then select the object, and if necessary resize the region to surround it.

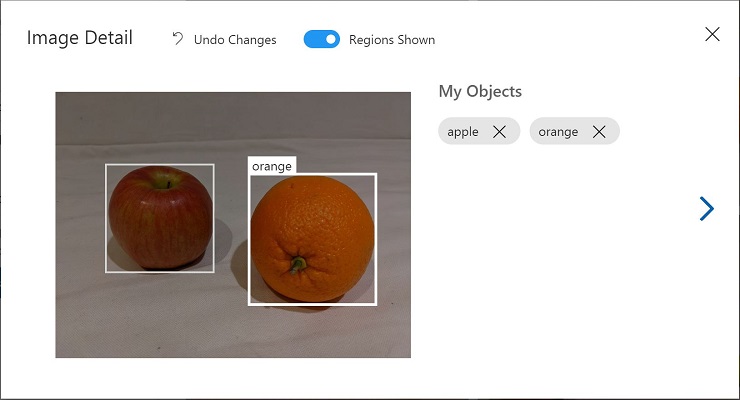


Alternatively, you can simply drag around the object to create a region.

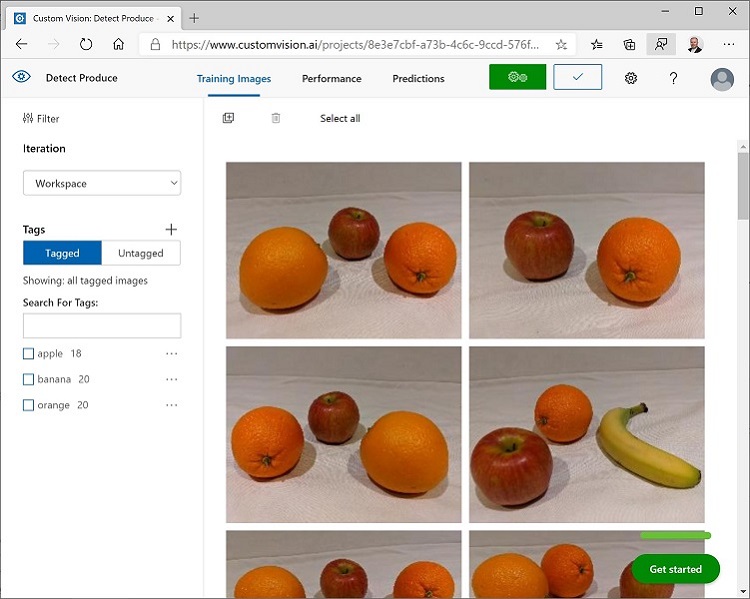
1. When the region surrounds the object, add a new tag with the appropriate object type (apple, banana, or orange) as shown here:



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



1. 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.
2. 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 accuracy 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

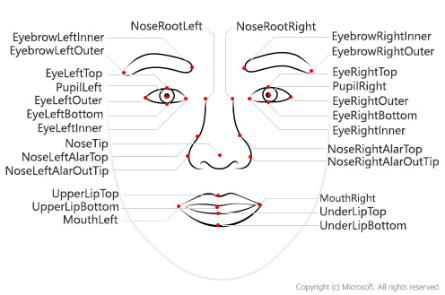
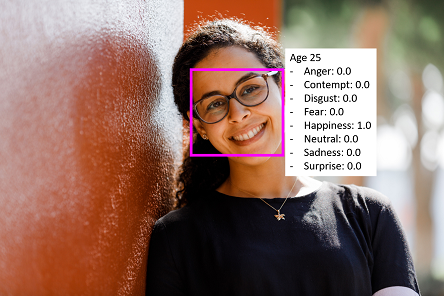
Same as in the previous unit.

## 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. Same as in previous module.

Detect and analyse faces with the Face service

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 from which you can infer information about a person, such as their perceived age or perceived emotional state, like this:
* **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.

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 determining age.
* **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.

## Face

Face offers the widest range of facial analysis capabilities. Face currently supports the following functionality:

* Face Detection
* Face Verification
* Find Similar Faces
* Group faces based on similarities
* Identify people

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:

* **Age**: a guess at an age
* **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)
* **Emotion**: what emotion is displayed
* **Exposure**: aspects such as underexposed or over exposed and applies to the face in the image and not the overall image exposure
* **Facial hair**: the estimated facial hair presence
* **Glasses**: if the person is wearing glasses
* **Hair**: the hair type and hair color
* **Head pose**: the face's orientation in a 3D space
* **Makeup**: whether the face in the image has makeup applied
* **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
* **Smile**: whether the person in the image is smiling

## 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.

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 sunglasses or 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, analyze, or identify human faces. For example, suppose the retail company 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. One way to accomplish this is to perform facial detection and analysis - in other words, determine if there are any faces in the images, and if so analyze their features.

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.

## Create a Cognitive Services resource

Same as in the previous unit.

## Run Cloud Shell

Same as in the previous unit.

## 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.
2. git clone https://github.com/MicrosoftLearning/AI-900-AIFundamentals ai-900
3. 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: code .
4. 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.
5. 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. You may need to use the separator bar to adjust the screen area as you work with the **Keys and Endpoint** and **Editor** panes.
6. After pasting the key and endpoint values, the first two lines of code should look similar to this:

$key="1a2b3c4d5e6f7g8h9i0j...."

$endpoint="https..."

1. 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:



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

cd ai-900

./find-faces.ps1 store-camera-1.jpg

1. Review the details of the faces found in the image, which include:
   * The location of the face in the image
   * The approximate age of the person
   * An indication of the emotional state of the person (based on proportional scores for a range of emotions)

Read text with the Computer Vision service

The ability for computer systems to process written or printed text is an area of 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.

## 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.

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.

## 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

Same as in the previous unit.

## Run Cloud Shell

Same as in the previous unit.

## 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.

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

1. 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: code .
2. 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:
3. 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. 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:

$key="1a2b3c4d5e6f7g8h9i0j...."

$endpoint="https..."

1. 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.

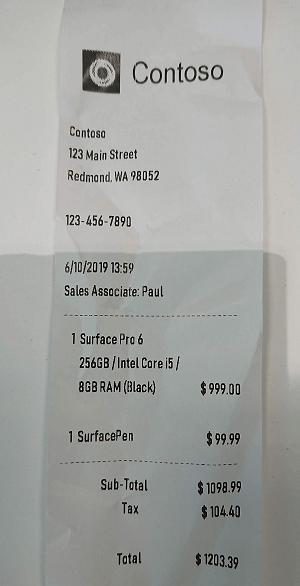
Let's use the **OCR** API, which enables you to synchronously analyze an image and read any text it contains. In this case, you have an advertising image for the fictional Northwind Traders retail company that includes some text.

1. In the PowerShell pane, enter the following commands to run the code to read the text:

cd ai-900

./ocr.ps1 advert.jpg

1. 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.

Analyse receipts with the Form Recognizer service

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.

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.

## 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. 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

Same as in the previous unit.

## Run Cloud Shell

Same as in the previous unit.

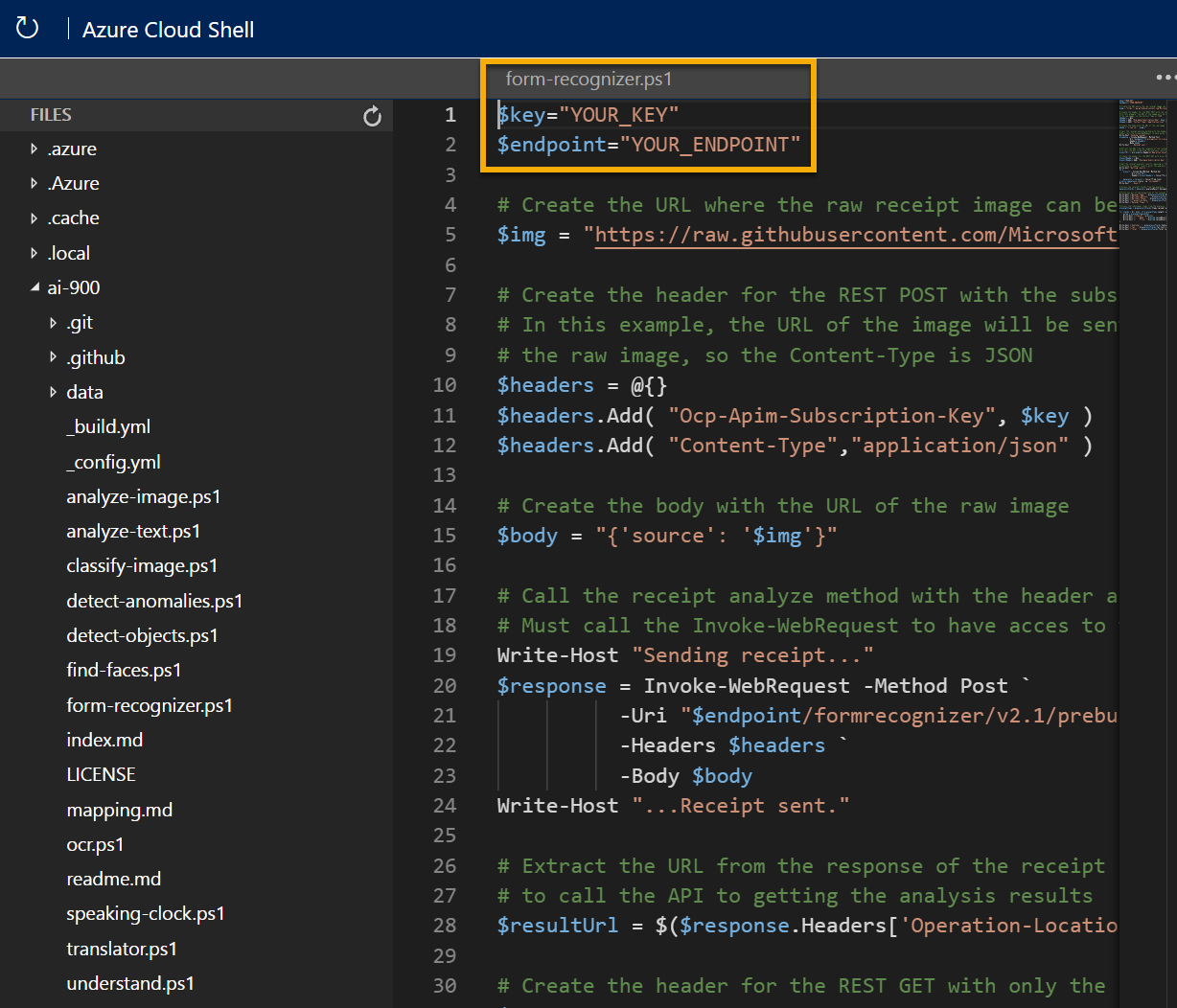
## Configure and run a client application

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.

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

1. 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: code .
2. 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.



1. 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. 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:

$key="1a2b3c4d5e6f7g8h9i0j...."

$endpoint="https..."

1. 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:
2. In the PowerShell pane, enter the following commands to run the code to read the text:

cd ai-900

./form-recognizer.ps1

1. 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.

# Explore natural language processing

Analyse text with the Language service

Analysing 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.

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."

"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.

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 |
| --- | --- | --- |
| Seattle | Location |  |
| 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](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 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 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.

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

1. 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: code .
2. 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:
3. 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\_ENDPOINT** placeholder values respectively. 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 lines of code should look similar to this:

$key="1a2b3c4d5e6f7g8h9i0j...."

$endpoint="https..."

1. 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.
2. In the Cloud Shell, enter the following command to run the code:

cd ai-900

./analyze-text.ps1 review1.txt

1. 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.

1. Review the output.
2. In the PowerShell pane, enter the following command to run the code: ./analyze-text.ps1 review2.txt
3. 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.

Recognize and synthesize speech

Increasingly, we expect 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 ex. "turn off the lights", and soliciting verbal answers to questions ex. "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:

* 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 provision an appropriate resource in your Azure subscription. You can choose to provision 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 supported languages: [Speech-to-text languages](https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support#speech-to-text) & [Text-to-speech languages](https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support#text-to-speech).

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](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

Same as in the previous units.

## 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.

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

1. 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: code .
2. 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:
3. 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. 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:

$key = "1a2b3c4d5e6f7g8h9i0j...."

$region="somelocation"

1. 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**.
2. 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:
3. In the PowerShell pane, enter the following command to run the code:

cd ai-900

./speaking-clock.ps1

1. 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.

Translate text and speech

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.

AI 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 turned 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 translation in Azure

Microsoft Azure provides cognitive services that support translation. Specifically, you can use the following services:

* The **Translator Text** service, which supports text-to-text translation.
* The **Speech** service, which enables speech-to-text and speech-to-speech translation.

## Azure resources for Translator Text and Speech

Before you can use the Translator Text or Speech services, you must provision appropriate resources in your Azure subscription.

There are dedicated **Translator Text** 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 Text service

The Translator Text 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.

### Optional Configurations

The Translator Text 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 Text 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](https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support#speech-translation).

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. 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 Translation 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 Translation 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. Open the Azure portal at [https://portal.azure.com](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 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, 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. 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

Same as in the previous units.

## 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.

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

1. 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: code .
2. In the **Files** pane on the left, expand **ai-900** and select **translator.ps1**. This file contains some code that uses the Translator service:
3. 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\_LOCATION** and **YOUR\_KEY** placeholder values respectively. 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 location values, the first lines of code should look similar to this:

$key="1a2b3c4d5e6f7g8h9i0j...."

$location="somelocation"

1. 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**.
2. The sample client application will use the Translator service to do several tasks:
3. Translate text from English into French, Italian, and Chinese.
4. Translate audio from English into text in French
5. 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.
6. In the Cloud Shell pane, enter the following command to run the code:

cd ai-900

./translator.ps1

1. Review the output. Did you see the translation from text in English to French, Italian, and Chinese? Did you see the English audio translated text in French?

Create a language model with Conversational Language Understanding

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 AI grows ever more sophisticated, this kind of conversational interaction with apps 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 apps, reservation systems, 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?" |  |

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.

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.

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](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:
3. 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**: Choose either the West US 2 or West Europe region
   * **Name**: Enter a unique name.
   * **Pricing tier**: Free (F0) (If this tier is not available, select Standard (S))
   * **Legal Terms**: Agree
   * **Responsible AI Notice**: Agree
4. Review and create the resource, and wait for deployment to complete. Then go to the deployed resource.
5. View the **Keys and Endpoint** page for your Language service resource. You will need the endpoint and keys to connect from client applications.

### 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.

1. In a new browser tab, open the Language Studio portal at [https://language.azure.com](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.
3. 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 your language resource, and click **Switch resource**.
   * At the top of the page, click **Language Studio** to return to the Language Studio home page.
4. At the top of the portal, in the **Create new** menu, select **Conversational language understanding**.
5. In the **Create a project** dialog box, in the **Choose project type** page, select **Conversation**, and click **Next**.
6. On the **Enter basic information** page, enter the following details and click **Next**:
   * **Choose project type**: Conversation project
   * **Name**: HomeAutomation
   * Use this exact name
   * **Description**: Simple home automation
   * **Utterances primary language**: English
   * **Enable multiple languages in project**: Do not select
7. On the Review and finish page, click **Create**.

### Create intents 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 **Build schema** pane on the left, 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. Click the **switch\_on** intent. It will take you to the **Tag utterances** 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. On the **Tagging** pane on the right-hand side of the screen, select **Add entity** and type **device** (in lower-case) and select **Done**.
4. In the turn the light on utterance, highlight the word "light". Then in the list that appears, in the Search for an entity box select **device**.
5. Now create a second utterance for the **switch\_on** intent. Type the phrase ***switch on the fan*** next to the **switch\_on** intent. Then select the word "fan" and assign it to the **device** entity you created previously.
6. The language service needs at least five different utterance examples for each intent to sufficiently train the language model. Add four more utterance examples to the **switch\_on** intent:
   * ***put the fan on***
   * ***put the light on***
   * ***switch on the light***
   * ***turn the fan on***

Tag fan or light with the device entity. When you're finished, verify that you have the following utterances and click **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 |

1. In the pane on the left, click **Build schema** 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).
2. Click on the **switch\_off** intent. It will take you to the **Tag utterances** page. Next to the **switch\_off** intent, add the utterance ***turn the light off*** and assign the word "light" to the **device** entity.
3. Add a second utterance to the **switch\_off** intent, with the utterance ***switch off the fan***. Then connect the word "fan" to the **device** entity.
4. Add four more utterance examples to the **switch\_off** intent.
   * ***put the fan off***
   * ***put the light off***
   * ***turn off the light***
   * ***switch the fan off***

Tag fan or light with the device entity. When you're finished, verify that you have the following utterances and click **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 **Train model**. Use the following settings:
   * **Train a new model**: Selected and choose a model name
   * **Run evaluation with training**: Enabled evaluation
   * 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 **Deploy model**.
2. Select your model name and click **Deploy Model**, and wait for it to be deployed.
3. When the model is deployed, click **Test model** on the left-hand side of the page, and then select your model.
4. Enter the following text, and then click **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)

1. Clear the text box and test the model with the following utterances:
   * 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 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.

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

1. 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:

cd ai-900

code .

1. 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 it needs the endpoint and key for your Language service model. You'll get the endpoint and key from the Language Studio.
2. Switch back to the browser tab containing Language Studio. Then in Language Studio, open the **Deploy 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***.
3. 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**. After pasting the key and endpoint values, the first lines of code should look similar to what you see below:

$endpointUrl="https://somename.cognitiveservices.azure.com/language/..."

$key = "0ab1c23de4f56gh7i8901234jkl567m8"

1. 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**.
2. In the PowerShell pane, enter the following command to run the code:

./understand.ps1 "Turn on the light"

1. Review the results. The app should have predicted that the intended action is to switch on the light.
2. Now try another command: ./understand.ps1 "Switch the fan off"
3. Review the results from this command. The app should have predicted that the intended action is to switch off the fan.
4. 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.

# Explore conversational AI

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 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.

Bot is often used as 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. 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.

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:

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](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
   * 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](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. 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 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

1. 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.
2. At the left of the Language Studio page, click **Deploy knowledge base**. Click **Deploy**.
3. 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.
4. 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 QnA Maker resource
   * **Location**: The same location as your QnA Maker service.
   * **Pricing tier**: F0
   * **App name**: Same as the ***Bot handle*** with ***.azurewebsites.net*** appended automatically
   * **SDK language**: Choose either C# or Node.js
   * **QnA Auth Key**: This should automatically be set to the authentication key for your knowledge base
   * **App service plan/location**: This should be set automatically to a suitable plan and location
   * **Application Insights**: Off
   * **Microsoft App ID and password**: Auto create App ID and password.
5. 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.)
6. 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).
7. 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.

# Explore decision support

Introduction to Anomaly Detector

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.

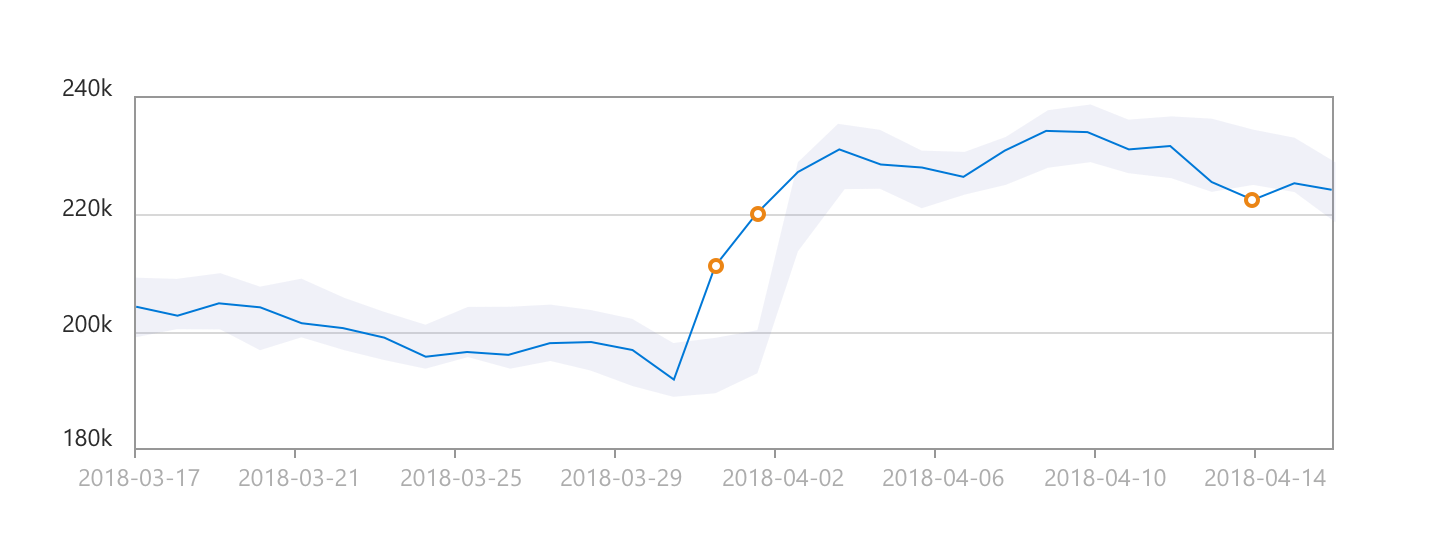
Other scenarios include:

* 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.

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.

{

"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](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

Same as in the previous units.

## 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 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.

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

1. 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: code .
2. 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:
3. 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. 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:

$key="1a2b3c4d5e6f7g8h9i0j...."

$endpoint="https..."

1. 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 Anomaly Detector service to analyze a file containing a series of date/times and numeric values.
2. In the PowerShell pane, enter the following commands to run the code:

cd ai-900

.\detect-anomalies.ps1

1. Review the results, noting that the final column in the results is **True** or **False** to indicate if the value is considered an anomaly or not.

# Explore knowledge mining

Introduction to Azure Cognitive Search

Every day we rely on search engines to find information quickly. Search engines are able to return results because they have content stored in a database known as an index.

Azure Cognitive Search is a private, enterprise, search solution that allows you to build a single fast index for content. The index can then be used for internal only use, or to enable searchable content on your public facing internet assets.

What is Azure Cognitive Search?

Integrating Azure Cognitive Search into apps and websites allows companies to provide customers with a rich search experience.

The Azure Cognitive Search service enables search over different types of content by letting you create and manage search indexes. You can import data from a variety of sources, with AI-powered indexing that can infer and extract searchable content from non-text sources. You decide what data is imported into the index, and set up indexers to pull that data into it, or push JSON formatted documents manually.

Azure Cognitive Search also lets you query search indexes. The 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) so Microsoft manages the infrastructure and availability, allowing your organization to benefit without the need to purchase or manage additional hardware resources.

### What can Azure Cognitive Search do?

Azure Cognitive Search exists to compliment existing technologies and provides a programmable search engine built on Apache Lucene, an open-source software library. Azure Cognitive Search is 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:

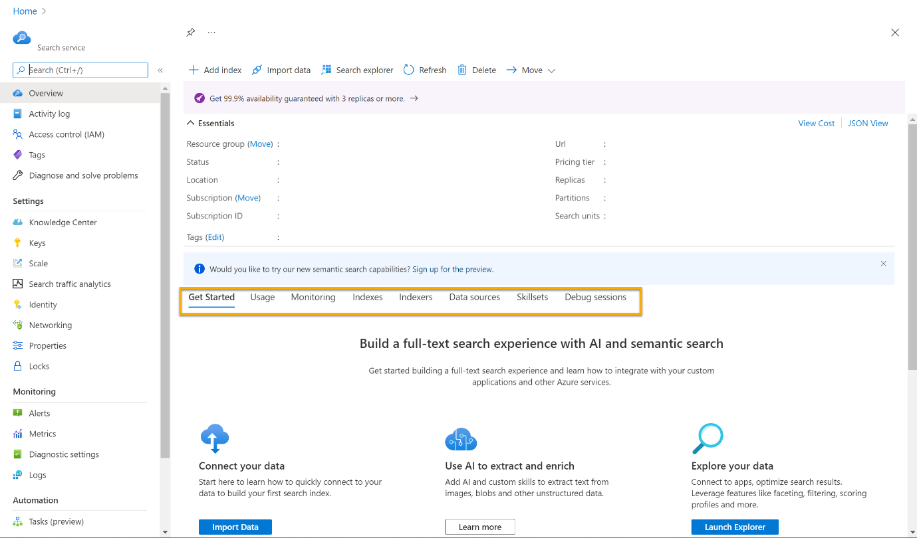
* **Data from any source**: Azure Cognitive Search accepts data from any source provided in JSON format, with auto crawling support for selected data sources in Azure.
* **Full text search and analysis**: Azure Cognitive Search offers full text search capabilities supporting both simple query and full 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 the same as those 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.

### Importing content to storage

In order to build with the Azure Search service, you will first need to upload your content to an Azure data source. Supported data storage sources include: Azure SQL Database, SQL Server on an Azure VM, Cosmos DB, Azure Blob storage, Azure Table storage

The content in storage will need to be exported from its original file type to JSON in order to populate an index. You will work with the index using Azure Cognitive Search service.

### Creating an Azure Cognitive Search resource

To use the Azure Cognitive Search service, you will 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 including:

* Usage
* Monitoring
* Indexes
* Indexers
* Data sources
* Skillsets
* Debug resources

You can create and customize Azure search index using the Azure portal or programmatically with the REST API or software development kits (SDKs).

## Use the Azure portal to create an index

Contained within the Azure Cognitive Search service dashboard is the Import data wizard, automates processes in the Azure portal to create various objects needed for the search engine. You 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 skillset, optional schedule, and optional configuration settings for error handing and base-64 encoding.
* **Skillset (Optional)**: A complete set of instructions for manipulating, transforming, and shaping content, including analysing 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 (Optional)**: Stores output from an AI enrichment pipeline in tables and blobs in Azure Storage for independent analysis or downstream processing.

Design an index

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.

### Index schema

The index includes a definition of the structure of the data in these documents, called its schema. Ex.:

{

"name": "hotels",

"fields": [

{ "name": "HotelId", "type": "Edm.String", "key": true, "filterable": true },

{ "name": "HotelName", "type": "Edm.String", "searchable": true, "filterable": false, "sortable": true, "facetable": false },

{ "name": "Rating", "type": "Edm.Double", "filterable": true, "sortable": true, "facetable": true },

{ "name": "Address", "type": "Edm.ComplexType",

"fields": [

{ "name": "StreetAddress", "type": "Edm.String", "filterable": false, "sortable": false, "facetable": false, "searchable": true },

{ "name": "City", "type": "Edm.String", "searchable": true, "filterable": true, "sortable": true, "facetable": true },

{ "name": "StateProvince", "type": "Edm.String", "searchable": true, "filterable": true, "sortable": true, "facetable": true },

{ "name": "PostalCode", "type": "Edm.String", "searchable": true, "filterable": true, "sortable": true, "facetable": true },

{ "name": "Country", "type": "Edm.String", "searchable": true, "filterable": true, "sortable": true, "facetable": true }

]

},

{ "name": "Location", "type": "Edm.GeographyPoint", "filterable": true, "sortable": true },

{ "name": "Rooms", "type": "Collection(Edm.ComplexType)",

"fields": [

{ "name": "Description", "type": "Edm.String", "searchable": true, "filterable": false, "sortable": false, "facetable": false, "analyzer": "en.lucene" },

{ "name": "BedOptions", "type": "Edm.String", "searchable": true }

]

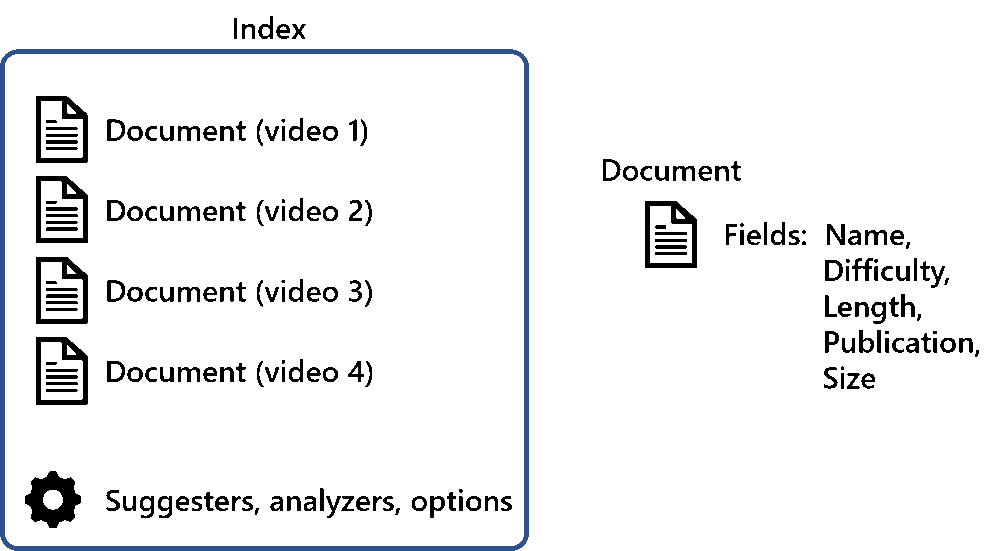
}

]

}

### Index attributes

An Azure Cognitive Search index can be thought of as a container of searchable documents. In database terms, the index is a table in the database, and each document is a row. 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.



Azure Cognitive Search needs to know how you'd like to search and display the fields in the documents, and 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?

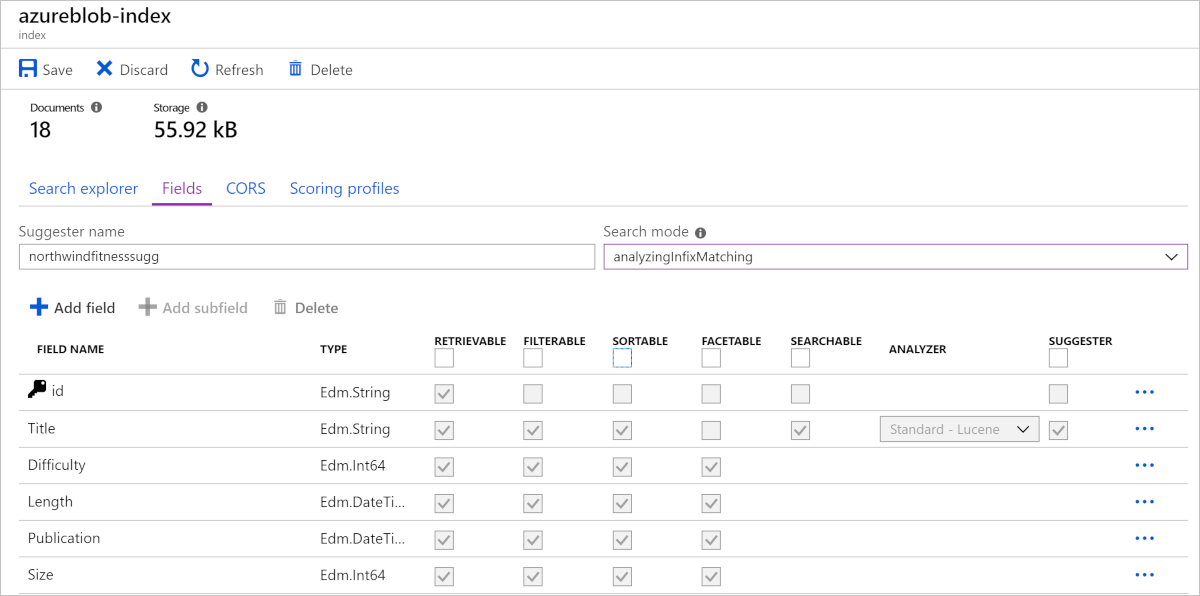
### Field behaviors

When creating your index, you need to choose the behaviors each field supports. The available options depend on what type of data is stored in the field:

* **Retrievable**: can this field be returned in the search results
* **Filterable**: can this field be used in filter expressions
* **Sortable**: can this field be sorted on in order by queries
* **Facetable**: can this field be used to group results to enable faceted navigation of the results
* **Searchable**: only available on text fields. Can this field be searched against
* **Analyzer to use**: only available on text fields. You choose the language analyzer for the field that processes text in a query.

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 indexer to build index

So far, we have discussed the index schema. Now we will describe how to create search documents and use them to populate the index. An index needs to be populated with JSON search documents before it can be queried.

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:

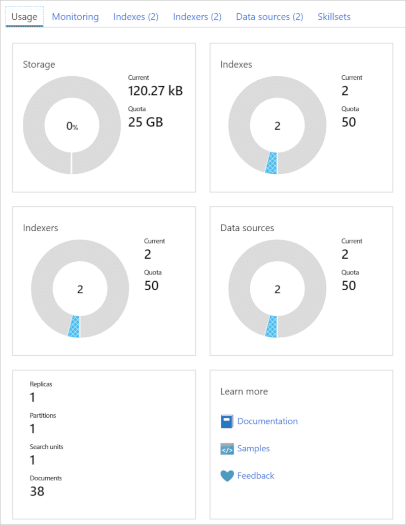
* **Push method**: JSON data is pushed into a search index via either the REST API or the .NET SDK. Pushing data has the most flexibility as it has no restrictions on the data source type, location, or frequency of execution.
* **Pull method**: Search service indexers can pull data from popular Azure data sources, and if necessary, export that data into JSON if it isn't already in that format.

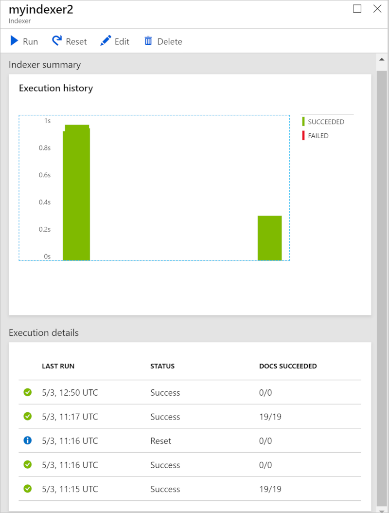
## Use the pull method to load data with an indexer

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.

Index data with Azure Cognitive Search

A good practice for index design is to use an iterative workflow. Below is a summary of how to use the Azure portal and pull method to create an index:

1. Create an Azure Cognitive Search resource in the portal.
2. Store content into one of the supported data sources:
   * Azure SQL Database
   * SQL Server on an Azure VM
   * Cosmos DB
   * Azure Blob storage
   * Azure Table storage
3. Move data from the data source into the index. Remember Azure Cognitive Search only supports JSON search documents. If the content moving into the index is not in JSON it must be exported to JSON. You can use the Azure portal to create an index. The portal has an Import data wizard to add fields, data types and assign behaviors to the index.
4. Query the index, analyze the results, and if necessary iterate on the index schema.
5. Use the Search explorer on the Azure portal to test searches in real time.

## 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.

## Enhance Azure Cognitive Search indexes with skills

Azure Cognitive Search has embedded AI. It works by using Azure Cognitive Services (specifically, Cognitive Search) to add Skillsets that include image processing, content extraction, or natural language processing (NLP). This makes it possible to index previously unsearchable, or unstructured content.

Image analyses skills range from OCR, to facial and landmark recognition, to analyzing an image for visual characteristics. Any text that can be lifted from an image becomes searchable in a search index. You can also perform sentiment analysis on text comments to look for customers having negative experiences. There are also skills that can detect language and perform text translation.

Query data in an Azure Cognitive Search index

After you import data into a search index, you can finally query the data.

## Search content using Azure Cognitive Search

Index and query design are closely linked. 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, including geo-search. Full Lucene is useful for advanced scenarios such as wildcard and fuzzy search, or term boosting.

### Query request Elements and Parsing

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 search: calm easy meditation (-"yoga" + -"pilates")

This query is trying to find a video that is calm and relaxing, but the person searching wants to exclude yoga and pilates.

Breaking the query into components, it's made up of search terms, (calm, easy, meditation), plus two verbatim phrases, "yoga" and "pilates", and operators (-, +, and ( )). The search terms can be matched in the search index in any order or location in the title of your videos. The two phrases will only match with exactly what is specified, so yoga 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. A title containing just calm would be a match. In this example, using -"yoga" would lead to the search results including all the titles that don't have the exact string "yoga" in it.

### Simple query syntax

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. This example search is written in this simple query syntax.

**Operators**

+: To ask for documents that contain all the search terms. For example, power + boxing would return all videos with both power **AND** boxing in the title.

|: To ask for documents that contain any of the search terms. For example, yoga | cardio would return all the videos with either yoga **OR** cardio in the title.

-: To ask for documents that don't contain a term. For example, -cycling would return all the videos that do **NOT** have cycling in the title.

\*: To include all matching suffix characters. If you need to search for words that begin with a term. For example, to return all the videos that have variations of crunch in their title, use crunch\*. The search would match with crunch, crunches, and crunched.

Remember that search queries default to match any of the search terms. You can change this behavior using the searchMode option. A query can be changed to match with all the search terms. If you don't pay attention to this, using a combination of terms (with some using the - operator), will lead to **OR** instead of **AND** searches. Append either &searchMode=all or &searchMode=any to specify the desired behavior.

calm easy meditation (-"iyengar yoga" + -"hot pilates")

Examining the search query again, can you anticipate the outcome? If you ran this query as is, the search engine wouldn't return the desired results. Given an any search, all videos that don't have (-"iyengar yoga" + -"hot pilates") in their titles would be considered a match, which is probably not what you want. There are two ways to rewrite this query to achieve the desired results. One way uses the operators to clarify the query:

(calm easy meditation) + (-"iyengar yoga" + -"hot pilates")

This search query groups the first three terms with the precedence operator ( ), so any of calm, easy, or meditation will be a match in the results. The + operator tells the search parser that the tile must have (calm, easy, **OR** meditation) **AND** (neither of the specific terms) in the result.

The other way is to change the search mode:

(calm | easy | meditation) (-"iyengar yoga" + -"hot pilates")&searchMode=all

Can you see that using &searchMode=all means that the first group of terms now needs to be separated by the **OR** | operator. If the query didn't contain the | operator, the search results would need to match (calm **AND** easy **AND** meditation) **AND** (neither of the phrases). Setting the search mode to all also means that the + operator becomes redundant, and can be removed.

### Handle results

After a query expression has been designed, you can choose how the results are shaped. If the result set is large, and you want to display them on a webpage, a typical user experience is to page the results:

Showing 1 - 5 or 36 videos or another way of displaying pagination Showing Page 3 of 10.

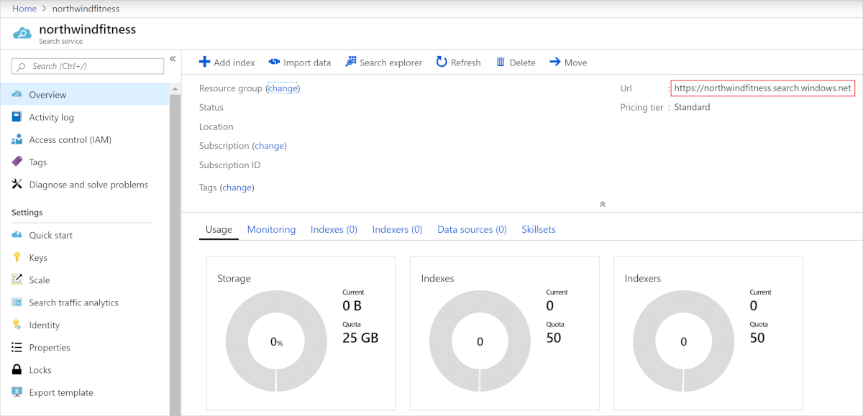
To support this experience, you limit the returned results with the &$top=X parameter. In the above example, you would append &$top=5 to your search query. If the user then wanted to see the next 5 videos, you would append &$top=5$skip=5. The skip parameter allows you to reach deeper into the results. The amount that needs to be skipped can be represented as the formula skip=(page\*top)-top. Using page 5 as an example, the parameter would be (5\*5)-5 = 20 so the value to skip is 20. The total count of results can be shown in the result set if you append &$count=true.

A useful feature to your customers is the ability to order the results on the length of exercise videos. For example, users may want to see the shortest videos first. If you're familiar with SQL queries, ordering the results works in a similar way. Append an &$orderby=fieldname asc|desc parameter to the end of the query string.

Exercise

Let's imagine you are working with a fitness startup that publishes exercise videos. You are asked to help build a search engine for customers to navigate through the video catalog. You will implement Azure Cognitive Search to provide a better search experience for customers.

## Create an Azure Cognitive Search resource

1. 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. **Instance Details**
   * **URL**: Enter a unique name, for example northwindfitness
   * **Location**: Choose any available region:
   * **Pricing tier**: S0
2. Select **Review + create**, and after you see the response **Validation Success**, select **Create**.
3. After deployment completes, select **Go to resource**. On its overview page you have the ability to add indexes, import data, and search created indexes.

## Run Cloud Shell

Same as in the previous units.

## Review content

Before you upload your content to storage, you want to take a look at its format. In this case, your database team has already exported your company's current video catalog into a JSON array. When you move this data from storage to the index, it will already be in the desired JSON document format. Recall that you could upload the raw video files to storage, but would need to export those files to JSON before populating your index. The content you are working with is in the following format:

[

{

"id": "cc74bc3d-95b4-457f-bf5e-59c577938034",

"Title": "Squats and Stars",

"Difficulty": "7",

"Length": "00:02:40",

"Publication": "2019-04-29 12:34:56",

"Tags": ["cardio","floor","burn"],

"Size": "346"

},

{

"id": "f94089de-d9f2-42d6-945d-276ae928564d",

"Title": "Full body workout",

"Difficulty": "9",

"Length": "00:15:30",

"Publication": "2019-04-24 11:14:06",

"Tags": ["cardio","floor","burn","free weights"],

"Size": "1897"

}, ...

Notice how each video has a title string, a difficulty rating integer, length in minutes, a publication date and time, and a file size in megabytes.

## Load content into Azure blob storage

There are several ways to upload content to Azure. In this case, we will use a script to load the content into Azure blob storage. Another way to upload content is step-by-step in the Azure portal, which we do not cover here.

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

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

1. 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: code .
2. In the **Files** pane on the left, expand **ai-900** and select **upload-content-to-storage.sh**. Edit the first line to add your resource group name. You can find your resource group name on the Overview page of your search service resource in the Azure portal. After pasting in your resource group name, the first line should look similar to this line:

export RESOURCE\_GROUP\_NAME="my-resource-group"

Make sure to save the document. 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**.

1. Copy the following commands and paste them into the Cloud Shell. Then press enter. The commands will go to the ai-900 folder, and use the script **upload-content-to-storage.sh** to create a storage account and container. It will also upload the JSON array of the video content to your storage blob.

cd ai-900

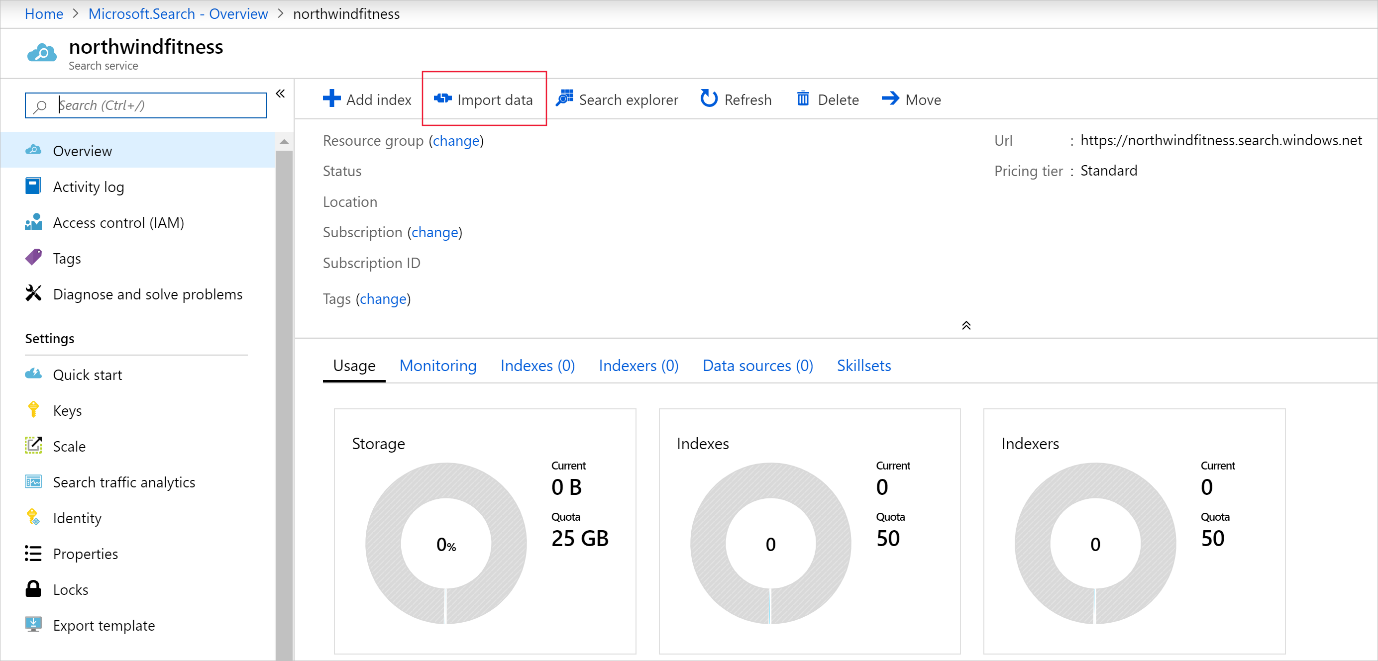
chmod +x upload-content-for-search.sh

./upload-content-for-search.sh

## Create an Azure Cognitive Search index

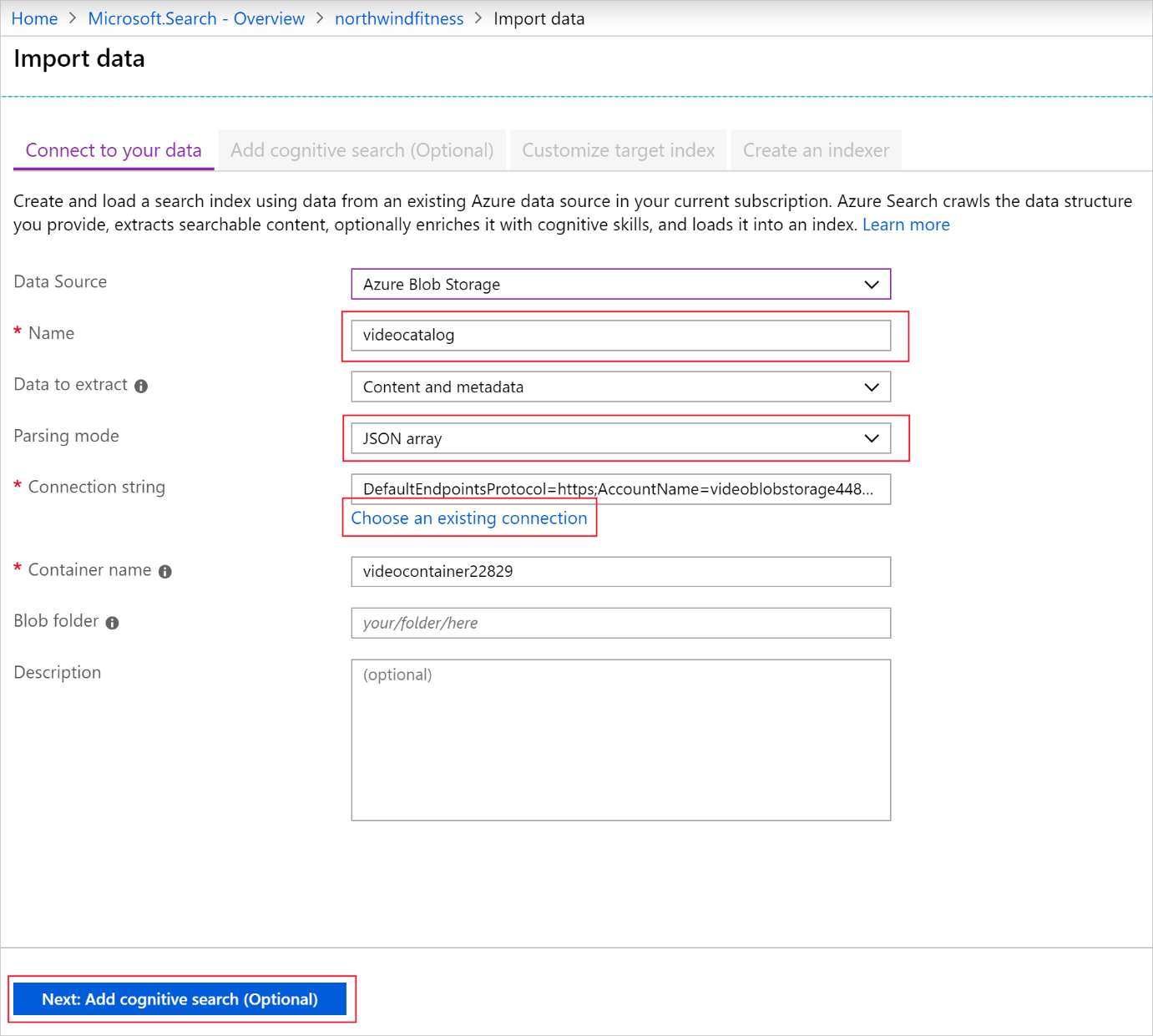
The Azure portal provides the Import data wizard that enables you to automatically create an index and indexer for supported data sources. You'll use the wizard to create an index, and import your search documents from storage into the Azure Cognitive Search index.

1. On the Azure portal menu or from the **Home** page, select **All resources**.
2. Select the Azure Cognitive Search resource you created to navigate to its overview page.
3. On the **northwindfitness** search Overview page, select **Import data**.

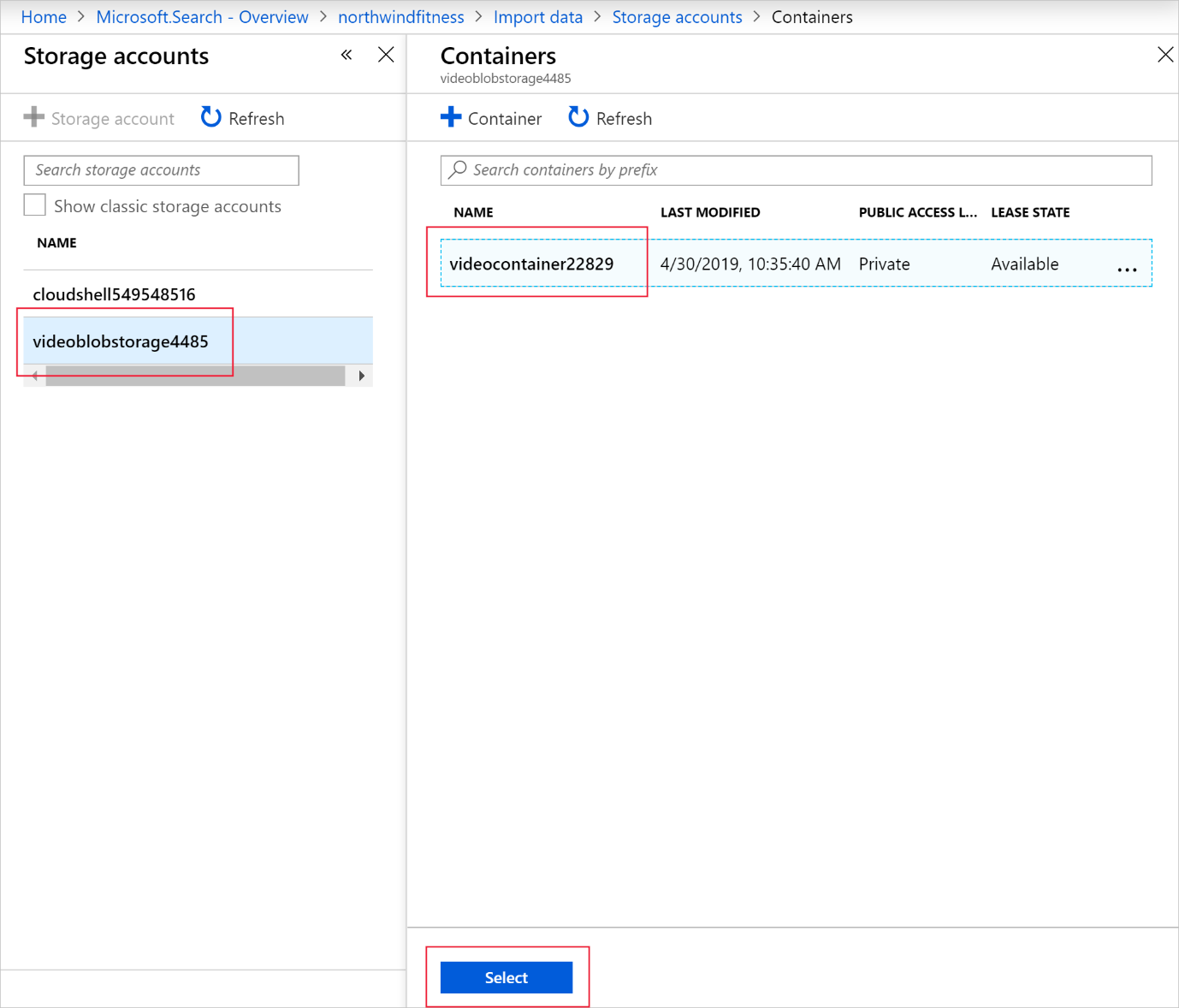


1. On the **Import data** page, complete the following fields.

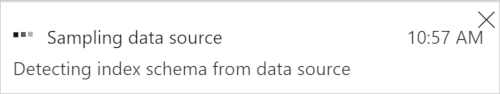
| Setting | Value |
| --- | --- |
| **Connect to your data** |  |
| **Data Source** | From the dropdown, select **Azure Blob Storage** |
| **Data source name** | Enter videocatalog |
| **Parsing mode** | From the dropdown, select **JSON array** |
| **Connection string** | Select **Choose an existing connection** link |



1. On the **Storage accounts** page, select the video storage account.

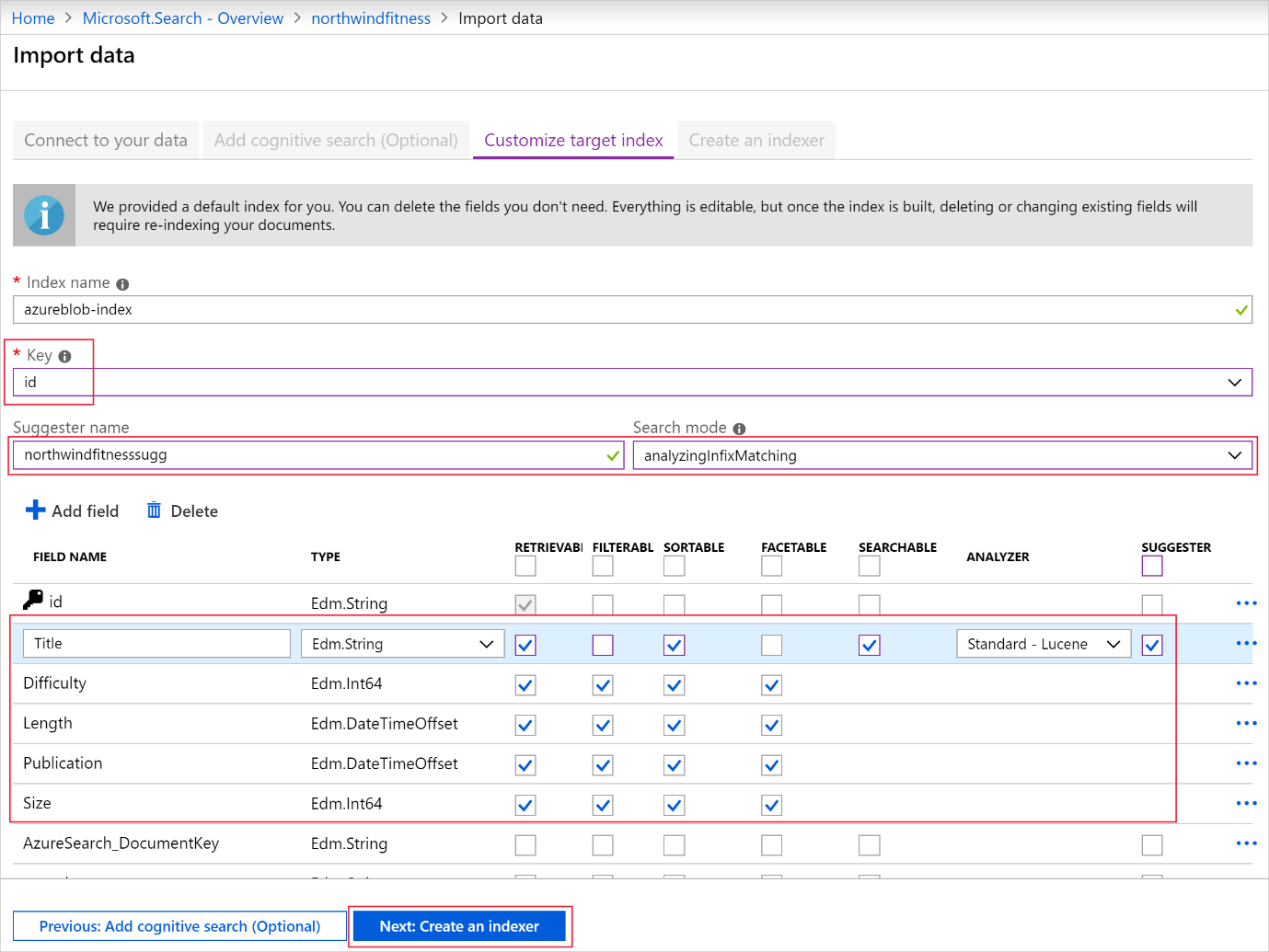


1. Select the video storage account.
2. Select the video container, and then select **Select**.
3. At the bottom of the page, select **Next: Add cognitive skills (Optional)**.



Azure Cognitive Search will read the contents of the JSON file, and create an index schema automatically.

1. On the **Add cognitive search (Optional)** tab, select **Skip to: Customize target index**.



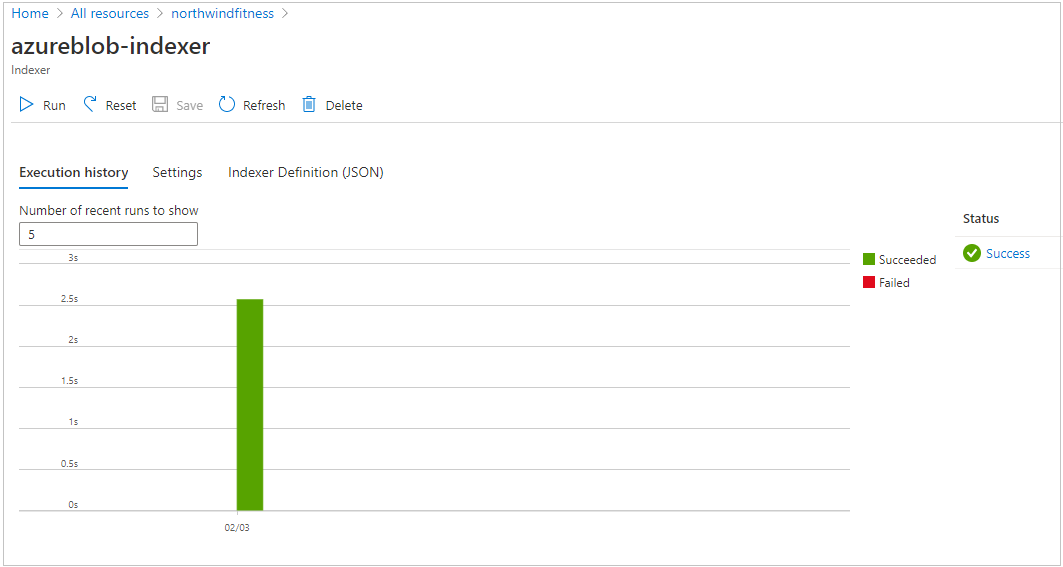
1. The **Customize target index** tab enables you to change the index schema created by the Import data wizard. The fields are populated by the wizard after reading the file in blob storage. Use the following table to complete the named fields:

| Field | Value |
| --- | --- |
| **Key** | From the dropdown, select **id** |
| **Suggester name** | **northwindfitness** |
| **Search mode** | From the dropdown, select **analyzingInfixMatching** |

1. Change the attributes and data types of the fields to match the following table:

| Field name | Type | Retrievable | Filterable | Sortable | Facetable | Searchable | Analyzer | Suggester |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Title** | Edm.String | ✔ |  | ✔ |  | ✔ | Standard - Lucene | ✔ |
| **Difficulty** | Edm.Int64 | ✔ | ✔ | ✔ | ✔ |  |  |  |
| **Length** | Edm.DateTimeOffset | ✔ | ✔ | ✔ | ✔ |  |  |  |
| **Publication** | Edm.DateTimeOffset | ✔ | ✔ | ✔ | ✔ |  |  |  |
| **Size** | Edm.Int64 | ✔ | ✔ | ✔ | ✔ |  |  |  |

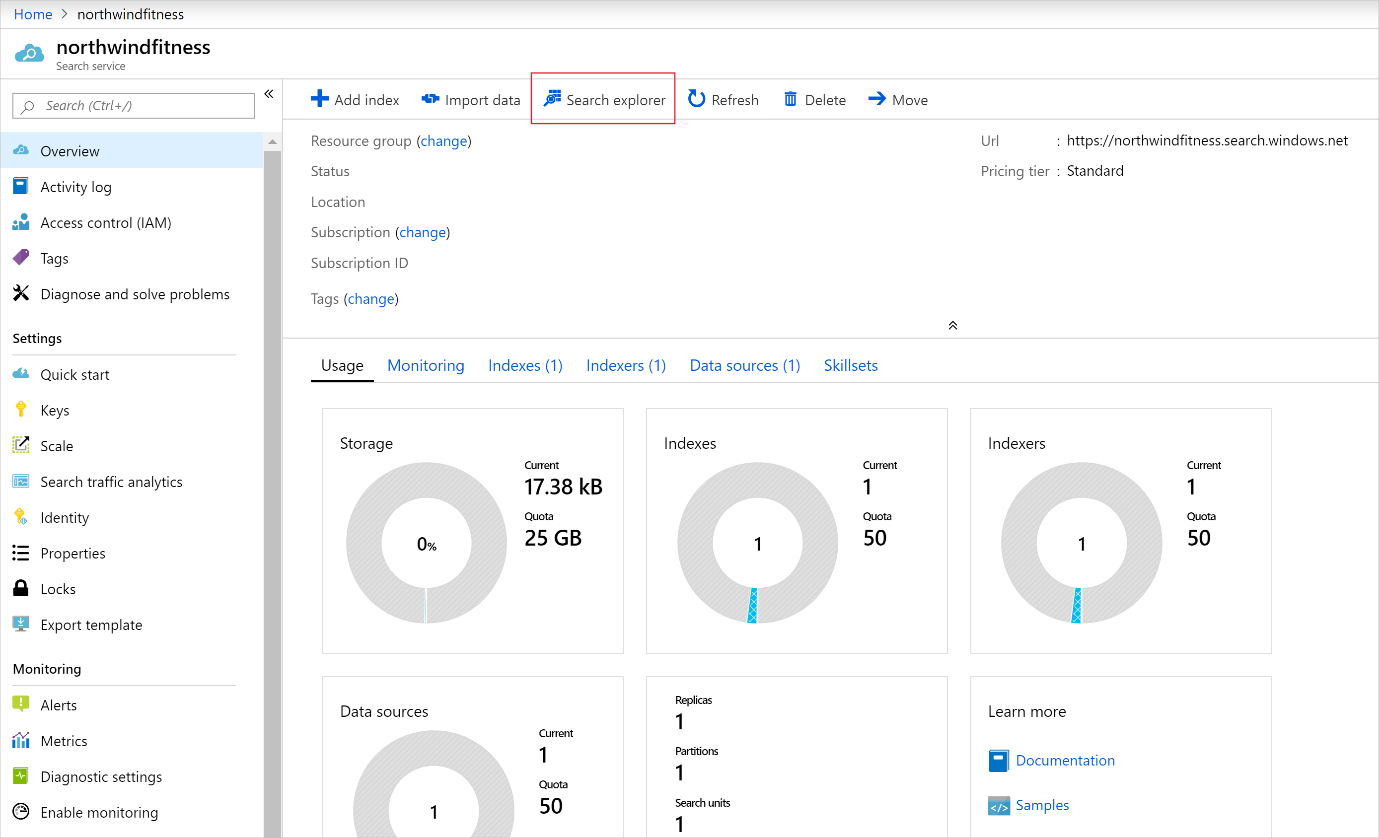
1. Select **Next: Create an indexer**.
2. On the **Create an indexer** tab, select **Submit** to begin building the indexer. When the process completes, the portal returns to the Search service overview page.
3. Select the **Indexers** tab, and then select **azureblob-indexer**.
4. At the top of **azureblob-indexer** page, select **Run**, and then select **Yes**.



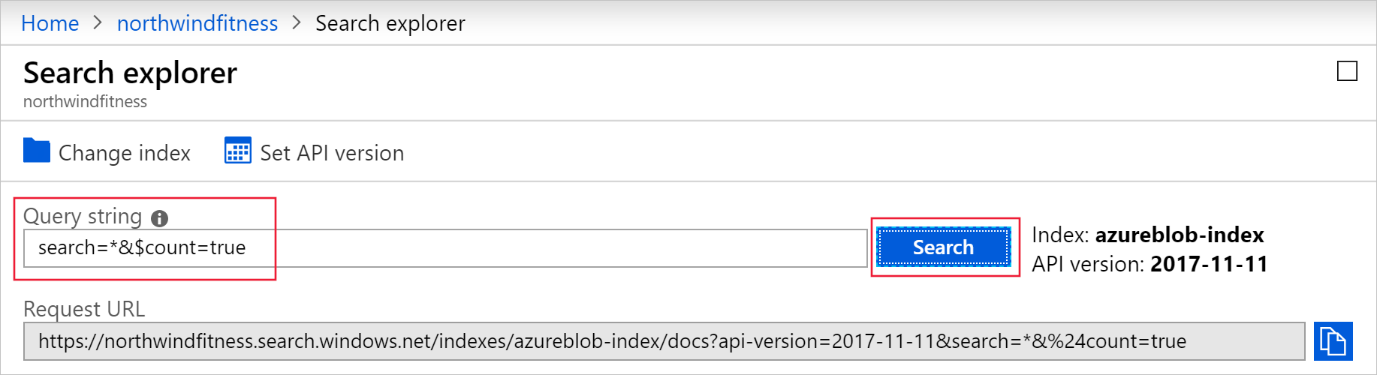
1. The indexer should import the video catalog, and show that 19 documents have been imported into the index.

## Query the index

Your web team has asked you to provide some example queries that the new search service can answer. You'll 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, review top results, and apply filters.



1. Scroll to the search service overview, and from the top menu bar, select **Search explorer**.



1. In the **Query string** field, enter search=\*&$count=true, and then select **Search**. The search query above returns all the documents in the search index, including a count of all the documents.
2. Enter yoga in the **Query string** field, and then select **Search**.
3. The search index should return a JSON document containing your search results. The matching documents are contained in the value array. Each item in the array is the data related to the video in the catalog. 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. Yoga Beginners is a better match because it begins with the search term.

# Sources

MS Azure fundamentals Certificate:

[Microsoft Certified: Azure AI Fundamentals - Learn | Microsoft Docs](https://docs.microsoft.com/en-us/learn/certifications/azure-ai-fundamentals/)

Examples of questions

[AI-900 Exam – Free Actual Q&As, Page 1 | ExamTopics](https://www.examtopics.com/exams/microsoft/ai-900/view/1/)

YouTube tutorial

[Azure AI Fundamentals Certification (AI-900) - Full Course to PASS the Exam - YouTube](https://www.youtube.com/watch?v=OwZHNH8EfSU&t=4844s)