Microsoft

Designing and Implementing a Data Science Solution on Azure

Exam Ref DP-100

Dayne Sorvisto

FREE SAMPLE CHAPTER |

Microsoft

Exam Ref DP-100 Designing and Implementing a Data Science Solution on Azure

Dayne Sorvisto

Exam Ref DP-100 Designing and Implementing a Data Science Solution on Azure

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

Hoboken, New Jersey

ISBN-13: 978-0-13-535060-7
ISBN-10: 0-13-535060-3

Library of Congress Control Number: 2024946348

$PrintCode

CREDITS

EDITOR-IN-CHIEF
Brett Bartow

EXECUTIVE EDITOR
Loretta Yates

ASSOCIATE EDITOR
Shourav Bose

DEVELOPMENT EDITOR
Songlin Qiu

MANAGING EDITOR
Sandra Schroeder

SENIOR PROJECT EDITOR
Tracey Croom

TECHNICAL EDITOR
Francesco Esposito

COPY EDITOR
Dan Foster

INDEXER
Timothy Wright

PROOFREADER
Barbara Mack

EDITORIAL ASSISTANT
Cindy Teeters

COVER DESIGNER
Twist Creative, Seattle

COMPOSITOR AND GRAPHICS
codeMantra

Contents at a glance

This page intentionally left blank

Contents

Acknowledgments

I'd like to thank my mom Allison and wife Kirsten for their insights, love, and support during the development of this book. I would also like to acknowledge my late grandfather Bruce for his advice and motivation and my dog Lucy for all the hours she kept me company while writing.

This page intentionally left blank

About the Author

DAYNE SORVISTO is a seasoned data engineer and technical author (*MLOps Lifecycle Toolkit*). Dayne has held senior technical positions including Staff Data Engineer, Software Developer, and Senior Machine Learning Engineer, and has a Master's degree in Pure Mathematics. You can connect with Dayne on LinkedIn at *linkedin.com/in/daynesorvisto* or visit his website *wyattsolutions.co* to learn more.

This page intentionally left blank

Introduction

This book is intended to cover all the skills measured in the exam DP-100 Designing and Implementing a Data Science Solution on Azure. You'll find in each chapter a combination of step-by-step instructional content as well as accompanying high-level theoretical material. The aim is to show you the buttons you need to click in order to carry out the tasks required as well as covering key concepts that you need to understand when designing a data science solution. Ultimately, we cover not only the how but also the why.

This book is written for IT professionals who intend to take the DP-100 exam as well as data engineers, data scientists, and other data professionals who want to learn to design and implement a data science solution in Azure. In addition to the exam material, the book is meant to enrich your knowledge of Azure Machine Learning by using it to implement machine learning operations in Azure and to design end-to-end data science solutions.

This book covers every major topic area found on the exam, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions, and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the "Need more review?" links you'll find in the text to find more information and take the time to research and study the topic.

Organization of this book

This book is organized by the "Skills measured" list published for the exam. The "Skills measured" list is available for each exam on the Microsoft Learn website: *microsoft.com/learn*. Each chapter in this book corresponds to a major topic area in the list, and the technical tasks in each topic area determine a chapter's organization. If an exam covers six major topic areas, for example, the book will contain six chapters.

Preparing for the exam

Microsoft certification exams are a great way to build your résumé and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. Although there is no substitute for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. This book is *not* designed to teach you new skills.

We recommend that you augment your exam preparation plan by using a combination of available study materials and courses. For example, you might use the *Exam Ref* and another study guide for your at-home preparation and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you. Learn more about available classroom training, online courses, and live events at *microsoft.com/learn*.

Note that this *Exam Ref* is based on publicly available information about the exam and the author's experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both onpremises and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL MICROSOFT CERTIFICATIONS**

For information about Microsoft certifications, including a full list of available certifications, go to *microsoft.com/learn*.

Access the Exam Updates chapter and online references

The final chapter of this book, "Exam DP-100: Designing and Implementing a Data Science Solution on Azure—updates" will be used to provide information about new content per new exam topics, content that has been removed from the exam objectives, and revised mapping of exam objectives to chapter content. The chapter will be made available from the link below as exam updates are released.

Throughout this book are addresses to webpages that the author has recommended you visit for more information. Some of these links can be very long and painstaking to type, so we've shortened them for you to make them easier to visit. We've also compiled them into a single list that readers of the print edition can refer to while they read.

The URLs are organized by chapter and heading. Every time you come across a URL in the book, find the hyperlink in the list to go directly to the webpage.

Download the Exam Updates chapter and the URL list at *MicrosoftPressStore.com/ERDP100/downloads*

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

MicrosoftPressStore.com/ERDP100/errata

If you discover an error that is not already listed, please submit it to us at the same page.

For additional book support and information, please visit *MicrosoftPressStore.com/Support*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *support.microsoft.com*.

Stay in touch

Let's keep the conversation going! We're on X / Twitter: *twitter.com/MicrosoftPress*.

This page intentionally left blank

CHAPTER 2

Explore data and train models

We're about to dive deep into the heart of the machine learning lifecycle: exploring data and training models. This chapter is designed to equip you with the skills and knowledge needed to handle data effectively and create optimized machine learning models using Azure Machine Learning.

Data exploration is the first step in any successful machine learning project. It's where you'll get to know your data, understand its characteristics, and prepare it for modeling. You'll learn how to access and wrangle data using Azure's data assets and datastores, making your data ready for the challenges ahead.

Model training is where the magic happens. You'll discover how to create models using the Azure Machine Learning Designer, leverage the power of automated machine learning for various data types, and even dive into custom model training using notebooks and Python SDKv2. We'll also cover hyperparameter tuning, a crucial step in optimizing your models for better performance.

By the end of this chapter, you'll have a solid understanding of how to explore data and train models in Azure Machine Learning, setting the stage for deploying and managing your models in the real world.

EXAM TIP

Follow the steps outlined in Skill 2.1 and Skill 2.2 to understand how to create models, data assets, and datastores in Azure Machine Learning Designer and by using the Python SDKv2. While you may prefer one over the other, pay close attention to the wording on the exam when a question asks about the Azure Machine Learning Designer or the SDK since it might impact how you answer the question.

Skills covered in this chapter:

- Skill 2.1: Explore data by using data assets and datastores
- Skill 2.2: Create models by using the Azure Machine Learning Designer
- Skill 2.3: Use automated machine learning to explore optimal models
- Skill 2.4: Use notebooks for custom model training
- Skill 2.5: Tune hyperparameters with Azure Machine Learning

As you journey through this chapter, remember that exploring data and training models are iterative processes. With each iteration, you'll gain deeper insights into your data and refine your models for better accuracy and performance. After acquiring the five skills in this chapter, you will be able to combine them to build training pipelines using automated machine learning and tune hyperparameters to iteratively improve the model performance using Azure Machine Learning.

Skill 2.1: Explore data by using data assets and datastores

In the process of developing a machine learning model, one of the first steps is exploring and understanding the data you're working with. Skill 2.1 focuses on the exploration of data using Azure Machine Learning's data assets and datastores. This skill is essential for data scientists and analysts who need to access, wrangle, and prepare data for model training. By mastering these techniques, you'll be able to create a solid foundation for building accurate and efficient machine learning models.

This skill covers how to:

- Access and wrangle data during interactive development
- Wrangle interactive data with Apache Spark

Access and wrangle data during interactive development

In this section, you'll learn how to access data stored in Azure Machine Learning datastores and perform data wrangling operations interactively. This is crucial for exploratory data analysis and preprocessing steps before model training.

Imagine you're working on a project to predict customer churn based on historical transaction data. You need to access this data from an Azure Blob Storage, clean it, and perform feature engineering to prepare it for model training. You'll use Python in a Jupyter Notebook environment within Azure Machine Learning to load the data, handle missing values, encode categorical variables, and normalize numerical features.

NEED MORE REVIEW? **WRANGLING DATA IN AZURE MACHINE LEARNING**

You can read more about wrangling data in Azure Machine learning at *https://learn.microsoft.com/en-us/azure/machine-learning/how-to-access-data-interactives*

To demonstrate the end-to-end process of exploring data and training a model to predict customer churn using Azure Machine Learning, we'll go through the following steps:

1. Before you begin, make sure you have an Azure Machine Learning workspace set up. You can create one using the Azure portal or the Azure Machine Learning SDK. For detailed instructions, see the section "Manage an Azure Machine Learning workspace" in Chapter 1.
2. Create a datastore: Link your Azure Blob Storage account to your Azure Machine Learning workspace by creating a datastore. For a review of managing data in Azure Machine Learning, see Chapter 1, "Manage data in an Azure Machine Learning workspace." For convenience, here are the instructions for creating a datastore:
    1. Navigate to the Datastores section in the left menu and select the Datastores option under the Data section.
    2. Add a new datastore by clicking the New Datastore (+) button at the top of the Datastores page.
3. Access and explore data: Use the Azure Machine Learning SDK to access your data and perform exploratory data analysis (EDA) using Pandas.
4. Preprocess and prepare data: Clean the data, handle missing values, encode categorical variables, and normalize numerical features.

Listing 2-1 shows a sample code snippet that demonstrates these steps: Setting up your Azure Machine Learning workspace, creating and accessing a datastore, and preprocessing data. Next, we will show how to implement these steps using `Workspace` and `Datastore` objects.

LISTING 2-1 Implementing preprocessing steps needed to train a model

```
from azureml.core import Workspace, Datastore
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Set up your Azure Machine Learning workspace
subscription_id = 'your-subscription-id'
resource_group = 'your-resource-group'
workspace_name = 'your-workspace-name'
workspace = Workspace(subscription_id, resource_group, workspace_name)

# Create a datastore (if not already created)
datastore_name = 'your-datastore-name'
container_name = 'your-container-name'
account_name = 'your-storage-account-name'
datastore = Datastore.register_azure_blob_container(workspace=workspace,
datastore_name=datastore_name,
container_name=container_name,
account_name=account_name)
```

```
# Access data from the datastore
datastore_path = [(datastore, 'path/to/your/data.csv')]
data = Dataset.Tabular.from_delimited_files(path=datastore_path)
df = data.to_pandas_dataframe()

# Explore and preprocess the data (assuming 'Churn' is the target variable and it's a
binary classification problem)
df.fillna(df.mean(), inplace=True)  # Handle missing values
df = pd.get_dummies(df, drop_first=True)  # Encode categorical variables
X = df.drop('Churn', axis=1)
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Normalize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Logistic Regression model we imported
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)
# Predict on the test set
y_pred = logreg.predict(X_test_scaled)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred) print(f'Accuracy: {accuracy:.2f}')
```

In this example, we've used a logistic regression model for simplicity, but you can replace it with any other model suitable for your use case. Remember to adjust the data preprocessing steps according to the specific requirements of your dataset and the model you choose.

Wrangle interactive data with Apache Spark

Apache Spark is a powerful tool for handling large-scale data processing and analysis. In this topic, you'll explore how to use Apache Spark within Azure Machine Learning to wrangle data interactively.

Consider a scenario where you're dealing with a massive dataset of social media posts, and you need to perform sentiment analysis. The dataset is too large to process on a single machine, so you decide to use Apache Spark to process and clean the data in parallel. The decision to use Spark is reasonable if you have more than 20 GB of data, for example, where the data is too large to fit completely in memory on a single machine. You can also use Spark for much larger data volumes up to petabytes of data. You'll learn how to initialize a Spark session in Azure Machine Learning, read the data, and perform text preprocessing tasks like tokenization, stopword removal, and stemming. Figure 2-1 shows the workflow.

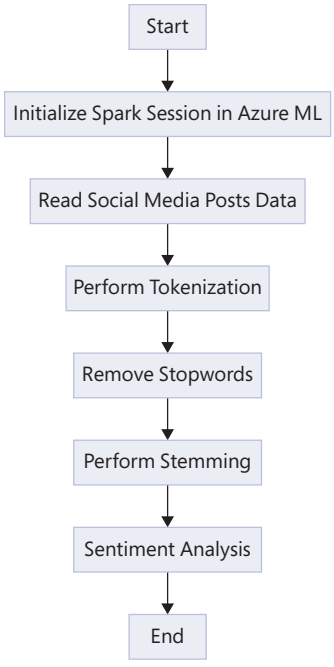

FIGURE 2-1 Workflow for sentiment analysis solution in Apache Spark

To use Apache Spark pools within Azure Machine Learning for wrangling a large dataset of social media posts stored in a datastore, you can follow these steps, which include creating a Spark pool and creating a datastore.

1. Set Up Your Azure Machine Learning Workspace: Make sure you have an Azure Machine Learning workspace set up.
2. Create a Datastore: Link your Azure Blob Storage account (where your social media posts dataset is stored) to your Azure Machine Learning workspace by creating a datastore.
3. Create a Spark Pool: In the Azure portal, navigate to your Azure Synapse Analytics workspace and create a Spark pool. You can check to make sure the Spark pool is correctly configured by verifying the node count, node size, and other settings on the Spark pool configuration page.

If you followed the above instructions to configure your datastore and have created a Spark pool, then you are ready to use the Spark pool to read the large dataset from the datastore, perform text preprocessing tasks like tokenization, stopword removal, and stemming, and prepare the data for sentiment analysis. Before reading the code in Listing 2-2, you should have a conceptual understanding of Spark's execution model to understand how model training can be distributed using a feature like Spark pools. Figure 2-2 illustrates how Spark's execution model is built on the concept of a directed acyclic graph (DAG) internally.

Listing 2-2 demonstrates how to use the Spark pool to read the large dataset from the datastore, perform text preprocessing tasks like tokenization, stopword removal, and stemming, and prepare the data for sentiment analysis.

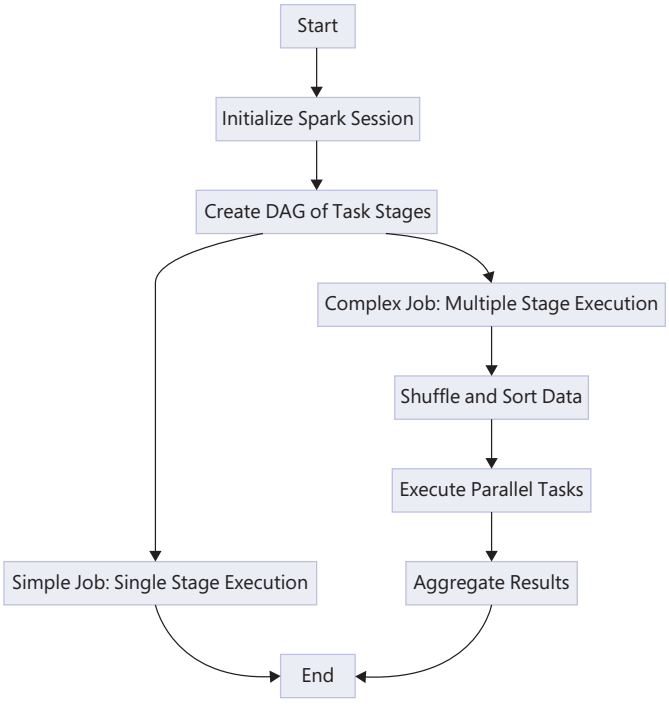

FIGURE 2-2 Workflow using Spark's directed acyclic graph (DAG) for task execution

LISTING 2-2 Spark pool for reading large datasets from the datastore and preprocessing tasks

```
from azureml.core import Workspace, Datastore
from azureml.core.compute import SynapseCompute
from azureml.core.compute_target import ComputeTargetException
from pyspark.sql import SparkSession
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF

# Set up your Azure Machine Learning workspace
subscription_id = 'your-subscription-id'
resource_group = 'your-resource-group'
workspace_name = 'your-workspace-name'
workspace = Workspace(subscription_id, resource_group, workspace_name)

# Create a Spark pool (if not already created)
spark_pool_name = "synapse-spark-pool"
try:
    spark_pool = SynapseCompute(workspace=workspace,
    name=spark_pool_name)
    print('Found existing Spark pool.')
except ComputeTargetException:
    print('Creating a new Spark pool.')
    spark_pool_config = SynapseCompute.
provisioning_configuration(compute_pool_name=spark_pool_name)
    spark_pool = ComputeTarget.create(workspace, spark_pool_name, spark_pool_config)
    spark_pool.wait_for_completion(show_output=True)
```

```
# Create a datastore (if not already created)
datastore_name = 'your-datastore-name'
container_name = 'your-container-name'
account_name = 'your-storage-account-name'
datastore = Datastore.register_azure_blob_container(workspace=workspace,
datastore_name=datastore_name,
container_name=container_name,
account_name=account_name)

# Initialize a Spark session
spark = SparkSession.builder.getOrCreate()

# Access data from the datastore
datastore_path = f"abfss://{container_name}@{account_name}.dfs.core.windows.net/path/to/
your/social_media_posts.csv"
data = spark.read.option("header", "true").csv(datastore_path)

# Preprocess the data
tokenizer = Tokenizer(inputCol="post", outputCol="tokens")
tokenized_data = tokenizer.transform(data)
remover = StopWordsRemover(inputCol="tokens", outputCol="filtered_tokens")
filtered_data = remover.transform(tokenized_data)
hashingTF = HashingTF(inputCol="filtered_tokens", outputCol="raw_features")
featurized_data = hashingTF.transform(filtered_data)
idf = IDF(inputCol="raw_features", outputCol="features")
idf_model = idf.fit(featurized_data)
final_data = idf_model.transform(featurized_data)
```

In this example, we used PySpark's `Tokenizer`, `StopWordsRemover`, `HashingTF`, and `IDF` to preprocess the text data. You can replace these with any other preprocessing steps suitable for your use case. After preprocessing, the `final_data DataFrame` (remember this is a Spark dataframe) will be ready for sentiment analysis or any other machine learning tasks.

Skill 2.2: Create models by using the Azure Machine Learning Designer

The Azure Machine Learning Designer enables you to create models for use in a training pipeline. In order to do this, we need to also be able to consume data assets such as training, validation, and test data in the Designer. These data assets can be used in the training pipeline, with inputs and outputs defined between steps. In this skill, you will develop the techniques and knowledge necessary to start building end-to-end data science solutions in Azure.

This skill covers how to:

- Create a training pipeline
- Consume data assets from the Designer
- Use custom code components in Designer
- Evaluate the model, including responsible AI guidelines

Create a training pipeline

A training pipeline in Azure Machine Learning Designer is a sequence of steps to prepare data, train a model, and evaluate its performance. It provides a visual and modular approach to building machine learning workflows. We will first log in to the Azure portal, create a new Azure workspace, compute resources, and then design a pipeline:

1. Log in to the Azure portal and create a new Azure Machine Learning workspace with the necessary configurations.
2. Create compute resources.
3. Navigate to the Compute page in Azure Machine Learning Studio and set up a compute cluster for training your model.
4. Design Your Pipeline:
    a. Go to the Designer page and create a new pipeline (see Figure 2-3).
    b. Drag and drop modules onto the canvas to define your workflow, including data preprocessing, model training, and evaluation steps.
5. Configure and Run.

Set up the properties for each module, such as selecting the algorithm for the Train Model module and defining the evaluation metrics in the Evaluate Model module.

Submit the pipeline as an experiment and monitor its progress. Once the experiment is complete, examine the output of the Evaluate Model module to assess the performance of your trained model.

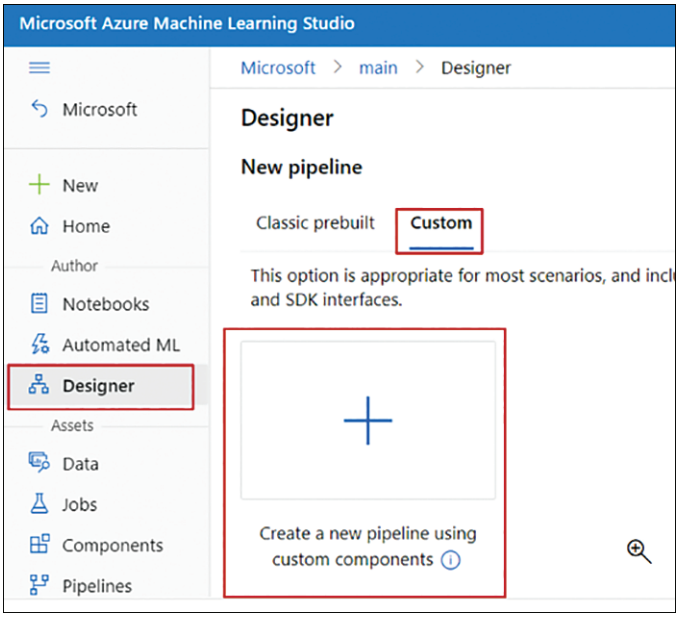

FIGURE 2-3 Creating a new pipeline

Consume data assets from the Designer

Data assets are important components of a training pipeline. They include datasets, data transformations, and data connections that are used throughout the pipeline to train and evaluate the model. You can use the Data page in Azure Machine Learning Studio to create new datasets or import existing ones. Supported data sources include web files, datastores, and local files.

Preprocessing data for model training

In the following steps, we will utilize modules to clean and transform data. Next, we will configure these modules to handle missing values and employ other modules to split data and eventually connect the final preprocessed data to our Train Model module for training. Here is a more detailed set of instructions you can follow on your own:

1. Utilize modules in Azure Machine Learning Studio such as Select Columns in Dataset and Normalize Data to clean and transform your data before training. The modules are in the module panel on the left side of the workspace, organized under category headings.
2. Configure these modules to select relevant features, handle missing values, and scale numerical data.
3. Employ the Split Data module to divide your dataset into training and validation sets.
4. Connect your preprocessed and split datasets to the Train Model module.
5. Ensure that the data flows correctly through the pipeline to provide the model with the necessary input for training.

Data assets form the backbone of a training pipeline in Azure Machine Learning Designer.

Proper management and utilization of these assets, from creation to preprocessing and splitting, are key to building an effective machine learning model.

Use custom code components in Designer

While Azure Machine Learning Designer provides a wide range of built-in modules, you may encounter scenarios where custom processing is required. Custom code components allow you to integrate Python or R scripts into your pipeline to perform specialized tasks.

Incorporating custom code

One way to use custom code in an Azure Machine Learning training pipeline is via a script. You can develop a Python or R script that performs the desired data processing or analysis task. For example, you might write a script to perform a unique data transformation or to generate custom features. More specifically, you can use the Execute Python Script module (see Figure 2-4). In the following steps, you will use an Execute Python Script module to upload your script and configure it. The configuration will involve both input and output ports. You can integrate this

script module with the rest of your pipeline by connecting an output of a previous module with the input of this module, and the output of your script module with subsequent modules.

1. Add the Execute Python Script module to your pipeline in the Designer.
2. Upload your script to the module and configure any necessary input and output ports.
3. Connect the output of a previous module (e.g., data preprocessing) to the input of the Execute Python Script module.
4. Ensure that the output of your custom script is connected to subsequent modules for further processing or model training.

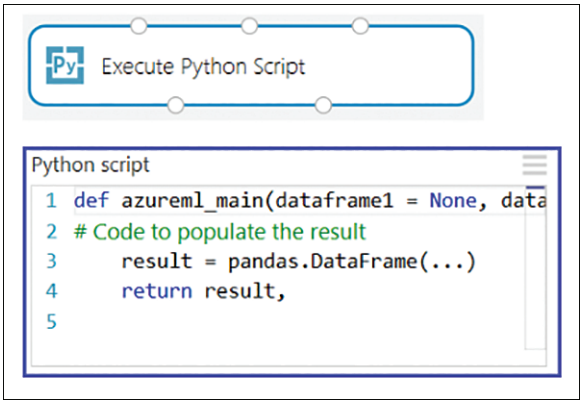

FIGURE 2-4 Using a custom Python script in Azure ML Designer

Notice that steps 3 and 4 are necessary to integrate your Python Script module with the rest of the pipeline, connecting output of the previous modules (which could itself be another data preprocessing step) to the input of your custom Python Script module (see Figure 2-5). Remember to connect the output of your custom module to the input of the next or subsequent module as well. It might seem obvious, but this is a subtle step because you can form DAGs (directed acyclic graphs) by connecting your module as inputs to many subsequent modules. The concept of a DAG is used when building pipelines that can have many parallel steps and can, for example, fan out. In addition to the fan-out pattern, a DAG can be used as a powerful abstraction for building sequential steps—steps that fan in or converge and manage parallelism and dependencies in your pipeline.

Run your pipeline to test the custom code component and, if necessary, iteratively refine it. Make any necessary adjustments to ensure that it performs as expected within the context of your workflow. Figure 2-5 shows how to connect inputs and outputs using a Python Script module.

Carefully configure each module in your pipeline. Double-check the parameters and settings to ensure they are appropriate for your data and the problem you're solving.

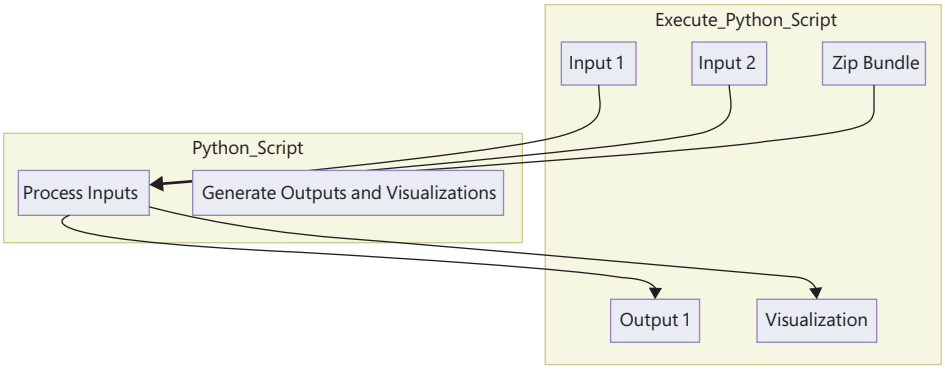

FIGURE 2-5 Connecting inputs and outputs in Python Script modules in the Designer

Verify that all modules are correctly connected in the pipeline. The output of one module should correctly feed into the input of the next. In the next section, we will look at a more robust procedure for evaluating the model and using responsible AI guidelines.

Evaluate the model, including responsible AI guidelines

Model evaluation is a critical step in the training pipeline. It helps you assess the performance of your model and ensure that it aligns with Responsible AI principles. Our first step is to understand how to evaluate the model using evaluation metrics.

Evaluation metrics

There is a module called the Evaluate Model. You can add this module to your pipeline after the training and scoring steps. This module provides various metrics such as accuracy, precision, recall, and F1 score to assess the performance of your classification model.

What does Evaluate Model do, and why should we use it? We can analyze the results of our models by examining the output of the Evaluate Model module to understand how well your model is performing.

IMPORTANT **MODEL METRICS AND THEIR USES IN PIPELINES**

Remember to pay attention to metrics that are particularly important for your specific use case and objectives.

Responsible AI considerations

Microsoft developed a standard called the Responsible AI Standard. This is a framework for building AI systems according to six principles:

- Model fairness
- Reliability and safety

- Privacy and security
- Inclusiveness
- Transparency
- Accountability

EXAM TIP

Understand each of the six responsible AI guidelines for the exam. The six responsible AI guidelines are not just useful for your exam but are an important concept for ensuring that your solution meets ethical standards and complies with privacy, security, and safety guidelines.

We can look at some of these principles in more detail and how we can practice the guidelines when designing our data science solutions in Azure by incorporating specific modules into our pipeline. Figure 2-6 illustrates the relationship between the responsible AI guidelines.

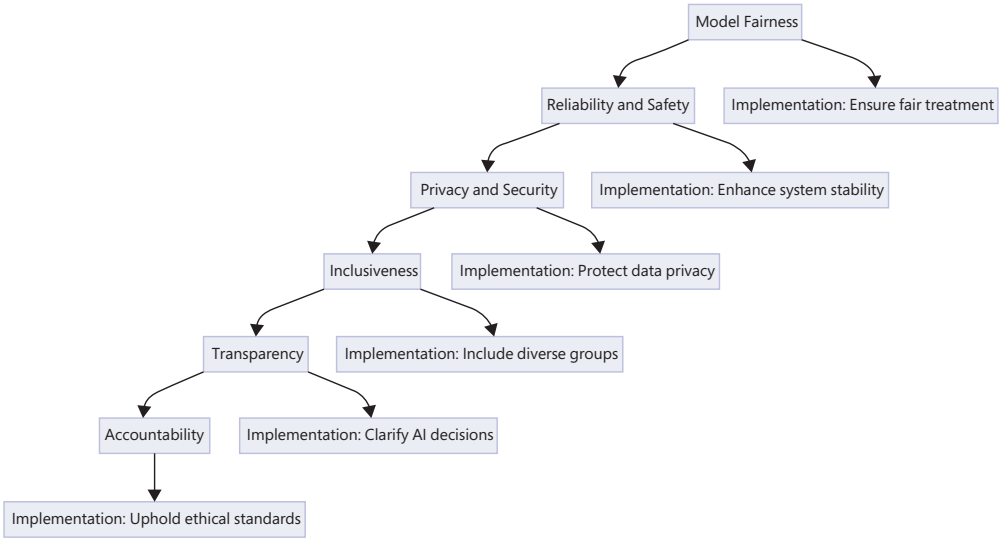

FIGURE 2-6 Pillars of the Responsible AI Guidelines from Microsoft

Fairness

When evaluating your model's predictions, it's important to integrate the fairness guideline to assess fairness across different demographic groups. This module helps detect any disparities and allows you to address them effectively, ensuring that your model maintains equity and avoids perpetuating biases. Considering demographic factors such as race or gender in

prediction assessment provides valuable insights into potential biases, enabling proactive steps to mitigate them. This approach promotes inclusivity and fairness, which are fundamental principles in AI development and deployment.

Analyzing predictions through a demographic lens offers a deeper understanding of your model's performance. It helps uncover and rectify any underlying biases within your data or algorithm, ultimately leading to more just and reliable outcomes. Incorporating fairness into your evaluation process allows you to enhance the credibility and reliability of your model while promoting social responsibility in AI development. This approach increases the trust in your model's outputs but also contributes to creating a more equitable landscape in the applications where it's utilized.

Explainability

When exploring your model's predictions, using the Model Interpretability module provides insights into how the model makes decisions. This tool helps you understand the factors influencing these decisions, fostering transparency in the process. Transparency is key for stakeholders to grasp how the model reaches its conclusions, building trust and facilitating informed decision-making.

However, it's important to distinguish between model interpretability and fairness assessment. While interpretability focuses on understanding the model's decision-making process, fairness evaluation examines whether these predictions exhibit biases across different demographic groups. Both are vital for model evaluation, serving distinct purposes. Interpretability aids in comprehending how the model functions internally, while fairness assessment ensures equitable outcomes for all demographic groups. Thus, integrating both modules into your evaluation process offers a holistic view of your model's performance and its impact on diverse populations.

Privacy and security

Ensure that your model adheres to privacy and security guidelines, particularly when handling sensitive data. Implement appropriate measures to protect data confidentiality and integrity.

Incorporating custom code components in your training pipeline allows you to extend the functionality of Azure Machine Learning Designer with specialized processing tasks.

Evaluating your model with a focus on performance metrics and Responsible AI principles ensures that your model is not only accurate but also fair, transparent, and secure. Figure 2-7 illustrates the process of making decisions on model fairness.

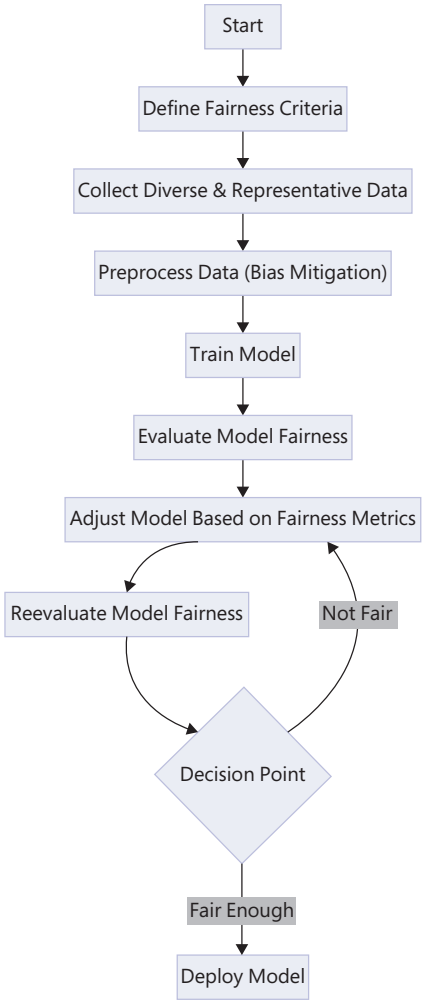

FIGURE 2-7 Ensuring model fairness in AI training

Skill 2.3: Use automated machine learning to explore optimal models

Azure Machine Learning service's automated ML capability is based on a breakthrough from the Microsoft Research division. It is distinct from competing solutions in the market. The approach combines ideas from collaborative filtering and Bayesian optimization. This combination allows it to search an enormous space of possible machine learning pipelines intelligently and efficiently. Essentially, it acts as a recommender system for machine learning pipelines. Just as streaming services recommend movies for users, automated ML recommends machine learning pipelines for datasets.

This skill covers how to:

- Use automated machine learning for tabular data
- Select and understand training options, including preprocessing and algorithms
- Evaluate an automated machine learning run, including responsible AI guidelines
- Use automated machine learning for computer vision
- Use automated machine learning for natural language processing (NLP)

Use automated machine learning for tabular data

Imagine you're a data scientist working for a telecom company. Your task is to develop a machine learning model to predict customer churn based on various customer attributes. You decide to use Azure Machine Learning's Automated Machine Learning (AutoML) feature to quickly build and deploy the model. We'll break this down into setting up your environment, preparing your tabular data, and then using AutoML on your tabular data by looking at a real-world scenario.

Working with tabular data in Azure Machine Learning

This section covers how you can use objects like MLTable for data processing. The MLTable object can be used with your tabular data (for example, a CSV file containing customer churn data). MLTable is a feature of Azure Machine Learning that allows you to define and save a series of data loading steps for tabular data. This makes it easier to reproduce data loading in different environments and share it with team members. MLTable supports various data sources, including CSV and Parquet files. Figure 2-8 shows selecting AutoML in the Designer.

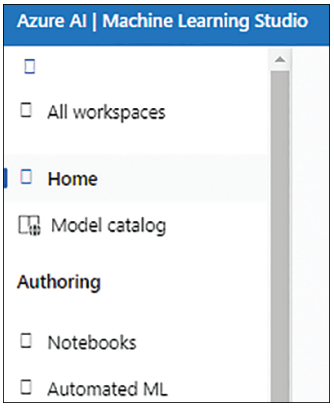

FIGURE 2-8 AutoML in the workspace

Here's how you can use MLTable with AutoML for tabular data:

1. Define Data Loading Steps: Use the mltable Python SDK (to clarify, mltable must be used via Python and not the UI) to define the steps for loading and preprocessing your data. This includes specifying the data source, filtering rows, selecting columns, and creating new columns based on the data.
2. Save Data Loading Steps: Once you have defined the data loading steps, you can save them into an MLTable file. This file contains the serialized steps, making it easy to reproduce the data loading process.
3. Load Data into a Pandas DataFrame: You can load the data defined by an MLTable into a Pandas DataFrame. This is useful for exploring the data and performing additional preprocessing before training a model.
4. Use MLTable with AutoML: When setting up an AutoML experiment for tabular data, you can use an MLTable as the data input. AutoML will automatically apply the data loading steps defined in the MLTable and use the resulting DataFrame for model training.
5. Create a Data Asset: To share the MLTable with team members and ensure reproducibility, you can create a data asset in Azure Machine Learning. This stores the MLTable in cloud storage and makes it accessible through a friendly name and version number.
6. Use Data Asset in Jobs: You can reference the data asset in Azure Machine Learning jobs, such as training or inference jobs. This allows you to use the same data loading steps consistently across different experiments and pipelines.

Here's an example of how to turn a CSV file into an MLTable using the SDK:

```
import mltable
# Define the data source (CSV file)
paths = [{'file': 'path/to/your/data.csv'}]
# Create an MLTable from the CSV file
tbl = mltable.from_delimited_files(paths)
# Apply any additional data loading steps (e.g., filtering, column selection)
tbl = tbl.filter("col('some_column') > 0")
tbl = tbl.select_columns(["column1", "column2"])
# Save the data loading steps into an MLTable file
tbl.save("./your_mltable_directory")
```

In this example, the CSV file is turned into an MLTable with some filtering and column selection steps. The resulting MLTable can then be used with AutoML for training machine learning models on tabular data.

Now that we understand how to work with tabular data in a pipeline, we can look at some specific scenarios for using AutoML on tabular data for a customer churn prediction pipeline. You can also use the Designer with AutoML and tabular data. Figure 2-9 shows creating a new AutoML run in the Designer.

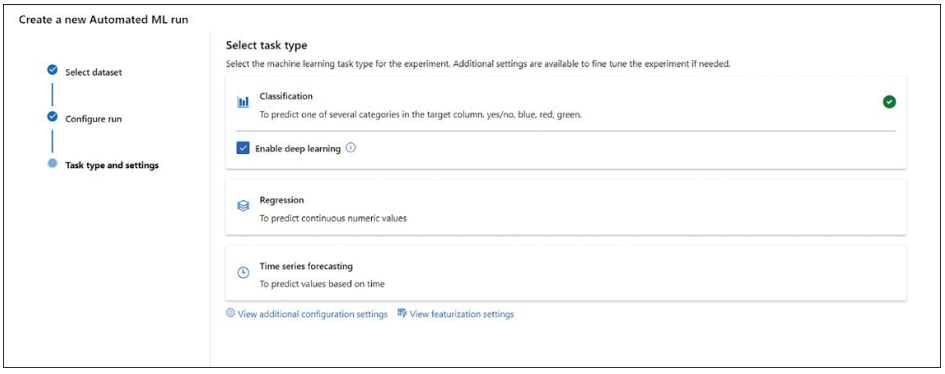

FIGURE 2-9 Advanced Features of Automated Machine Learning for model development

Select and understand training options, including preprocessing and algorithms

Automated Machine Learning (AutoML) in Azure Machine Learning is a powerful tool that automates the process of selecting the best machine learning algorithms and hyperparameters for your data. This simplifies the machine learning workflow, making it accessible to data scientists, analysts, and developers, regardless of their expertise in machine learning. In the following section, we will look at automating machine learning concepts including training data, validation, featurization, preprocessing, distributed training, model selection, and ensemble learning in the context of Azure's AutoML capabilities.

- **Automated Machine Learning** AutoML in Azure provides various training options to cater to different requirements and preferences. These options are designed to optimize the model development process, ensuring efficiency and effectiveness in training machine learning models.
- **Training Data and Validation** AutoML allows users to provide training data in different formats, including MLTable for tabular data. Users can specify separate datasets for training and validation or let AutoML automatically split the training data for validation purposes. This helps in evaluating the model's performance and avoiding overfitting. For time-series forecasting, AutoML supports advanced configurations like rolling-origin cross-validation to ensure robust model evaluation.
- **Featurization and Preprocessing** AutoML automates the featurization and preprocessing steps, which are crucial for preparing the data for model training. This includes handling missing values, encoding categorical variables, and scaling numerical features. Users can customize these steps by specifying featurization settings, such as blocking certain transformers or defining custom transformations. This flexibility allows users to tailor the data preprocessing to their specific needs, ensuring that the input data is in the optimal format for training.

- **Distributed Training** For large datasets and complex models, AutoML supports distributed training. This allows the training process to be distributed across multiple compute nodes, significantly reducing the training time. Users can specify the number of nodes to use for training, enabling parallel execution of model training. Distributed training is particularly beneficial for tasks like deep learning and NLP, where the computational requirements are high.
- **Model Selection and Hyperparameter Tuning** AutoML automates the selection of machine learning algorithms and the tuning of hyperparameters. It iterates through a predefined list of algorithms and tests different hyperparameter combinations to find the best-performing model. Users can control the number of iterations and set limits on the training time to manage computational resources effectively.
- **Ensemble Models** AutoML supports ensemble models, which combine predictions from multiple models to improve accuracy. It uses techniques like voting and stacking to create ensembles, automatically selecting the best models to include in the ensemble based on their performance.

Table 2-1 outlines the algorithms that are supported by Automated Machine Learning (AutoML) in Azure Machine Learning for various learning tasks.

TABLE 2-1 Automated Machine Learning algorithms

Task Type	Algorithms
Classification	- Logistic Regression* - Light GBM* - Gradient Boosting* - Decision Tree* - K Nearest Neighbors* - Linear SVC* - Support Vector Classification (SVC) - Random Forest - Extremely Randomized Trees* - Xgboost* - Naive Bayes* - Stochastic Gradient Descent (SGD)*
Regression	- Elastic Net* - Light GBM* - Gradient Boosting* - Decision Tree* - K Nearest Neighbors* - LARS Lasso* - Stochastic Gradient Descent (SGD) - Random Forest - Extremely Randomized Trees - Xgboost* - Xgboost
Time Series Forecasting	- AutoARIMA - Prophet - Elastic Net - Light GBM - K Nearest Neighbors - Decision Tree - LARS Lasso - Extremely Randomized Trees* - Random Forest - TCNForecaster - Gradient Boosting - ExponentialSmoothing - SeasonalNaive - Average - Naive - SeasonalAverage
Image Classification	- MobileNet - ResNet - ResNeSt - SE-ResNeXt50 - ViT
Image Classification Multi-label	Refer to ClassificationMultilabelPrimaryMetrics Enum
Image Object Detection	- YOLOv5 - Faster RCNN ResNet FPN - RetinaNet ResNet FPN
NLP Text Classification Multi-label	Refer to supported algorithms for NLP tasks
NLP Text Named Entity Recognition (NER)	Refer to supported algorithms for NLP tasks

Algorithms marked with an asterisk (*) are default models.

For NLP tasks, AutoML supports a range of pretrained text DNN models, including but not limited to BERT, GPT-4, RoBERTa, T5, and LaMDA.

> ***NEED MORE REVIEW?*** **OFFICIAL ALGORITHM LIST**
>
> **If you'd like to read further about what algorithms are supported by AutoML, the list is maintained at *https://learn.microsoft.com/en-us/azure/machine-learning/how-to-configure-auto-train?view=azureml-api-2&tabs=python#supported-algorithms***

Before showing an example of how you can select and use various training options in Automated Machine Learning (AutoML) with the Azure Machine Learning Python SDK v2, we need to list some of the options that are available:

- **Primary Metric** This is the metric that AutoML will optimize for model selection. Common metrics include accuracy for classification tasks and mean_squared_error for regression tasks.
- **Validation Strategy** AutoML supports several validation strategies such as cross-validation and train-validation splits. This helps in evaluating the model's performance on unseen data.
- **Max Trials** This specifies the maximum number of different algorithm and parameter combinations that AutoML will try before selecting the best model.
- **Max Concurrent Trials** This is the maximum number of trials that can run in parallel, which can speed up the training process.
- **Timeout** You can set a maximum amount of time for the AutoML experiment. Once the time limit is reached, AutoML will stop trying new models.
- **Featurization** AutoML can automatically preprocess and featurize the input data, which includes handling missing values, encoding categorical variables, and more.

The following code example shows how to configure these training options in AutoML using the Azure Machine Learning Python SDK:

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

# Set up the MLClient
credential = DefaultAzureCredential()
subscription_id = "your-subscription-id"
resource_group = "your-resource-group"
workspace_name = "your-workspace-name"
ml_client = MLClient(credential, subscription_id, resource_group, workspace_name)

# Define the training data
training_data_input = Input(type=AssetTypes.MLTABLE, path="./data/training_data/")

# Configure the AutoML job
automl_job = automl.classification(
compute="your-compute-cluster",
experiment_name="automl_classification_example",
```

```
training_data=training_data_input,
target_column_name="target",
primary_metric="accuracy",
validation_data_split=0.2,
max_trials=100,
max_concurrent_trials=4,
timeout_minutes=60,
enable_model_explainability=True
)

# Submit the AutoML job
submitted_job = ml_client.jobs.create_or_update(automl_job)
print(f"Submitted job: {submitted_job}")

# Get the URL to monitor the job
print(f"Monitor your job at: {submitted_job.services['Studio'].endpoint}")
```

In this example, we've configured the primary metric as accuracy, set a validation data split of 20%, limited the maximum number of trials to 100, allowed up to 4 trials to run concurrently, and set a timeout of 60 minutes. We've also enabled model explainability to interpret the model's predictions.

You can adjust these options based on your specific requirements and the nature of your dataset. Whether you're a seasoned data scientist or a developer new to machine learning, AutoML provides the tools you need to develop and deploy machine learning models with ease. In the next section, we will look at the last piece of the above example: evaluating an Automated Machine Learning Run according to responsible AI guidelines.

Evaluate an automated machine learning run, including responsible AI guidelines

Depending on the type of machine learning task (classification, regression, etc.), different metrics are used to evaluate the model's performance.

Classification metrics

Classification metrics include accuracy, precision, and recall having specific meaning as ratios of true and false positives to actual positive predictions as well as metrics like F1 Score and AUC-ROC, or area under the receiver-operating curve. Monitoring the performance of your classification models using accuracy, F1 Score, or AUC-ROC to detect model drift and to decide when to retrain the model are concepts we will explore in later chapters, so it is important to understand the definitions for the following classification metrics:

- **Accuracy** Proportion of correct predictions
- **Precision** Ratio of true positives to all positive predictions
- **Recall** Ratio of true positives to all actual positives
- **F1 Score** Harmonic mean of precision and recall
- **AUC-ROC** Area under the Receiver Operating Characteristic curve

Regression Metrics

Not all supervised machine learning problems are classification problems. Regression problems could involve predicting a continuous response variable—for example, forecasting demand for a new product line requires its own set of performance metrics to measure the error between predicted and actual values. Here are a few important regression metrics that you could encounter frequently in the real world as well as on exam questions:

- **Mean Absolute Error (MAE)** Average of absolute differences between predicted and actual values
- **Mean Squared Error (MSE)** Average of squared differences between predicted and actual values
- **Root Mean Squared Error (RMSE)** Square root of MSE
- **R-squared** Proportion of variance in the dependent variable that is predictable from the independent variables

Using evaluation metrics in AutoML

When you run an AutoML experiment, it automatically calculates and logs these metrics for each model. You can access these metrics through the Azure Machine Learning Studio or programmatically using the SDK.

Visualizations for model evaluation

AutoML provides various visualizations to help you understand the model's performance:

- **Confusion Matrix** For classification tasks, this shows the number of correct and incorrect predictions for each class.
- **ROC Curve** For binary classification, this plots the true positive rate against the false positive rate at various threshold levels.
- **Precision-Recall Curve** For binary classification, this shows the trade-off between precision and recall for different threshold levels.
- **Residuals Plot** For regression tasks, this shows the difference between actual and predicted values.

After the AutoML run is complete, you can retrieve the best model based on the primary metric you specified. You can then evaluate this model on a test dataset to get a sense of its real-world performance.

Here's an example of how you can retrieve and evaluate the best model from an AutoML run:

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

# Set up the MLClient
credential = DefaultAzureCredential()
```

```
subscription_id = "your-subscription-id"
resource_group = "your-resource-group"
workspace_name = "your-workspace-name"
ml_client = MLClient(credential, subscription_id, resource_group, workspace_name)

# Get the best model from the AutoML run
best_model = ml_client.jobs.get_best_model(
experiment_name="automl_classification_example",
job_name="automl_job_name"
)

# Evaluate the best model on a test dataset
test_data = Input(type=AssetTypes.MLTABLE, path="./data/test_data/")
evaluation_results = ml_client.jobs.evaluate(
model=best_model,
test_data=test_data
)
```

In this example, we retrieve the best model from a completed AutoML run and evaluate it on a separate test dataset. The evaluation results provide metrics that help us understand the model's performance.

NEED MORE REVIEW? **AUTOMATING AND EVALUATING AUTOMATED MACHINE LEARNING RUNS**

You can read further about automating and evaluating machine learning runs at *https://learn.microsoft.com/en-us/azure/machine-learning/how-to-understand-automated-ml*

Predicting customer churn with Azure AutoML

Suppose you are a data scientist tasked with creating a machine learning model to predict customer churn for a telecom company. To accomplish this, you decide to leverage Azure's Automated Machine Learning (AutoML) feature, which simplifies the process of building and deploying models. Here's a step-by-step guide to help you prepare tabular data for use with Automated Machine Learning capabilities (see Figure 2-10 for an example using the Designer):

1. Set Up Your Environment: Create an Azure Machine Learning workspace. This is your centralized environment for managing and monitoring your machine learning models.
2. Install the Azure Machine Learning SDK v2 for Python: Run pip install azure-ai-ml in your terminal. This SDK enables you to interact with Azure Machine Learning services and resources programmatically.
3. Prepare Your Tabular Data: Gather your dataset. Ensure that your dataset includes various customer attributes and a churn label indicating whether the customer has churned.
4. Format Your Data: Structure your data in a tabular format with rows representing individual customers and columns representing attributes. The target column should be the churn label.
5. Upload Your Dataset to Azure: Convert your dataset to an MLTable and upload it to Azure. MLTable is a tabular data format supported by Azure AutoML.

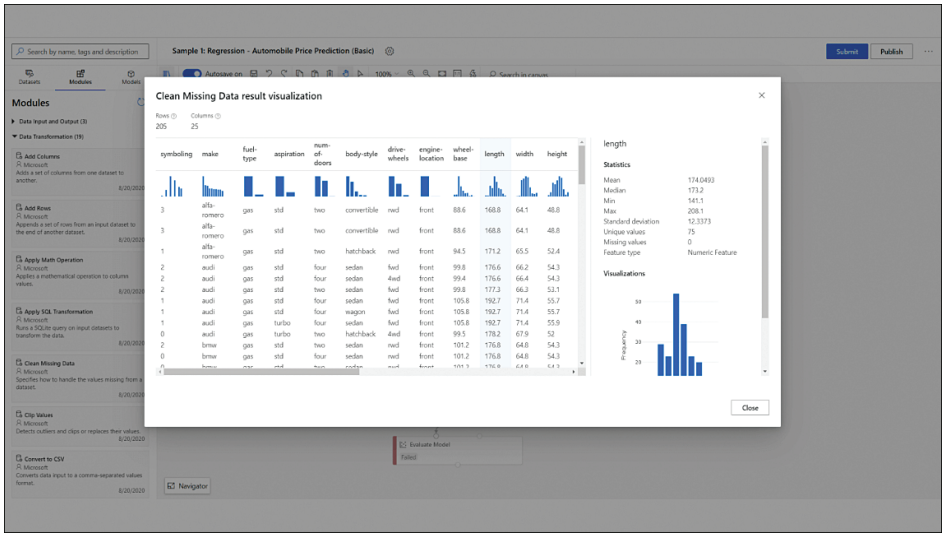

FIGURE 2-10 Data connection and feature preparation in Azure Machine Learning

Specify the task type as classification since you're predicting a binary outcome (churn or no churn). Choose accuracy as your primary metric to evaluate model performance. Also, decide on your data splitting strategy (e.g., cross-validation or train-validation split). Determine the maximum duration for the experiment (timeout minutes) and the maximum number of trials (max trials). This helps in managing computational resources and experiment time.

Run your AutoML experiment

The following code shows how to use the Azure Machine Learning SDK to submit your AutoML experiment for execution. The purpose of the code is to show in detail how to use AutoML, including configuring limits like time outs and max trials. Keep an eye on the experiment's progress through the Azure Machine Learning Studio or SDK. You can review the performance of different models as they are generated.

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

# Set up workspace
credential = DefaultAzureCredential()
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<WORKSPACE_NAME>"
ml_client = MLClient(credential, subscription_id, resource_group, workspace)

# Prepare data
train_data_input = Input(type=AssetTypes.MLTABLE, path="./data/customer_churn_data")

# Configure AutoML experiment
```

```
classification_job = automl.classification(
compute="<COMPUTE_NAME>",
experiment_name="customer_churn_prediction",
training_data=train_data_input,
 target_column_name="Churn",
primary_metric="accuracy",
n_cross_validations=5
)

# Set limits (optional)
classification_job.set_limits(
 timeout_minutes=60,
max_trials=20
)

# Run the experiment
returned_job = ml_client.jobs.create_or_update(classification_job)
print(f"Created job: {returned_job}")
```

Use automated machine learning for computer vision

Imagine you are a data scientist tasked with developing a model to classify animal images. Your goal is to utilize Azure Automated Machine Learning (AutoML) for computer vision tasks to accomplish this.

Setting up the environment

To kickstart your machine learning journey, the first step is to establish an Azure Machine Learning workspace, acting as a centralized hub for overseeing and tracking your machine learning models' progress. This workspace provides a unified platform for managing resources, conducting experiments, and deploying models seamlessly. Following this, installing the Azure Machine Learning CLI v2 and Python SDK v2 equips you with the necessary tools to interact with Azure services efficiently. These resources empower you to leverage Azure's capabilities effectively, enabling streamlined development, deployment, and management of machine learning solutions within your workspace.

Selecting the task type

In this project, the task type selected is image classification, which serves as a cornerstone determining the approach and algorithms utilized by AutoML for model training. Image classification involves categorizing images into predefined classes or categories based on their visual features. This choice significantly influences the techniques employed during the training phase, as well as the algorithms leveraged to optimize model performance.

Image classification tasks typically require specialized algorithms capable of understanding and extracting meaningful features from images to accurately classify them. AutoML, being an automated machine learning platform, adapts its approach based on the specified task type. For image classification, it employs algorithms specifically designed to process image data efficiently, such as convolutional neural networks (CNNs). CNNs are particularly well-suited for

image-related tasks due to their ability to automatically learn hierarchical representations of visual features from the input images.

Furthermore, the choice of image classification as the task type underscores the importance of selecting appropriate evaluation metrics and validation strategies tailored to this specific problem domain. Metrics such as accuracy, precision, recall, and F1-score are commonly used to assess the performance of image classification models. Additionally, techniques like cross-validation or stratified sampling may be employed to ensure robust evaluation and prevent overfitting. Therefore, the decision to focus on image classification guides the entire workflow of model training within the AutoML framework, shaping the selection of algorithms, evaluation metrics, and validation strategies to achieve optimal results.

Preparing the data

Your next step is to organize your labeled image data. Format this data into JSONL format, ensuring that each line contains an image URL and the corresponding label. If your data is in a different format, such as Pascal VOC or COCO, convert it to JSONL using available helper scripts. A minimum of 10 images is recommended to start the training process. Here is an example of JSONL format to help visualize what this looks like for an image URL and a label that can have values "cat", "dog", "bird", "car", and "tree":

```
{"image_url": "http://example.com/image1.jpg", "label": "cat"}
{"image_url": "http://example.com/image2.jpg", "label": "dog"}
{"image_url": "http://example.com/image3.jpg", "label": "bird"}
{"image_url": "http://example.com/image4.jpg", "label": "car"}
{"image_url": "http://example.com/image5.jpg", "label": "tree"}
```

Create an MLTable for your training and validation data using Azure CLI or Python SDK. This involves specifying the path to your JSONL files and defining any necessary data transformations. MLTable serves as a structured representation of your data for AutoML.

Setting up compute for training

Choose a GPU-enabled compute target, such as the NC or ND series VMs, to train your computer vision models. The choice of compute target affects the speed and efficiency of model training.

Configure your AutoML experiment by setting parameters like the task type, primary metric, and job limits (e.g., `timeout_minutes`, `max_trials`, and `max_concurrent_trials`). This step involves defining the boundaries and objectives of the model training process. Figure 2-11 shows the menu for submitting an Automated ML Job including basic settings and Task settings like task type mentioned previously.

Evaluating and deploying the model

After training, evaluate the best model based on the primary metric in accordance with the responsible AI guidelines covered earlier. Register this model in your Azure Machine Learning workspace and deploy it as a web service for making predictions. This final step makes your model accessible for real-world applications. Figure 2-12 shows selecting computer vision task-specific options in AutoML and the different options available as well as where to select data for training.

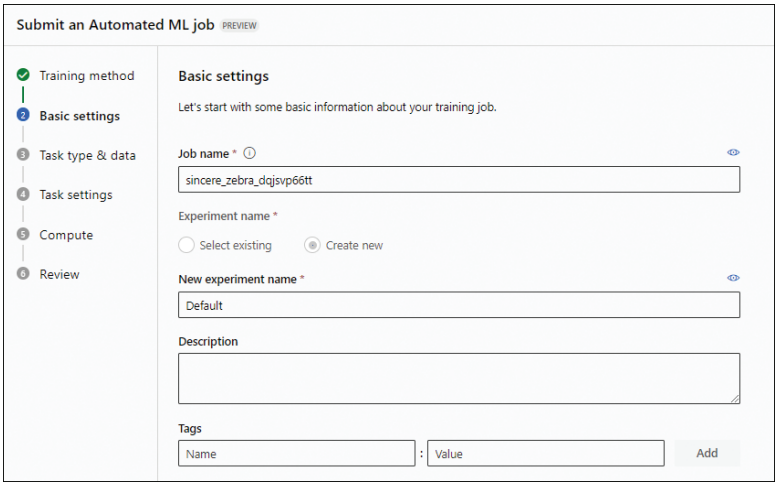

FIGURE 2-11 Submitting an AutoML job in Azure Machine Learning

NEED MORE REVIEW? **COMPUTER VISION USING AUTOML**

If you'd like to read further about compute vision using AutoML, the documentation is maintained at *https://learn.microsoft.com/en-us/azure/machine-learning/how-to-auto-train-image-models*

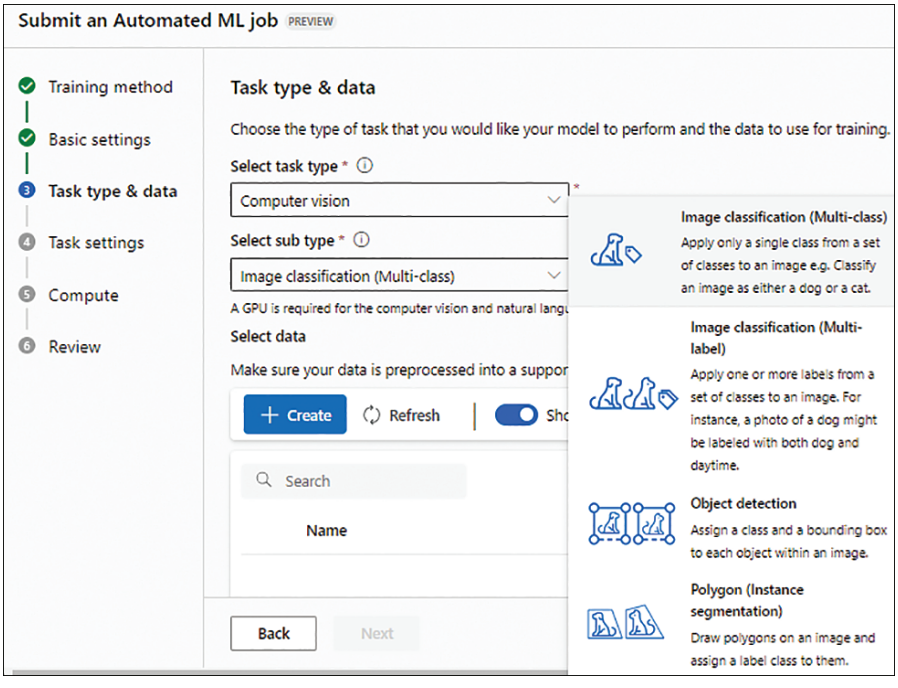

FIGURE 2-12 Select a computer vision task type using AutoML

Use automated machine learning for natural language processing (NLP)

Imagine again that you are a data scientist aiming to develop a natural language processing (NLP) model for classifying movie reviews into genres. You plan to use Azure Automated Machine Learning (AutoML) for NLP tasks. Figure 2-13 shows the high-level architecture for configuring AutoML to perform NLP tasks in Azure Machine Learning; however, in this chapter, we'll concentrate specifically on Automated Machine Learning for NLP tasks.

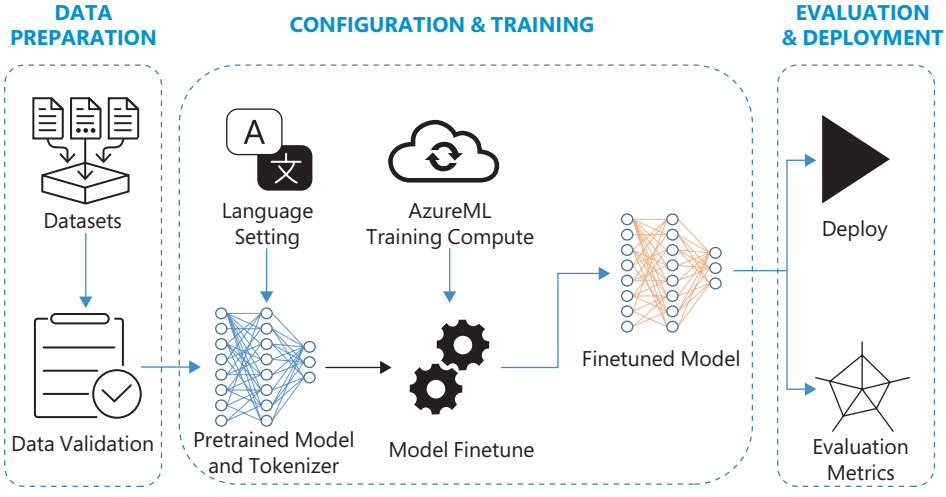

FIGURE 2-13 NLP using AutoML

Setting up the environment

The first step is to create an Azure Machine Learning workspace, which acts as a centralized platform for managing and overseeing NLP models. Additionally, configuring a GPU training compute within the workspace enhances the efficiency of training large-scale NLP models by harnessing the parallel processing power of GPUs. Moreover, installing the Azure Machine Learning CLI v2 and Python SDK v2 equips you with essential tools to seamlessly interact with Azure services. This facilitates smooth integration of NLP pipelines, experimentation, and deployment processes within your workspace. Collectively, these resources empower you to leverage Azure's capabilities effectively for developing, fine-tuning, and deploying NLP solutions with optimal performance and scalability.

Selecting the NLP task

For this project, choose `text_classification` as your NLP task. This task involves classifying each movie review into a specific genre. Organize your dataset in a CSV format with columns for the review text and the corresponding genre labels. Ensure that the data is labeled correctly for the classification task. Figure 2-14 shows how to configure an AutoML experiment for an NLP task.

This page intentionally left blank

Index

A

B

C

D

E

F

G

H

I

J

K-L

M

N

O

P

Q-R

S

T

U-V

W

X-Y-Z