Work with Data

Data is the foundation on which machine learning models are built. Managing data centrally in the cloud, and making it accessible to teams of data scientists who are running experiments and training models on multiple workstations and compute targets is an important part of any professional data science solution.

In this notebook, you'll explore two Azure Machine Learning objects for working with data: *datastores*, and *data assets*.

Connection Information

Actual files/storage services

Before you start

You'll need the latest version of the **azureml-ai-ml** package to run the code in this notebook. Run the cell below to verify that it is installed.

> **Note**: If the **azure-ai-ml** package is not installed, run `pip install azure-ai-ml` to install it.

```
pip show azure-ai-ml
```

Connect to your workspace

With the required SDK packages installed, now you're ready to connect to your workspace.

To connect to a workspace, we need identifier parameters - a subscription ID, resource group name, and workspace name. Since you're working with a compute instance, managed by Azure Machine Learning, you can use the default values to connect to the workspace.

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

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

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

List the datastores

When you create the Azure Machine Learning workspace, an Azure Storage Account is created too. The Storage Account includes Blob and file storage and are automatically

connected with your workspace as **datastores**. You can list all datastores connected to your workspace:

```
stores = ml_client.datastores.list()
for ds_name in stores:
    print(ds_name.name)
```

Note the `workspaceblobstore` which connects to the **azureml-blobstore-...** container you explored earlier. The `workspacefilestore` connects to the **code-...** file share.

Create a datastore

Whenever you want to connect another Azure storage service with the Azure Machine Learning workspace, you can create a datastore. Note that creating a datastore, creates the connection between your workspace and the storage, it doesn't create the storage service itself.

To create a datastore and connect to a (already existing) storage, you'll need to specify:

- The class to indicate with what type of storage service you want to connect. The example below connects to a Blob storage (`AzureBlobDatastore`).
- `name`: The display name of the datastore in the Azure Machine Learning workspace.
- `description`: Optional description to provide more information about the datastore.
- `account_name`: The name of the Azure Storage Account.
- `container_name`: The name of the container to store blobs in the Azure Storage Account.
- `credentials`: Provide the method of authentication and the credentials to authenticate. The example below uses an account key.

Important:

- Replace the **YOUR-STORAGE-ACCOUNT-NAME** with the name of the Storage Account that was automatically created for you.
- Replace the **XXXX-XXXX** for `account_key` with the account key of your Azure Storage Account.

Remember you can retrieve the account key by navigating to the Azure portal, go to your Storage Account, from the **Access keys** tab, copy the **Key** value for key1 or key2.

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

store = AzureBlobDatastore( To create a datastore and connect to an already existing storage service.
    name="blob_training_data",
    description="Blob Storage for training data",
    account_name="YOUR-STORAGE-ACCOUNT-NAME",
    container_name="training-data",
    credentials=AccountKeyConfiguration(
```

```
        account_key="XXXX-XXXX"
    ),
)

ml_client.create_or_update(store)
```

List the datastores again to verify that a new datastore named `blob_training_data` has been created:

```
stores = ml_client.datastores.list()
for ds_name in stores:
    print(ds_name.name)
```

Create data assets

from azure.ai.ml.constants import AssetTypes
AssetTypes.URI_FILE, AssetTypes.URI_FOLDER, AssetTypes.MLTABLE

To point to a specific folder or file in a datastore, you can create data assets. There are three types of data assets:

- `URI_FILE` points to a specific file. (Need to specify path, that can be local/cloud.)
- `URI_FOLDER` points to a specific folder. (local/cloud path, folder doesn't have to exist)
- `MLTABLE` points to a MLTable file which specifies how to read one or more files within a folder.
  - (local/cloud path, containing MLtable and csv file)
  - need to install mltable library to read the data

Read like any other files. e.g, pd.read_csv (...)

You'll create all three types of data assets to experience the differences between them.

To create a `URI_FILE` data asset, you have to specify a path that points to a specific file. The path can be a local path or cloud path.

In the example below, you'll create a data asset by referencing a *local* path. To ensure the data is always available when working with the Azure Machine Learning workspace, local files will automatically be uploaded to the default datastore. In this case, the `diabetes.csv` file will be uploaded to **LocalUpload** folder in the **workspaceblobstore** datastore.

To create a data asset from a local file, run the following cell:

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

my_path = './data/diabetes.csv'

my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FILE,
    description="Data asset pointing to a local file, automatically
uploaded to the default datastore",
    name="diabetes-local"
)

ml_client.data.create_or_update(my_data)
```

To create a `URI_FOLDER` data asset, you have to specify a path that points to a specific folder. The path can be a local path or cloud path.

In the example below, you'll create a data asset by referencing a *cloud* path. The path doesn't have to exist yet. The folder will be created when data is uploaded to the path.

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

datastore_path = 'azureml://datastores/blob_training_data/paths/data-
asset-path/'

my_data = Data(
    path=datastore_path,
    type=AssetTypes.URI_FOLDER,
    description="Data asset pointing to data-asset-path folder in
datastore",
    name="diabetes-datastore-path"
)

ml_client.data.create_or_update(my_data)
```

To create a `MLTable` data asset, you have to specify a path that points to a folder which contains a MLTable file. The path can be a local path or cloud path.

In the example below, you'll create a data asset by referencing a *local* path which contains an MLTable and CSV file.

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

local_path = 'data/'

my_data = Data(
    path=local_path,
    type=AssetTypes.MLTABLE,
    description="MLTable pointing to diabetes.csv in data folder",
    name="diabetes-table"
)

ml_client.data.create_or_update(my_data)
```

To verify that the new data assets have been created, you can list all data assets in the workspace again:

```
datasets = ml_client.data.list()
for ds_name in datasets:
    print(ds_name.name)
```

Read data in notebook

Initially, you may want to work with data assets in notebooks, to explore the data and experiment with machine learning models. Any `URI_FILE` or `URI_FOLDER` type data assets are read as you would normally read data. For example, to read a CSV file a data asset points to, you can use the pandas function `read_csv()`.

A `MLTable` type data asset is already *read* by the **MLTable** file, which specifies the schema and how to interpret the data. Since the data is already *read*, you can easily convert a MLTable data asset to a pandas dataframe.

You'll need to install the `mltable` library (which you did in the terminal). Then, you can convert the data asset to a dataframe and visualize the data.

```
import mltable

registered_data_asset = ml_client.data.get(name='diabetes-table',
version=1)
tbl = mltable.load(f"azureml:/{registered_data_asset.id}")
df = tbl.to_pandas_dataframe()
df.head(5)
```

Use data in a job

After using a notebook for experimentation. You can use scripts to train machine learning models. A script can be run as a job, and for each job you can specify inputs and outputs.

You can use either **data assets** or **datastore paths** as inputs or outputs of a job.

The cells below creates the **move-data.py** script in the **src** folder. The script reads the input data with the `read_csv()` function. The script then stores the data as a CSV file in the output path.

```
import os

# create a folder for the script files
script_folder = 'src'
os.makedirs(script_folder, exist_ok=True)
print(script_folder, 'folder created')

%%writefile $script_folder/move-data.py
# import libraries
import argparse
import pandas as pd
import numpy as np
from pathlib import Path

def main(args):
    # read data
    df = get_data(args.input_data)
```

```
    output_df = df.to_csv((Path(args.output_datastore) /
"diabetes.csv"), index = False)

# function that reads the data
def get_data(path):
    df = pd.read_csv(path)

    # Count the rows and print the result
    row_count = (len(df))
    print('Analyzing {} rows of data'.format(row_count))

    return df

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--input_data", dest='input_data',
                        type=str)
    parser.add_argument("--output_datastore", dest='output_datastore',
                        type=str)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

To submit a job that runs the **move-data.py** script, run the cell below.

The job is configured to use the data asset `diabetes-local`, pointing to the local **diabetes.csv** file as input. The output is a path pointing to a folder in the new datastore `blob_training_data`.

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

# configure input and output
my_job_inputs = {
    "local_data": Input(type=AssetTypes.URI_FILE,
path="azureml:diabetes-local:1")
}

my_job_outputs = {
    "datastore_data": Output(type=AssetTypes.URI_FOLDER,
path="azureml://datastores/blob_training_data/paths/datastore-path")
}

# configure job
job = command(
    code="./src",
    command="python move-data.py --input_data ${{inputs.local_data}}
--output_datastore ${{outputs.datastore_data}}",
    inputs=my_job_inputs,
    outputs=my_job_outputs,
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="move-diabetes-data",
    experiment_name="move-diabetes-data"
)

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```