

Use datasets

5 minutes

Datasets are the primary way to pass data to experiments that train models.

Work with tabular datasets

You can read data directly from a tabular dataset by converting it into a Pandas or Spark dataframe:

Python

 Copy

```
df = tab_ds.to_pandas_dataframe()  
# code to work with dataframe goes here, for example:  
print(df.head())
```

Pass a tabular dataset to an experiment script

When you need to access a dataset in an experiment script, you must pass the dataset to the script. There are two ways you can do this.

Use a script argument for a tabular dataset

You can pass a tabular dataset as a script argument. When you take this approach, the argument received by the script is the unique ID for the dataset in your workspace. In the script, you can then get the workspace from the run context and use it to retrieve the dataset by its ID.

ScriptRunConfig:

Python

 Copy

```
env = Environment('my_env')  
packages = CondaDependencies.create(conda_packages=['pip'],  
                                   pip_packages=['azureml-defaults',  
                                                'azureml-dataprep[pandas]'])  
env.python.conda_dependencies = packages  
  
script_config = ScriptRunConfig(source_directory='my_dir',  
                                script='script.py',
```

```
arguments=['--ds', tab_ds],  
environment=env)
```

Script:

Python

 Copy

```
from azureml.core import Run, Dataset  
  
parser.add_argument('--ds', type=str, dest='dataset_id')  
args = parser.parse_args()  
  
run = Run.get_context()  
ws = run.experiment.workspace  
dataset = Dataset.get_by_id(ws, id=args.dataset_id)  
data = dataset.to_pandas_dataframe()
```

Use a named input for a tabular dataset

Alternatively, you can pass a tabular dataset as a *named input*. In this approach, you use the **as_named_input** method of the dataset to specify a name for the dataset. Then in the script, you can retrieve the dataset by name from the run context's **input_datasets** collection without needing to retrieve it from the workspace. Note that if you use this approach, you still need to include a script argument for the dataset, even though you don't actually use it to retrieve the dataset.

ScriptRunConfig:

Python

 Copy

```
env = Environment('my_env')  
packages = CondaDependencies.create(conda_packages=['pip'],  
                                   pip_packages=['azureml-defaults',  
                                                'azureml-dataprep[pandas]'])  
env.python.conda_dependencies = packages  
  
script_config = ScriptRunConfig(source_directory='my_dir',  
                                script='script.py',  
                                arguments=['--ds',  
tab_ds.as_named_input('my_dataset')],  
                                environment=env)
```

Script:

Python

 Copy


```
from azureml.core import Run

parser.add_argument('--ds', type=str, dest='ds_id')
args = parser.parse_args()

run = Run.get_context()
dataset = run.input_datasets['my_dataset']
data = dataset.to_pandas_dataframe()
```

Work with file datasets

When working with a file dataset, you can use the `to_path()` method to return a list of the file paths encapsulated by the dataset:

Python	 Copy
<pre>for file_path in file_ds.to_path(): print(file_path)</pre>	


Pass a file dataset to an experiment script

Just as with a Tabular dataset, there are two ways you can pass a file dataset to a script. However, there are some key differences in the way that the dataset is passed.

Use a script argument for a file dataset

You can pass a file dataset as a script argument. Unlike with a tabular dataset, you must specify a mode for the file dataset argument, which can be **as_download** or **as_mount**. This provides an access point that the script can use to read the files in the dataset. In most cases, you should use **as_download**, which copies the files to a temporary location on the compute where the script is being run. However, if you are working with a large amount of data for which there may not be enough storage space on the experiment compute, use **as_mount** to stream the files directly from their source.

ScriptRunConfig:

Python	 Copy
<pre>env = Environment('my_env') packages = CondaDependencies.create(conda_packages=['pip'], pip_packages=['azureml-defaults', 'azureml-dataprep[pandas]']) env.python.conda_dependencies = packages</pre>	

```
script_config = ScriptRunConfig(source_directory='my_dir',  
                                script='script.py',  
                                arguments=['--ds', file_ds.as_download()],  
                                environment=env)
```

Script:

Python

 Copy

```
from azureml.core import Run  
import glob  
  
parser.add_argument('--ds', type=str, dest='ds_ref')  
args = parser.parse_args()  
run = Run.get_context()  
  
imgs = glob.glob(args.ds_ref + "/*.jpg")
```

Use a named input for a file dataset

You can also pass a file dataset as a *named input*. In this approach, you use the **as_named_input** method of the dataset to specify a name before specifying the access mode. Then in the script, you can retrieve the dataset by name from the run context's **input_datasets** collection and read the files from there. As with tabular datasets, if you use a named input, you still need to include a script argument for the dataset, even though you don't actually use it to retrieve the dataset.

ScriptRunConfig:

Python

 Copy

```
env = Environment('my_env')  
packages = CondaDependencies.create(conda_packages=['pip'],  
                                   pip_packages=['azureml-defaults',  
                                                'azureml-dataprep[pandas]'])  
env.python.conda_dependencies = packages  
  
script_config = ScriptRunConfig(source_directory='my_dir',  
                                script='script.py',  
                                arguments=['--ds',  
file_ds.as_named_input('my_ds').as_download()],  
                                environment=env)
```

Script:

Python

 Copy

```
from azureml.core import Run
import glob

parser.add_argument('--ds', type=str, dest='ds_ref')
args = parser.parse_args()
run = Run.get_context()

dataset = run.input_datasets['my_ds']
imgs= glob.glob(dataset + "/*.jpg")
```

Next unit: Exercise - Work with data

[Continue >](#)