training

November 28, 2023

[1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
import os
import numpy as np
import dataiku
import pandas as pd, numpy as np
from dataiku import pandasutils as pdu
```

```
invalid escape sequence \s
invalid escape sequence \:
invalid escape sequence \:
```

[2]:
```
def get_dataset_objects(source_id, target_id):
    # Read recipe inputs
    training_dataset = dataiku.Folder(source_id)
    print("Training Dataset Info.")
    print(training_dataset.get_info())

    # Write recipe outputs
    models_folder = dataiku.Folder(target_id)
    print("Models Folder Info.")
    print(models_folder.get_info())

    return training_dataset, models_folder


source_id = "FvaDN1ly" #This is the folder id corresponding to␣
 ↪'training_dataset' folder
target_id = "EMHPYeer" #This is where my models should be saved

training_dataset, models_folder = get_dataset_objects(source_id, target_id)
```

```
Training Dataset Info.
{'projectKey': 'IMAGECLASSIFICATION', 'directoryBasedPartitioning': False,
'name': 'training_dataset', 'id': 'FvaDN1ly', 'accessInfo': {'bucket': 'gis-
```

```
data-ap-southeast-1', 'root': '/space-5dfbbb07-dku/node-2f7d11aa/managed-dss-
data/IMAGECLASSIFICATION/FvaDN1ly'}, 'type': 'S3'}
Models Folder Info.
{'projectKey': 'IMAGECLASSIFICATION', 'directoryBasedPartitioning': False,
'name': 'models', 'id': 'EMHPYeer', 'accessInfo': {'bucket': 'gis-data-ap-
southeast-1', 'root': '/space-5dfbbb07-dku/node-2f7d11aa/managed-dss-
data/IMAGECLASSIFICATION/EMHPYeer'}, 'type': 'S3'}
```

[4]:
```
training_dataset.list_paths_in_partition()
```

[4]:
```
['/test/Cat/13.jpg',
 '/test/Cat/14.jpg',
 '/test/Cat/15.jpg',
 '/test/Cat/17.jpg',
 '/test/Cat/29.jpg',
 '/test/Cat/31.jpg',
 '/test/Cat/33.jpg',
 '/test/Cat/34.jpg',
 '/test/Cat/43.jpg',
 '/test/Cat/57.jpg',
 '/test/Cat/6.jpg',
 '/test/Cat/68.jpg',
 '/test/Cat/78.jpg',
 '/test/Cat/79.jpg',
 '/test/Cat/84.jpg',
 '/test/Cat/88.jpg',
 '/test/Cat/89.jpg',
 '/test/Cat/9.jpg',
 '/test/Cat/91.jpg',
 '/test/Cat/94.jpg',
 '/test/Dog/0.jpg',
 '/test/Dog/11.jpg',
 '/test/Dog/13.jpg',
 '/test/Dog/14.jpg',
 '/test/Dog/2.jpg',
 '/test/Dog/23.jpg',
 '/test/Dog/4.jpg',
 '/test/Dog/42.jpg',
 '/test/Dog/45.jpg',
 '/test/Dog/46.jpg',
 '/test/Dog/48.jpg',
 '/test/Dog/61.jpg',
 '/test/Dog/72.jpg',
 '/test/Dog/73.jpg',
 '/test/Dog/76.jpg',
 '/test/Dog/77.jpg',
 '/test/Dog/82.jpg',
```

```
'/test/Dog/83.jpg',
'/test/Dog/9.jpg',
'/test/Dog/93.jpg',
'/train/Cat/1.jpg',
'/train/Cat/10.jpg',
'/train/Cat/11.jpg',
'/train/Cat/12.jpg',
'/train/Cat/16.jpg',
'/train/Cat/18.jpg',
'/train/Cat/19.jpg',
'/train/Cat/2.jpg',
'/train/Cat/25.jpg',
'/train/Cat/27.jpg',
'/train/Cat/28.jpg',
'/train/Cat/3.jpg',
'/train/Cat/30.jpg',
'/train/Cat/32.jpg',
'/train/Cat/35.jpg',
'/train/Cat/36.jpg',
'/train/Cat/38.jpg',
'/train/Cat/39.jpg',
'/train/Cat/4.jpg',
'/train/Cat/40.jpg',
'/train/Cat/41.jpg',
'/train/Cat/42.jpg',
'/train/Cat/44.jpg',
'/train/Cat/45.jpg',
'/train/Cat/46.jpg',
'/train/Cat/47.jpg',
'/train/Cat/48.jpg',
'/train/Cat/5.jpg',
'/train/Cat/51.jpg',
'/train/Cat/52.jpg',
'/train/Cat/53.jpg',
'/train/Cat/56.jpg',
'/train/Cat/58.jpg',
'/train/Cat/59.jpg',
'/train/Cat/60.jpg',
'/train/Cat/61.jpg',
'/train/Cat/62.jpg',
'/train/Cat/63.jpg',
'/train/Cat/64.jpg',
'/train/Cat/67.jpg',
'/train/Cat/7.jpg',
'/train/Cat/71.jpg',
'/train/Cat/72.jpg',
'/train/Cat/73.jpg',
```

```
'/train/Cat/74.jpg',
'/train/Cat/75.jpg',
'/train/Cat/76.jpg',
'/train/Cat/77.jpg',
'/train/Cat/8.jpg',
'/train/Cat/80.jpg',
'/train/Cat/81.jpg',
'/train/Cat/82.jpg',
'/train/Cat/83.jpg',
'/train/Cat/85.jpg',
'/train/Cat/90.jpg',
'/train/Cat/92.jpg',
'/train/Cat/93.jpg',
'/train/Cat/95.jpg',
'/train/Cat/96.jpg',
'/train/Cat/97.jpg',
'/train/Dog/1.jpg',
'/train/Dog/10.jpg',
'/train/Dog/12.jpg',
'/train/Dog/15.jpg',
'/train/Dog/18.jpg',
'/train/Dog/19.jpg',
'/train/Dog/21.jpg',
'/train/Dog/22.jpg',
'/train/Dog/24.jpg',
'/train/Dog/25.jpg',
'/train/Dog/26.jpg',
'/train/Dog/28.jpg',
'/train/Dog/29.jpg',
'/train/Dog/3.jpg',
'/train/Dog/30.jpg',
'/train/Dog/31.jpg',
'/train/Dog/32.jpg',
'/train/Dog/33.jpg',
'/train/Dog/34.jpg',
'/train/Dog/35.jpg',
'/train/Dog/36.jpg',
'/train/Dog/38.jpg',
'/train/Dog/43.jpg',
'/train/Dog/44.jpg',
'/train/Dog/47.jpg',
'/train/Dog/49.jpg',
'/train/Dog/50.jpg',
'/train/Dog/51.jpg',
'/train/Dog/52.jpg',
'/train/Dog/53.jpg',
'/train/Dog/55.jpg',
```

```
'/train/Dog/56.jpg',
'/train/Dog/57.jpg',
'/train/Dog/58.jpg',
'/train/Dog/62.jpg',
'/train/Dog/63.jpg',
'/train/Dog/64.jpg',
'/train/Dog/65.jpg',
'/train/Dog/66.jpg',
'/train/Dog/67.jpg',
'/train/Dog/68.jpg',
'/train/Dog/7.jpg',
'/train/Dog/70.jpg',
'/train/Dog/71.jpg',
'/train/Dog/74.jpg',
'/train/Dog/75.jpg',
'/train/Dog/78.jpg',
'/train/Dog/79.jpg',
'/train/Dog/80.jpg',
'/train/Dog/84.jpg',
'/train/Dog/85.jpg',
'/train/Dog/86.jpg',
'/train/Dog/88.jpg',
'/train/Dog/89.jpg',
'/train/Dog/90.jpg',
'/train/Dog/91.jpg',
'/train/Dog/92.jpg',
'/train/Dog/94.jpg',
'/train/Dog/96.jpg',
'/train/Dog/97.jpg',
'/valid/Cat/0.jpg',
'/valid/Cat/20.jpg',
'/valid/Cat/21.jpg',
'/valid/Cat/22.jpg',
'/valid/Cat/23.jpg',
'/valid/Cat/24.jpg',
'/valid/Cat/26.jpg',
'/valid/Cat/37.jpg',
'/valid/Cat/49.jpg',
'/valid/Cat/50.jpg',
'/valid/Cat/54.jpg',
'/valid/Cat/55.jpg',
'/valid/Cat/65.jpg',
'/valid/Cat/66.jpg',
'/valid/Cat/69.jpg',
'/valid/Cat/70.jpg',
'/valid/Cat/86.jpg',
'/valid/Cat/87.jpg',
```

```
 '/valid/Cat/98.jpg',
 '/valid/Cat/99.jpg',
 '/valid/Dog/16.jpg',
 '/valid/Dog/17.jpg',
 '/valid/Dog/20.jpg',
 '/valid/Dog/27.jpg',
 '/valid/Dog/37.jpg',
 '/valid/Dog/39.jpg',
 '/valid/Dog/40.jpg',
 '/valid/Dog/41.jpg',
 '/valid/Dog/5.jpg',
 '/valid/Dog/54.jpg',
 '/valid/Dog/59.jpg',
 '/valid/Dog/6.jpg',
 '/valid/Dog/60.jpg',
 '/valid/Dog/69.jpg',
 '/valid/Dog/8.jpg',
 '/valid/Dog/81.jpg',
 '/valid/Dog/87.jpg',
 '/valid/Dog/95.jpg',
 '/valid/Dog/98.jpg',
 '/valid/Dog/99.jpg']
```

[5]:
```
training_dataset.get_info()
```

[5]:
```
{'projectKey': 'IMAGECLASSIFICATION',
 'directoryBasedPartitioning': False,
 'name': 'training_dataset',
 'id': 'FvaDN1ly',
 'accessInfo': {'bucket': 'gis-data-ap-southeast-1',
  'root': '/space-5dfbbb07-dku/node-2f7d11aa/managed-dss-
data/IMAGECLASSIFICATION/FvaDN1ly'},
 'type': 'S3'}
```

[6]:
```
dataset_folders = [training_dataset.
 ↪get_path_details()['children'][folder_number]['name']
             for folder_number in range(len(training_dataset.
 ↪get_path_details()['children']))]
dataset_folders
```

[6]:
```
['test', 'train', 'valid']
```

[7]:
```
def create_model():
    model = Sequential([
        Flatten(input_shape=(64, 64, 3)),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
```

```
        Dense(2, activation='softmax')  # Adjust the number of units for your␣
↪classification
    ])

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
    return model
```

[ ]:
```
def train_model(train_data_path, valid_data_path, models_path,␣
↪save_labels_path):
    train_datagen = ImageDataGenerator(rescale=1./255)
    valid_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        directory='/train',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary'
    )

    valid_generator = valid_datagen.flow_from_directory(
        directory='/valid',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary'
    )

    # Save class labels
    class_labels = list(train_generator.class_indices.keys())
    np.save(save_labels_path, class_labels)

    model = create_model()

    checkpoint_callback = ModelCheckpoint(
        filepath=os.path.join(models_path, 'model_{epoch:02d}.h5'),
        save_freq='epoch',
        save_best_only=False,
        save_weights_only=False,
        verbose=1
    )

    model.fit(
        train_generator,
        epochs=5,
        validation_data=valid_generator,
        callbacks=[checkpoint_callback]
    )
```

```
    # Save the final model
    model.save(os.path.join(models_path, 'final_model.h5'))

if __name__ == "__main__":
    root_path = "/"
    train_data_path = os.path.join(root_path, "training_dataset")
    models_path = os.path.join(root_path, "models")
    save_labels_path = os.path.join(models_path, "class_labels.npy")

    train_model(train_data_path, train_data_path, models_path, save_labels_path)
```

[4]:
```
training_dataset.list_paths_in_partition()
```

[4]:
```
['/test/Cat/10.jpg',
 '/test/Cat/12.jpg',
 '/test/Cat/13.jpg',
 '/test/Cat/14.jpg',
 '/test/Cat/15.jpg',
 '/test/Cat/16.jpg',
 '/test/Cat/17.jpg',
 '/test/Cat/20.jpg',
 '/test/Cat/22.jpg',
 '/test/Cat/23.jpg',
 '/test/Cat/26.jpg',
 '/test/Cat/28.jpg',
 '/test/Cat/29.jpg',
 '/test/Cat/31.jpg',
 '/test/Cat/32.jpg',
 '/test/Cat/33.jpg',
 '/test/Cat/34.jpg',
 '/test/Cat/35.jpg',
 '/test/Cat/37.jpg',
 '/test/Cat/40.jpg',
 '/test/Cat/43.jpg',
 '/test/Cat/44.jpg',
 '/test/Cat/50.jpg',
 '/test/Cat/51.jpg',
 '/test/Cat/57.jpg',
 '/test/Cat/6.jpg',
 '/test/Cat/60.jpg',
 '/test/Cat/68.jpg',
 '/test/Cat/78.jpg',
 '/test/Cat/79.jpg',
 '/test/Cat/83.jpg',
 '/test/Cat/84.jpg',
 '/test/Cat/88.jpg',
```

```
'/test/Cat/89.jpg',
'/test/Cat/9.jpg',
'/test/Cat/91.jpg',
'/test/Cat/94.jpg',
'/test/Dog/0.jpg',
'/test/Dog/10.jpg',
'/test/Dog/11.jpg',
'/test/Dog/13.jpg',
'/test/Dog/14.jpg',
'/test/Dog/2.jpg',
'/test/Dog/23.jpg',
'/test/Dog/24.jpg',
'/test/Dog/29.jpg',
'/test/Dog/3.jpg',
'/test/Dog/32.jpg',
'/test/Dog/35.jpg',
'/test/Dog/36.jpg',
'/test/Dog/4.jpg',
'/test/Dog/42.jpg',
'/test/Dog/45.jpg',
'/test/Dog/46.jpg',
'/test/Dog/47.jpg',
'/test/Dog/48.jpg',
'/test/Dog/61.jpg',
'/test/Dog/64.jpg',
'/test/Dog/70.jpg',
'/test/Dog/72.jpg',
'/test/Dog/73.jpg',
'/test/Dog/76.jpg',
'/test/Dog/77.jpg',
'/test/Dog/81.jpg',
'/test/Dog/82.jpg',
'/test/Dog/83.jpg',
'/test/Dog/86.jpg',
'/test/Dog/87.jpg',
'/test/Dog/9.jpg',
'/test/Dog/93.jpg',
'/test/Dog/96.jpg',
'/test/Dog/99.jpg',
'/train/Cat/0.jpg',
'/train/Cat/1.jpg',
'/train/Cat/10.jpg',
'/train/Cat/11.jpg',
'/train/Cat/12.jpg',
'/train/Cat/13.jpg',
'/train/Cat/14.jpg',
'/train/Cat/16.jpg',
```

```
'/train/Cat/17.jpg',
'/train/Cat/18.jpg',
'/train/Cat/19.jpg',
'/train/Cat/2.jpg',
'/train/Cat/21.jpg',
'/train/Cat/24.jpg',
'/train/Cat/25.jpg',
'/train/Cat/27.jpg',
'/train/Cat/28.jpg',
'/train/Cat/3.jpg',
'/train/Cat/30.jpg',
'/train/Cat/31.jpg',
'/train/Cat/32.jpg',
'/train/Cat/33.jpg',
'/train/Cat/34.jpg',
'/train/Cat/35.jpg',
'/train/Cat/36.jpg',
'/train/Cat/38.jpg',
'/train/Cat/39.jpg',
'/train/Cat/4.jpg',
'/train/Cat/40.jpg',
'/train/Cat/41.jpg',
'/train/Cat/42.jpg',
'/train/Cat/44.jpg',
'/train/Cat/45.jpg',
'/train/Cat/46.jpg',
'/train/Cat/47.jpg',
'/train/Cat/48.jpg',
'/train/Cat/5.jpg',
'/train/Cat/51.jpg',
'/train/Cat/52.jpg',
'/train/Cat/53.jpg',
'/train/Cat/54.jpg',
'/train/Cat/55.jpg',
'/train/Cat/56.jpg',
'/train/Cat/57.jpg',
'/train/Cat/58.jpg',
'/train/Cat/59.jpg',
'/train/Cat/6.jpg',
'/train/Cat/60.jpg',
'/train/Cat/61.jpg',
'/train/Cat/62.jpg',
'/train/Cat/63.jpg',
'/train/Cat/64.jpg',
'/train/Cat/65.jpg',
'/train/Cat/67.jpg',
'/train/Cat/69.jpg',
```

```
'/train/Cat/7.jpg',
'/train/Cat/70.jpg',
'/train/Cat/71.jpg',
'/train/Cat/72.jpg',
'/train/Cat/73.jpg',
'/train/Cat/74.jpg',
'/train/Cat/75.jpg',
'/train/Cat/76.jpg',
'/train/Cat/77.jpg',
'/train/Cat/78.jpg',
'/train/Cat/8.jpg',
'/train/Cat/80.jpg',
'/train/Cat/81.jpg',
'/train/Cat/82.jpg',
'/train/Cat/83.jpg',
'/train/Cat/85.jpg',
'/train/Cat/89.jpg',
'/train/Cat/9.jpg',
'/train/Cat/90.jpg',
'/train/Cat/91.jpg',
'/train/Cat/92.jpg',
'/train/Cat/93.jpg',
'/train/Cat/95.jpg',
'/train/Cat/96.jpg',
'/train/Cat/97.jpg',
'/train/Cat/98.jpg',
'/train/Cat/99.jpg',
'/train/Dog/1.jpg',
'/train/Dog/10.jpg',
'/train/Dog/12.jpg',
'/train/Dog/14.jpg',
'/train/Dog/15.jpg',
'/train/Dog/16.jpg',
'/train/Dog/17.jpg',
'/train/Dog/18.jpg',
'/train/Dog/19.jpg',
'/train/Dog/20.jpg',
'/train/Dog/21.jpg',
'/train/Dog/22.jpg',
'/train/Dog/23.jpg',
'/train/Dog/24.jpg',
'/train/Dog/25.jpg',
'/train/Dog/26.jpg',
'/train/Dog/27.jpg',
'/train/Dog/28.jpg',
'/train/Dog/29.jpg',
'/train/Dog/3.jpg',
```

```
'/train/Dog/30.jpg',
'/train/Dog/31.jpg',
'/train/Dog/32.jpg',
'/train/Dog/33.jpg',
'/train/Dog/34.jpg',
'/train/Dog/35.jpg',
'/train/Dog/36.jpg',
'/train/Dog/37.jpg',
'/train/Dog/38.jpg',
'/train/Dog/39.jpg',
'/train/Dog/4.jpg',
'/train/Dog/40.jpg',
'/train/Dog/41.jpg',
'/train/Dog/43.jpg',
'/train/Dog/44.jpg',
'/train/Dog/45.jpg',
'/train/Dog/46.jpg',
'/train/Dog/47.jpg',
'/train/Dog/49.jpg',
'/train/Dog/5.jpg',
'/train/Dog/50.jpg',
'/train/Dog/51.jpg',
'/train/Dog/52.jpg',
'/train/Dog/53.jpg',
'/train/Dog/54.jpg',
'/train/Dog/55.jpg',
'/train/Dog/56.jpg',
'/train/Dog/57.jpg',
'/train/Dog/58.jpg',
'/train/Dog/59.jpg',
'/train/Dog/6.jpg',
'/train/Dog/60.jpg',
'/train/Dog/61.jpg',
'/train/Dog/62.jpg',
'/train/Dog/63.jpg',
'/train/Dog/64.jpg',
'/train/Dog/65.jpg',
'/train/Dog/66.jpg',
'/train/Dog/67.jpg',
'/train/Dog/68.jpg',
'/train/Dog/7.jpg',
'/train/Dog/70.jpg',
'/train/Dog/71.jpg',
'/train/Dog/73.jpg',
'/train/Dog/74.jpg',
'/train/Dog/75.jpg',
'/train/Dog/78.jpg',
```

```
'/train/Dog/79.jpg',
'/train/Dog/80.jpg',
'/train/Dog/82.jpg',
'/train/Dog/84.jpg',
'/train/Dog/85.jpg',
'/train/Dog/86.jpg',
'/train/Dog/88.jpg',
'/train/Dog/89.jpg',
'/train/Dog/9.jpg',
'/train/Dog/90.jpg',
'/train/Dog/91.jpg',
'/train/Dog/92.jpg',
'/train/Dog/93.jpg',
'/train/Dog/94.jpg',
'/train/Dog/96.jpg',
'/train/Dog/97.jpg',
'/train/Dog/98.jpg',
'/valid/Cat/0.jpg',
'/valid/Cat/15.jpg',
'/valid/Cat/20.jpg',
'/valid/Cat/21.jpg',
'/valid/Cat/22.jpg',
'/valid/Cat/23.jpg',
'/valid/Cat/24.jpg',
'/valid/Cat/26.jpg',
'/valid/Cat/37.jpg',
'/valid/Cat/4.jpg',
'/valid/Cat/47.jpg',
'/valid/Cat/49.jpg',
'/valid/Cat/50.jpg',
'/valid/Cat/52.jpg',
'/valid/Cat/54.jpg',
'/valid/Cat/55.jpg',
'/valid/Cat/65.jpg',
'/valid/Cat/66.jpg',
'/valid/Cat/68.jpg',
'/valid/Cat/69.jpg',
'/valid/Cat/70.jpg',
'/valid/Cat/71.jpg',
'/valid/Cat/72.jpg',
'/valid/Cat/73.jpg',
'/valid/Cat/76.jpg',
'/valid/Cat/77.jpg',
'/valid/Cat/79.jpg',
'/valid/Cat/81.jpg',
'/valid/Cat/82.jpg',
'/valid/Cat/84.jpg',
```

```
 '/valid/Cat/86.jpg',
 '/valid/Cat/87.jpg',
 '/valid/Cat/94.jpg',
 '/valid/Cat/95.jpg',
 '/valid/Cat/98.jpg',
 '/valid/Cat/99.jpg',
 '/valid/Dog/0.jpg',
 '/valid/Dog/11.jpg',
 '/valid/Dog/16.jpg',
 '/valid/Dog/17.jpg',
 '/valid/Dog/20.jpg',
 '/valid/Dog/27.jpg',
 '/valid/Dog/37.jpg',
 '/valid/Dog/39.jpg',
 '/valid/Dog/40.jpg',
 '/valid/Dog/41.jpg',
 '/valid/Dog/43.jpg',
 '/valid/Dog/49.jpg',
 '/valid/Dog/5.jpg',
 '/valid/Dog/53.jpg',
 '/valid/Dog/54.jpg',
 '/valid/Dog/59.jpg',
 '/valid/Dog/6.jpg',
 '/valid/Dog/60.jpg',
 '/valid/Dog/62.jpg',
 '/valid/Dog/63.jpg',
 '/valid/Dog/67.jpg',
 '/valid/Dog/69.jpg',
 '/valid/Dog/72.jpg',
 '/valid/Dog/77.jpg',
 '/valid/Dog/79.jpg',
 '/valid/Dog/8.jpg',
 '/valid/Dog/81.jpg',
 '/valid/Dog/83.jpg',
 '/valid/Dog/84.jpg',
 '/valid/Dog/85.jpg',
 '/valid/Dog/87.jpg',
 '/valid/Dog/91.jpg',
 '/valid/Dog/94.jpg',
 '/valid/Dog/95.jpg',
 '/valid/Dog/97.jpg',
 '/valid/Dog/98.jpg',
 '/valid/Dog/99.jpg']
```

[5]:
```
help(training_dataset)
```

```
Help on Folder in module dataiku.core.managed_folder object:
```

```
class Folder(dataiku.core.base.Computable)
 |  Folder(lookup, project_key=None, ignore_flow=False)
 |
 |  Handle to interact with a folder.
 |
 |  .. note::
 |
 |      This class is also available as ``dataiku.Folder``
 |
 |  Method resolution order:
 |      Folder
 |      dataiku.core.base.Computable
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, lookup, project_key=None, ignore_flow=False)
 |      Obtain a handle for a named folder.
 |
 |      :param string lookup: name or identifier of the managed folder
 |      :param string project_key: project key of the managed folder, if it is
not in the current project.
 |      :param boolean ignore_flow: if True, create the handle regardless of
whether the managed folder is an input
 |                                  or output of the recipe (default: False)
 |
 |  clear(self)
 |      Remove all files from the folder.
 |
 |  clear_partition(self, partition)
 |      Remove all files from a specific partition of the folder.
 |
 |      :param string partition: identifier of the partition to clear
 |
 |  clear_path(self, path)
 |      Remove a file or directory from the managed folder.
 |
 |      .. caution::
 |
 |          Deprecated. use :meth:`delete_path` instead
 |
 |      :param string path: path inside the folder to the file or directory to
delete
 |
 |  delete_path(self, path)
 |      Remove a file or directory from the managed folder.
 |
```

```
 |      :param string path: path inside the folder to the file or directory to
delete
 |
 |  file_path(self, filename)
 |      Get the filesystem path for a given file within the folder.
 |
 |      .. important::
 |          This method can only be called for managed folders that are stored
on the local filesystem of the DSS server. For
 |          non-filesystem managed folders (HDFS, S3, …), you need to use the
various read/download and write/upload APIs.
 |
 |      :param string filename: path of the file within the folder
 |
 |      :returns: the full path of the file on the local filesystem
 |      :rtype: string
 |
 |  get_download_stream(self, path)
 |      Get a file-like object that allows you to read a single file from this
folder.
 |
 |      Usage example:
 |
 |      .. code-block:: python
 |
 |          with folder.get_download_stream("myfile") as stream:
 |              data = stream.readline()
 |              print("First line of myfile is: {}".format(data))
 |
 |      .. note::
 |          The file-like returned by this method is not seekable.
 |
 |      :param string path: path inside the managed folder
 |
 |      :returns: the data of the file at `path` inside the managed folder
 |      :rtype: file-like
 |
 |  get_id(self)
 |      Get the identifier of the folder.
 |
 |      :rtype: string
 |
 |  get_info(self, sensitive_info=False)
 |      Get information about the location and settings of this managed folder
 |
 |      Usage sample:
 |
 |      .. code-block:: python
```

```
 |
 |          # construct the URL to a S3 object
 |          folder = dataiku.Folder("my_folder_name")
 |          folder_info = folder.get_info()
 |          access_info = folder_info["accessInfo"]
 |          folder_base_url = 's3://%s%s' % (access_info['bucket'],
access_info['root'])
 |          target_url = '%s/some/path/to/a/file' % folder_base_url
 |
 |      :param boolean sensitive_info: if True, the credentials of the
connection of the managed folder are
 |                                     returned, if they're accessible to the
user. (default: False)
 |
 |      :returns: information about the folder. Fields are:
 |
 |                  * **id** : identifier of the folder
 |                  * **projectKey** : project of the folder
 |                  * **name** : name of the folder
 |                  * **type** : type of the folder (S3 / HDFS / GCS / …)
 |                  * **directoryBasedPartitioning** : whether the partitioning
schema of the folder (if any) maps partitions to sub-folders
 |                  * **path** : path of the folder of the filesystem, for
folder on the local filesystem
 |                  * **accessInfo** : extra information about the filesystem
underlying the folder. The exact fields depend on the folder type. Typically
contains the connection root and the parts needed to build a full URI, like
bucket and storage account name. If the **sensitive_info** parameter is True,
then credentials of the connection will be added (if accessible to the user)
 |
 |      :rtype: dict
 |
 |  get_last_check_values(self, partition='')
 |      Get the set of last values of the checks on this folder, as a
:class:`dataiku.core.metrics.ComputedChecks` object
 |
 |      :param string partition: (optional), the partition for which to fetch
the values. On partitioned folders,
 |                               the partition value to use for accessing
metrics on the whole dataset (ie. all
 |                               partitions) is ALL
 |
 |      :rtype: :class:`dataiku.core.metrics.ComputedChecks`
 |
 |  get_last_metric_values(self, partition='')
 |      Get the set of last values of the metrics on this folder.
 |
 |      :param string partition: (optional) a partition identifier to get
```

```
metrics for. If not set, returns the metrics
 |                                of a non-partitioned folder, and the metrics of
the whole managed folder for a
 |                                partitioned managed folder
 |
 |      :rtype: :class:`dataiku.core.metrics.ComputedMetrics`
 |
 |  get_metric_history(self, metric_lookup, partition='')
 |      Get the set of all values a given metric took on this folder.
 |
 |      :param string metric_lookup: metric name or unique identifier
 |      :param string partition: (optional) a partition identifier to get
metrics for. If not set, returns the metrics
 |                                of a non-partitioned folder, and the metrics of
the whole managed folder for a
 |                                partitioned managed folder
 |
 |      :returns: an object containing the values of the `metric_lookup` metric,
cast to the appropriate type (double,
 |                boolean, …). Top-level fields are:
 |
 |                  * **metricId** : identifier of the metric
 |                  * **metric** : dict of the metric's definition
 |                  * **valueType** : type of the metric values in the values
array
 |                  * **lastValue** : most recent value, as a dict of:
 |
 |                    * **time** : timestamp of the value computation
 |                    * **value** : value of the metric at time
 |
 |                  * **values** : list of values, each one a dict of the same
structure as **lastValue**
 |
 |      :rtype: dict
 |
 |  get_name(self)
 |      Get the name of the folder.
 |
 |      :rtype: string
 |
 |  get_partition_folder(self, partition)
 |      Get the filesystem path of the directory corresponding to the partition
(if the partitioning is directory-based).
 |
 |      :param string partition: identifier of the partition
 |
 |      :returns: sub-path inside the folder that corresponds to the partition.
None if the partitioning scheme doesn't map
```

```
 |                partitions to sub-folders
 |      :rtype: string
 |
 |  get_partition_info(self, partition)
 |      Get information about a partition of this managed folder
 |
 |      :param string partition: partition identifier
 |
 |      :returns: information about the partition. Fields are:
 |
 |                  * **id** : identifier of the folder
 |                  * **projectKey** : project of the folder
 |                  * **name** : name of the folder
 |                  * **folder** : if the partitioning scheme maps partitions to
a subfolder, the path of the subfolder within the managed folder
 |                  * **paths** : paths of the files in the partition, relative
to the managed folder
 |
 |      :rtype: dict
 |
 |  get_path(self)
 |      Get the filesystem path of this managed folder.
 |
 |      .. important::
 |          This method can only be called for managed folders that are stored
on the local filesystem of the DSS server. For
 |          non-filesystem managed folders (HDFS, S3, …), you need to use the
various read/download and write/upload APIs.
 |
 |      Usage example:
 |
 |      .. code-block:: python
 |
 |          # read a model off a local managed folder
 |          folder = dataiku.Folder("folder_where_models_are_stored")
 |          with open(os.path.join(f.get_path(), "path/to/model.pkl"), 'rb') as
fd:
 |              model = pickle.load(fd)
 |
 |      :returns: a path on the local filesystem that the python process can
read from and write to
 |      :rtype: string
 |
 |  get_path_details(self, path='/')
 |      Get details about a specific path (file or directory) in the folder.
 |
 |      :param string path: path inside the folder to the file or directory
 |
```

```
 |      :returns: information about the file or folder at `path`, as a dict.
Fields are:
 |
 |                  * **exists** : whether there is a file or folder at `path`
 |                  * **directory** : True if `path` denotes a directory in the
managed folder, False if it's a file
 |                  * **fullPath** : path inside the folder
 |                  * **size** : if a file, the size in bytes of the file
 |                  * **lastModified** : last modification time of the file or
directory at `path`, in milliseconds since epoch
 |                  * **mimeType** : for files, the detected MIME type
 |                  * **children** : for directories, a list of the contents of
the directory, each element having the present structure (not recursive)
 |
 |      :rtype: dict
 |
 |  get_writer(self, path)
 |      Get a writer object to write incrementally to a specific path in the
managed folder.
 |
 |      If the file already exists, it will be replaced.
 |
 |      :param string path: Target path of the file to write in the managed
folder
 |
 |      :rtype: :class:`dataiku.core.managed_folder.ManagedFolderWriter`
 |
 |  is_partitioning_directory_based(self)
 |      Whether the partitioning of the folder maps partitions to sub-
directories.
 |
 |      :rtype: boolean
 |
 |  list_partitions(self)
 |      Get the partitions in the folder.
 |
 |      :returns: a list of partition identifiers
 |      :rtype: list[string]
 |
 |  list_paths_in_partition(self, partition='')
 |      Gets the paths of the files for the given partition.
 |
 |      :param string partition: identifier of the partition. Use `''` to get
the paths of all the files in the folder, regardless
 |                               of the partition
 |
 |      :returns: a list of paths within the folder
 |      :rtype: list[string]
```

```
 |
 |  read_json(self, filename)
 |      Read a JSON file within the folder and return its parsed content.
 |
 |      Usage example:
 |
 |      .. code-block:: python
 |
 |          folder = dataiku.Folder("my_folder_id")
 |          # write a JSON-serializable object
 |          folder.write_json("/some/path/in/folder", my_object)
 |
 |          # read back the object
 |          my_object_again = folder.read_json("/some/path/in/folder")
 |
 |      :param string filename: path of the file within the folder
 |
 |      :returns: the content of the file
 |      :rtype: list or dict, depending on the content of the file
 |
 |  save_external_check_values(self, values_dict, partition='')
 |      Save checks on this folder. The checks are saved with the type
"external".
 |
 |      :param dict values_dict: the values to save, as a dict. The keys of the
dict are used as check names
 |      :param string partition: (optional), the partition for which to fetch
the values. On partitioned folders,
 |                               the partition value to use for accessing
metrics on the whole dataset (ie. all
 |                               partitions) is ALL
 |
 |  save_external_metric_values(self, values_dict, partition='')
 |      Save metrics on this folder. The metrics are saved with the type
"external".
 |
 |      :param dict values_dict: the values to save, as a dict. The keys of the
dict are used as metric names
 |      :param string partition: (optional), the partition for which to fetch
the values. On partitioned folders,
 |                               the partition value to use for accessing
metrics on the whole dataset (ie. all
 |                               partitions) is ALL
 |
 |  upload_data(self, path, data)
 |      Upload binary data to a specific path in the managed folder.
 |
 |      If the file already exists, it will be replaced.
```

```
 |
 |      :param string path: Target path of the file to write in the managed
folder
 |      :param bytes data: str or unicode data to upload
 |
 |  upload_file(self, path, file_path)
 |      Upload a local file to a specific path in the managed folder.
 |
 |      If the file already exists, it will be replaced.
 |
 |      :param string path: Target path of the file to write in the managed
folder
 |      :param string file_path: Absolute path to a local file
 |
 |  upload_stream(self, path, f)
 |      Upload the content of a file-like object to a specific path in the
managed folder.
 |
 |      If the file already exists, it will be replaced.
 |
 |      .. code-block:: python
 |
 |          # This copies a local file to the managed folder
 |          with open("local_file_to_upload") as f:
 |              folder.upload_stream("name_of_file_in_folder", f)
 |
 |      :param string path: Target path of the file to write in the managed
folder
 |      :param stream f: file-like object open for reading
 |
 |  write_json(self, filename, obj)
 |      Write a JSON-serializable object as JSON to a file within the folder.
 |
 |      :param string filename: Path of the target file within the folder
 |      :param object obj: JSON-serializable object to write (generally dict or
list)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from dataiku.core.base.Computable:
 |
 |  add_read_partitions(self, spec)
 |      Add a partition or range of partitions to read.
 |
 |      The spec argument must be given in the DSS partition spec format.
 |      You cannot manually set partitions when running inside
 |      a Python recipe. They are automatically set using the dependencies.
 |
 |  set_write_partition(self, spec)
```

```
|      Sets which partition of the dataset gets written to when
|      you create a DatasetWriter. Setting the write partition is
|      not allowed in Python recipes, where write is controlled by
|      the Flow.
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from dataiku.core.base.Computable:
|
|  full_name
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from dataiku.core.base.Computable:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```

[ ]: