Use with Pandas

This document is a quick introduction to using `datasets` with Pandas, with a particular focus on how to process
datasets using Pandas functions, and how to convert a dataset to Pandas or from Pandas.

This is particularly useful as it allows fast operations, since `datasets` uses PyArrow under the hood and PyArrow is well integrated with Pandas.

Dataset format

By default, datasets return regular Python objects: integers, floats, strings, lists, etc.

To get Pandas DataFrames or Series instead, you can set the format of the dataset to `pandas` using Dataset.with_format():

```
>>> from datasets import Dataset
>>> data = {"col_0": ["a", "b", "c", "d"], "col_1": [0., 0., 1., 1.]}
>>> ds = Dataset.from_dict(data)
>>> ds = ds.with_format("pandas")
>>> ds[0]       # pd.DataFrame
  col_0  col_1
0     a    0.0
>>> ds[:2]      # pd.DataFrame
  col_0  col_1
0     a    0.0
1     b    0.0
>>> ds["data"]  # pd.Series
0    a
1    b
2    c
3    d
Name: col_0, dtype: object
```

This also works for `IterableDataset` objects obtained e.g. using `load_dataset(..., streaming=True)`:

```
>>> ds = ds.with_format("pandas")
>>> for df in ds.iter(batch_size=2):
...     print(df)
...     break
  col_0  col_1
0     a    0.0
1     b    0.0
```

Process data

Pandas functions are generally faster than regular hand-written python functions, and therefore they are a good option to optimize data processing. You can use Pandas functions to process a dataset in Dataset.map() or Dataset.filter():

```
>>> from datasets import Dataset
>>> data = {"col_0": ["a", "b", "c", "d"], "col_1": [0., 0., 1., 1.]}
>>> ds = Dataset.from_dict(data)
>>> ds = ds.with_format("pandas")
>>> ds = ds.map(lambda df: df.assign(col_2=df.col_1 + 1), batched=True)
>>> ds[:2]
  col_0  col_1  col_2
0     a    0.0    1.0
1     b    0.0    1.0
>>> ds = ds.filter(lambda df: df.col_0 == "b", batched=True)
>>> ds[0]
  col_0  col_1  col_2
0     b    0.0    1.0
```

We use `batched=True` because it is faster to process batches of data in Pandas rather than row by row. It's also possible to use `batch_size=` in `map()` to set the size of each `df`.

This also works for IterableDataset.map() and IterableDataset.filter().

Import or Export from Pandas

To import data from Pandas, you can use Dataset.from_pandas():

```
ds = Dataset.from_pandas(df)
```

And you can use Dataset.to_pandas() to export a Dataset to a Pandas DataFrame:

```
df = Dataset.to_pandas()
```