Use with Polars

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

This is particularly useful as it allows fast zero-copy operations, since both `datasets` and Polars use Arrow under the hood.

Dataset format

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

To get Polars DataFrames or Series instead, you can set the format of the dataset to `polars` 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("polars")
>>> ds[0]       # pl.DataFrame
shape: (1, 2)
┌───────┬───────┐
│ col_0 ┆ col_1 │
│ ---   ┆ ---   │
│ str   ┆ f64   │
╞═══════╪═══════╡
│ a     ┆ 0.0   │
└───────┴───────┘
>>> ds[:2]      # pl.DataFrame
shape: (2, 2)
┌───────┬───────┐
│ col_0 ┆ col_1 │
│ ---   ┆ ---   │
│ str   ┆ f64   │
╞═══════╪═══════╡
│ a     ┆ 0.0   │
│ b     ┆ 0.0   │
└───────┴───────┘
>>> ds["data"]  # pl.Series
shape: (4,)
Series: 'col_0' [str]
[
        "a"
        "b"
        "c"
        "d"
]
```

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

```
>>> ds = ds.with_format("polars")
>>> for df in ds.iter(batch_size=2):
...     print(df)
...     break
shape: (2, 2)
┌───────┬───────┐
│ col_0 ┆ col_1 │
│ ---   ┆ ---   │
│ str   ┆ f64   │
╞═══════╪═══════╡
│ a     ┆ 0.0   │
│ b     ┆ 0.0   │
└───────┴───────┘
```

Process data

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

```
>>> import polars as pl
>>> 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("polars")
>>> ds = ds.map(lambda df: df.with_columns(pl.col("col_1").add(1).alias("col_2")), batched=True)
>>> ds[:2]
shape: (2, 3)
┌───────┬───────┬───────┐
│ col_0 ┆ col_1 ┆ col_2 │
│ ---   ┆ ---   ┆ ---   │
│ str   ┆ f64   ┆ f64   │
╞═══════╪═══════╪═══════╡
│ a     ┆ 0.0   ┆ 1.0   │
│ b     ┆ 0.0   ┆ 1.0   │
└───────┴───────┴───────┘
>>> ds = ds.filter(lambda df: df["col_0"] == "b", batched=True)
>>> ds[0]
shape: (1, 3)
┌───────┬───────┬───────┐
│ col_0 ┆ col_1 ┆ col_2 │
│ ---   ┆ ---   ┆ ---   │
│ str   ┆ f64   ┆ f64   │
╞═══════╪═══════╪═══════╡
│ b     ┆ 0.0   ┆ 1.0   │
└───────┴───────┴───────┘
```

We use `batched=True` because it is faster to process batches of data in Polars 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().

Example: data extraction

Many functions are available in Polars and for any data type: string, floats, integers, etc. You can find the full list here. Those functions are written in Rust and run on batches of data which enables fast data processing.

Here is an example that shows a 5x speed boost using Polars instead of a regular python function to extract solutions from a LLM reasoning dataset:

```
from datasets import load_dataset

ds = load_dataset("ServiceNow-AI/R1-Distill-SFT", "v0", split="train")

# Using a regular python function
pattern = re.compile("boxed\\{(.*)\\}")
result_ds = ds.map(lambda x: {"value_solution": m.group(1) if (m:=pattern.search(x["solution"])) e
# Time: 10s

# Using a Polars function
expr = pl.col("solution").str.extract("boxed\\{(.*)\\}").alias("value_solution")
result_ds = ds.with_format("polars").map(lambda df: df.with_columns(expr), batched=True)
# Time: 2s
```

Import or Export from Polars

To import data from Polars, you can use `Dataset.from_polars()`:

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

And you can use `Dataset.to_polars()` to export a Dataset to a Polars DataFrame:

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