Load tabular data

A tabular dataset is a generic dataset used to describe any data stored in rows and columns, where the rows represent an example and the columns represent a feature (can be continuous or categorical). These datasets are commonly stored in CSV files, Pandas DataFrames, and in database tables. This guide will show you how to load and create a tabular dataset from:

- CSV files
- Pandas DataFrames
- HDF5 files
- Databases

CSV files

🤗 Datasets can read CSV files by specifying the generic `csv` dataset builder name in the load_dataset() method. To load more than one CSV file, pass them as a list to the `data_files` parameter:

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("csv", data_files="my_file.csv")

# load multiple CSV files
>>> dataset = load_dataset("csv", data_files=["my_file_1.csv", "my_file_2.csv", "my_file_3.csv"])
```

You can also map specific CSV files to the train and test splits:

```
>>> dataset = load_dataset("csv", data_files={"train": ["my_train_file_1.csv", "my_train_file_2.cs
```

To load remote CSV files, pass the URLs instead:

```
>>> base_url = "https://huggingface.co/datasets/lhoestq/demo1/resolve/main/data/"
>>> dataset = load_dataset('csv', data_files={"train": base_url + "train.csv", "test": base_url +
```

To load zipped CSV files:

```
>>> url = "https://domain.org/train_data.zip"
>>> data_files = {"train": url}
>>> dataset = load_dataset("csv", data_files=data_files)
```

Pandas DataFrames

🤗 Datasets also supports loading datasets from Pandas DataFrames with the from_pandas() method:

```
>>> from datasets import Dataset
>>> import pandas as pd

# create a Pandas DataFrame
>>> df = pd.read_csv("https://huggingface.co/datasets/imodels/credit-card/raw/main/train.csv")
>>> df = pd.DataFrame(df)
# load Dataset from Pandas DataFrame
>>> dataset = Dataset.from_pandas(df)
```

Use the `splits` parameter to specify the name of the dataset split:

```
>>> train_ds = Dataset.from_pandas(train_df, split="train")
>>> test_ds = Dataset.from_pandas(test_df, split="test")
```

If the dataset doesn't look as expected, you should explicitly specify your dataset features. A pandas.Series may not always carry enough information for Arrow to automatically infer a data type. For example, if a DataFrame is of length `0` or if the Series only contains `None/NaN` objects, the type is set to `null`.

HDF5 files

HDF5 files are commonly used for storing large amounts of numerical data in scientific computing and machine learning. Loading HDF5 files with 🤗 Datasets is similar to loading CSV files:

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("hdf5", data_files="data.h5")
```

Note that the HDF5 loader assumes that the file has "tabular" structure, i.e. that all datasets in the file have (the same number of) rows on their first dimension.

Databases

Datasets stored in databases are typically accessed with SQL queries. With 🤗 Datasets, you can connect to a database, query for the data you need, and create a dataset out of it. Then you can use all the processing features of 🤗 Datasets to prepare your dataset for training.

SQLite

SQLite is a small, lightweight database that is fast and easy to set up. You can use an existing database if you'd like, or follow along and start from scratch.

Start by creating a quick SQLite database with this Covid-19 data from the New York Times:

```
>>> import sqlite3
>>> import pandas as pd

>>> conn = sqlite3.connect("us_covid_data.db")
>>> df = pd.read_csv("https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv
>>> df.to_sql("states", conn, if_exists="replace")
```

This creates a `states` table in the `us_covid_data.db` database which you can now load into a dataset.

To connect to the database, you'll need the URI string that identifies your database. Connecting to a database with a URI caches the returned dataset. The URI string differs for each database dialect, so be sure to check the Database URLs for whichever database you're using.

For SQLite, it is:

```
>>> uri = "sqlite:///us_covid_data.db"
```

Load the table by passing the table name and URI to from_sql():

```
>>> from datasets import Dataset

>>> ds = Dataset.from_sql("states", uri)
>>> ds
Dataset({
    features: ['index', 'date', 'state', 'fips', 'cases', 'deaths'],
    num_rows: 54382
})
```

Then you can use all of 🤗 Datasets process features like filter() for example:

```
>>> ds.filter(lambda x: x["state"] == "California")
```

You can also load a dataset from a SQL query instead of an entire table, which is useful for querying and joining multiple tables.

Load the dataset by passing your query and URI to from_sql():

```
>>> from datasets import Dataset

>>> ds = Dataset.from_sql('SELECT * FROM states WHERE state="California";', uri)
>>> ds
Dataset({
    features: ['index', 'date', 'state', 'fips', 'cases', 'deaths'],
    num_rows: 1019
})
```

Then you can use all of 🤗 Datasets process features like filter() for example:

```
>>> ds.filter(lambda x: x["cases"] > 10000)
```

PostgreSQL

You can also connect and load a dataset from a PostgreSQL database, however we won't directly demonstrate how in the documentation because the example is only meant to be run in a notebook. Instead, take a look at how to install and setup a PostgreSQL server in this notebook!

After you've setup your PostgreSQL database, you can use the from_sql() method to load a dataset from a table or query.