Build and load

Nearly every deep learning workflow begins with loading a dataset, which makes it one of the most important steps. With 🤗 Datasets, there are more than 900 datasets available to help you get started with your NLP task. All you have to do is call: load_dataset() to take your first step. This function is a true workhorse in every sense because it builds and loads every dataset you use.

ELI5: `load_dataset`

Let's begin with a basic Explain Like I'm Five.

A dataset is a directory that contains:

- Some data files in generic formats (JSON, CSV, Parquet, text, etc.)
- A dataset card named `README.md` that contains documentation about the dataset as well as a YAML header to define the datasets tags and configurations

The load_dataset() function fetches the requested dataset locally or from the Hugging Face Hub.
The Hub is a central repository where all the Hugging Face datasets and models are stored.

If the dataset only contains data files, then load_dataset() automatically infers how to load the data files from their extensions (json, csv, parquet, txt, etc.).
Under the hood, 🤗 Datasets will use an appropriate DatasetBuilder based on the data files format. There exist one builder per data file format in 🤗 Datasets:

- datasets.packaged_modules.text.Text for text
- datasets.packaged_modules.csv.Csv for CSV and TSV
- datasets.packaged_modules.json.Json for JSON and JSONL
- datasets.packaged_modules.parquet.Parquet for Parquet
- datasets.packaged_modules.arrow.Arrow for Arrow (streaming file format)
- datasets.packaged_modules.sql.Sql for SQL databases
- datasets.packaged_modules.imagefolder.ImageFolder for image folders
- datasets.packaged_modules.audiofolder.AudioFolder for audio folders

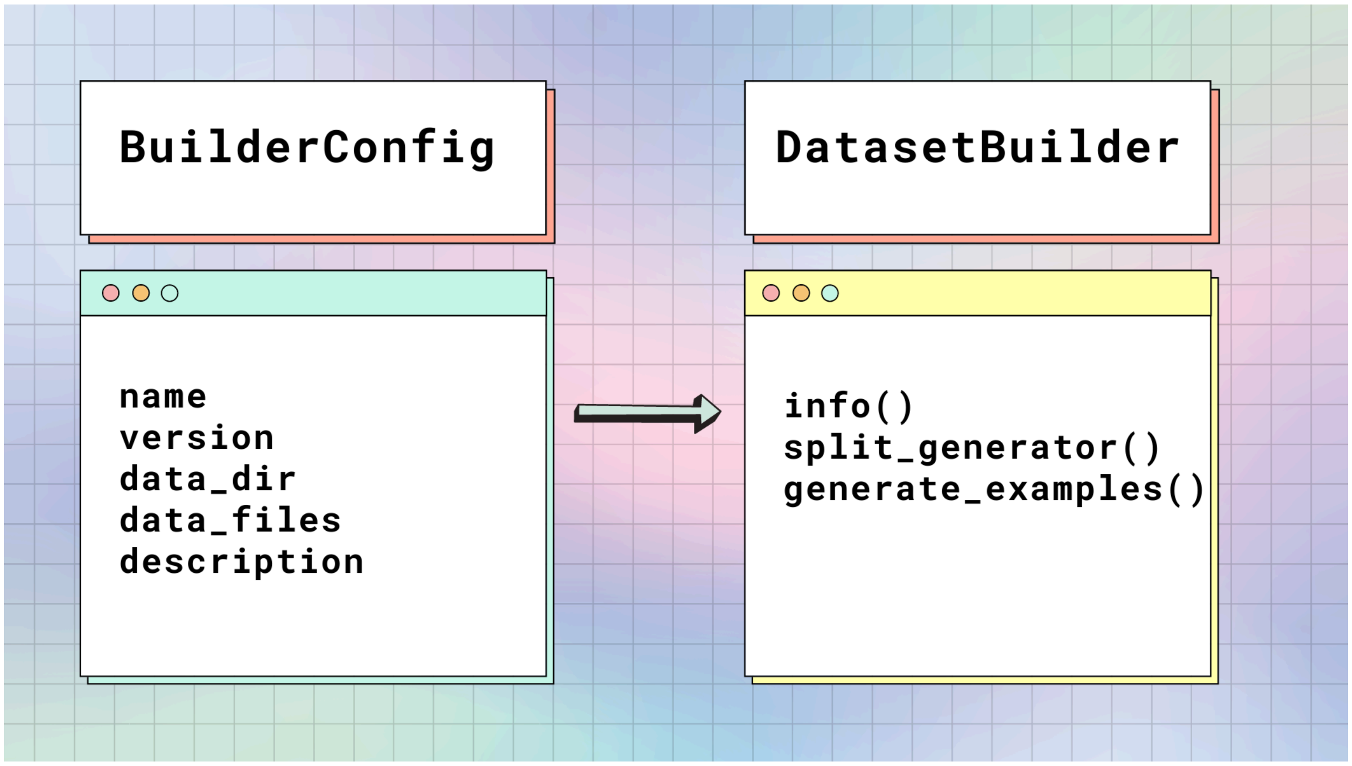
> [!TIP]
> Read the Share section to learn more about how to share a dataset.

🤗 Datasets downloads the dataset files from the original URL, generates the dataset and caches it in an Arrow table on your drive.
If you've downloaded the dataset before, then 🤗 Datasets will reload it from the cache to save you the trouble of downloading it again.

Now that you have a high-level understanding about how datasets are built, let's take a closer look at the nuts and bolts of how all this works.

Building a dataset

When you load a dataset for the first time, 🤗 Datasets takes the raw data file and builds it into a table of rows and typed columns. There are two main classes responsible for building a dataset: BuilderConfig and DatasetBuilder.



BuilderConfigdatasets-builderconfig

BuilderConfig is the configuration class of DatasetBuilder. The BuilderConfig contains the following basic attributes about a dataset:

Attribute	Description
`name`	Short name of the dataset.
`version`	Dataset version identifier.
`data_dir`	Stores the path to a local folder containing the data files.
`data_files`	Stores paths to local data files.
`description`	Description of the dataset.

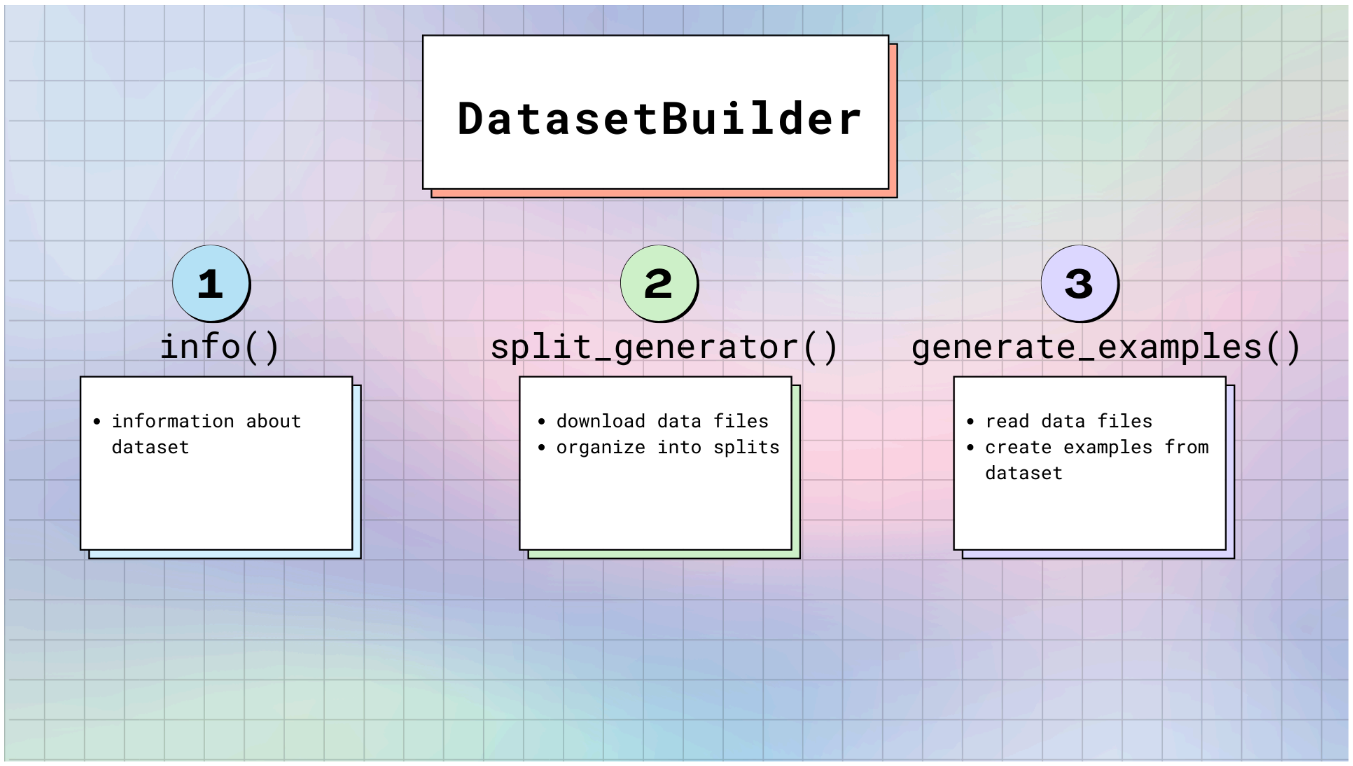
If you want to add additional attributes to your dataset such as the class labels, you can subclass the base BuilderConfig class. There are two ways to populate the attributes of a BuilderConfig class or subclass:

- Provide a list of predefined BuilderConfig class (or subclass) instances in the datasets `DatasetBuilder.BUILDER_CONFIGS()` attribute.
- When you call load_dataset(), any keyword arguments that are not specific to the method will be used to set the associated attributes of the BuilderConfig class. This will override the predefined attributes if a specific configuration was selected.

You can also set the DatasetBuilder.BUILDER_CONFIG_CLASS to any custom subclass of BuilderConfig.

DatasetBuilderdatasets-datasetbuilder

DatasetBuilder accesses all the attributes inside BuilderConfig to build the actual dataset.



There are three main methods in DatasetBuilder:

1. `DatasetBuilder._info()` is in charge of defining the dataset attributes. When you call `dataset.info`, 🤗 Datasets returns the information stored here. Likewise, the Features are also specified here. Remember, the Features are like the skeleton of the dataset. It provides the names and types of each column.
2. `DatasetBuilder._split_generator` downloads or retrieves the requested data files, organizes them into splits, and defines specific arguments for the generation process. This method has a DownloadManager that downloads files or fetches them from your local filesystem. Within the DownloadManager, there is a DownloadManager.download_and_extract() method that accepts a dictionary of URLs to the original data files, and downloads the requested files. Accepted inputs include: a single URL or path, or a list/dictionary of URLs or paths. Any compressed file types like TAR, GZIP and ZIP archives will be automatically extracted.
   Once the files are downloaded, SplitGenerator organizes them into splits. The SplitGenerator contains the name of the split, and any keyword arguments that are provided to the `DatasetBuilder._generate_examples` method. The keyword arguments can be specific to each split, and typically comprise at least the local path to the data files for each split.
3. `DatasetBuilder._generate_examples` reads and parses the data files for a split. Then it yields dataset examples according to the format specified in the `features` from

`DatasetBuilder._info()`. The input of `DatasetBuilder._generate_examples` is actually the `filepath` provided in the keyword arguments of the last method.
The dataset is generated with a Python generator, which doesn't load all the data in memory. As a result, the generator can handle large datasets. However, before the generated samples are flushed to the dataset file on disk, they are stored in an `ArrowWriter` buffer. This means the generated samples are written by batch. If your dataset samples consumes a lot of memory (images or videos), then make sure to specify a low value for the `DEFAULT_WRITER_BATCH_SIZE` attribute in DatasetBuilder. We recommend not exceeding a size of 200 MB.

Maintaining integrity

To ensure a dataset is complete, load_dataset() will perform a series of tests on the downloaded files to make sure everything is there. This way, you don't encounter any surprises when your requested dataset doesn't get generated as expected. load_dataset() verifies:

- The number of splits in the generated `DatasetDict`.
- The number of samples in each split of the generated `DatasetDict`.
- The list of downloaded files.
- The SHA256 checksums of the downloaded files (disabled by default).

If the dataset doesn't pass the verifications, it is likely that the dataset author made some changes in the data files.

In this case, an error is raised to alert that the dataset has changed.
To ignore the error, one needs to specify `verification_mode="no_checks"` in load_dataset().
Anytime you see a verification error, feel free to open a discussion or pull request in the corresponding dataset "Community" tab, so that the integrity checks for that dataset are updated.

Security

The dataset repositories on the Hub are scanned for malware, see more information here.