



# Quickstart

This quickstart is intended for developers who are ready to dive into the code and see an example of how to integrate 🤖 Datasets into their model training workflow. If you're a beginner, we recommend starting with our [tutorials](#), where you'll get a more thorough introduction.

Each dataset is unique, and depending on the task, some datasets may require additional steps to prepare it for training. But you can always use 🤖 Datasets tools to load and process a dataset. The fastest and easiest way to get started is by loading an existing dataset from the [Hugging Face Hub](#). There are thousands of datasets to choose from, spanning many tasks. Choose the type of dataset you want to work with, and let's get started!

## Audio

[Resample an audio dataset and get it ready for a model to classify what type of banking issue a speaker is calling about.](#)

## Vision

[Apply data augmentation to an image dataset and get it ready for a model to diagnose disease in bean plants.](#)

## NLP

[Tokenize a dataset and get it ready for a model to determine whether a pair of sentences have the same meaning.](#)

### [!TIP]

Check out [Chapter 5](#) of the Hugging Face course to learn more about other important topics such as loading remote or local datasets, tools for cleaning up a dataset, and creating your own dataset.

Start by installing 🤖 Datasets:

```
pip install datasets
```

🤖 Datasets also support audio and image data formats:

- To work with audio datasets, install the [Audio](#) feature:

```
pip install datasets[audio]
```

- To work with image datasets, install the [Image](#) feature:

```
pip install datasets[vision]
```

Besides 🧠 Datasets, make sure your preferred machine learning framework is installed:

```
```bash pip install torch ``` ```bash pip install tensorflow ```
```

## Audio

Audio datasets are loaded just like text datasets. However, an audio dataset is preprocessed a bit differently. Instead of a tokenizer, you'll need a [feature extractor](#). An audio input may also require resampling its sampling rate to match the sampling rate of the pretrained model you're using. In this quickstart, you'll prepare the [MInDS-14](#) dataset for a model train on and classify the banking issue a customer is having.

1. Load the MInDS-14 dataset by providing the [load\\_dataset\(\)](#) function with the dataset name, dataset configuration (not all datasets will have a configuration), and a dataset split:

```
>>> from datasets import load_dataset, Audio

>>> dataset = load_dataset("PolyAI/minds14", "en-US", split="train")
```

2. Next, load a pretrained [Wav2Vec2](#) model and its corresponding feature extractor from the 🧠 [Transformers](#) library. It is totally normal to see a warning after you load the model about some weights not being initialized. This is expected because you are loading this model checkpoint for training with another task.

```
>>> from transformers import AutoModelForAudioClassification, AutoFeatureExtractor

>>> model = AutoModelForAudioClassification.from_pretrained("facebook/wav2vec2-base")
>>> feature_extractor = AutoFeatureExtractor.from_pretrained("facebook/wav2vec2-base")
```

3. The [MInDS-14](#) dataset card indicates the sampling rate is 8kHz, but the Wav2Vec2 model

was pretrained on a sampling rate of 16kHz. You'll need to upsample the `audio` column with the `cast_column()` function and `Audio` feature to match the model's sampling rate.

```
>>> dataset = dataset.cast_column("audio", Audio(sampling_rate=16000))
>>> dataset[0]["audio"]
<datasets.features._torchcodec.AudioDecoder object at 0x11642b6a0>
```

4. Create a function to preprocess the audio `array` with the feature extractor, and truncate and pad the sequences into tidy rectangular tensors. The most important thing to remember is to call the audio `array` in the feature extractor since the `array` - the actual speech signal - is the model input.

Once you have a preprocessing function, use the `map()` function to speed up processing by applying the function to batches of examples in the dataset.

```
>>> def preprocess_function(examples):
...     audio_arrays = [x.get_all_samples().data for x in examples["audio"]]
...     inputs = feature_extractor(
...         audio_arrays,
...         sampling_rate=16000,
...         padding=True,
...         max_length=100000,
...         truncation=True,
...     )
...     return inputs

>>> dataset = dataset.map(preprocess_function, batched=True)
```

5. Use the `rename_column()` function to rename the `intent_class` column to `labels`, which is the expected input name in `Wav2Vec2ForSequenceClassification`:

```
>>> dataset = dataset.rename_column("intent_class", "labels")
```

6. Set the dataset format according to the machine learning framework you're using.

Use the `[set_format()](/docs/datasets/v4.2.0/en/package_reference/main_classes#datasets.Dataset.set_format)` function to set the dataset format to `'torch'` and

specify the columns you want to format. This function applies formatting on-the-fly. After converting to PyTorch tensors, wrap the dataset in `[`torch.utils.data.DataLoader`](https://albion.github.io/doc_view/data.html?highlight=torch%20utils%20data%20dataloader#torch.utils.data.DataLoader):`

```
>>> from torch.utils.data import DataLoader

>>> dataset.set_format(type="torch", columns=["input_values", "labels"])
>>> dataloader = DataLoader(dataset, batch_size=4)
```

Use the `prepare_tf_dataset` method from 🤗 Transformers to prepare the dataset to be compatible with TensorFlow, and ready to train/fine-tune a model, as it wraps a HuggingFace [Dataset](#) as a `tf.data.Dataset` with collation and batching, so one can pass it directly to Keras methods like `fit()` without further modification.

```
>>> import tensorflow as tf

>>> tf_dataset = model.prepare_tf_dataset(
...     dataset,
...     batch_size=4,
...     shuffle=True,
... )
```

7. Start training with your machine learning framework! Check out the 🤗 Transformers [audio classification guide](#) for an end-to-end example of how to train a model on an audio dataset.

## Vision

Image datasets are loaded just like text datasets. However, instead of a tokenizer, you'll need a [feature extractor](#) to preprocess the dataset. Applying data augmentation to an image is common in computer vision to make the model more robust against overfitting. You're free to use any data augmentation library you want, and then you can apply the augmentations with 🤗 Datasets. In this quickstart, you'll load the [Beans](#) dataset and get it ready for the model to train on and identify disease from the leaf images.

1. Load the Beans dataset by providing the `load_dataset()` function with the dataset name and a dataset split:

```
>>> from datasets import load_dataset, Image

>>> dataset = load_dataset("AI-Lab-Makerere/beans", split="train")
```

Most image models work with RGB images. If your dataset contains images in a different mode, you can use the `cast_column()` function to set the mode to RGB:

```
>>> dataset = dataset.cast_column("image", Image(mode="RGB"))
```

The Beans dataset contains only RGB images, so this step is unnecessary here.

2. Now you can add some data augmentations with any library ([Albumentations](#), [imgaug](#), [Kornia](#)) you like. Here, you'll use `torchvision` to randomly change the color properties of an image:

```
>>> from torchvision.transforms import Compose, ColorJitter, ToTensor

>>> jitter = Compose(
...     [ColorJitter(brightness=0.5, hue=0.5), ToTensor()]
... )
```

3. Create a function to apply your transform to the dataset and generate the model input: `pixel_values`.

```
>>> def transforms(examples):
...     examples["pixel_values"] = [jitter(image.convert("RGB")) for image in examples["image"]]
...     return examples
```

4. Use the `with_transform()` function to apply the data augmentations on-the-fly:


```
>>> dataset = dataset.with_transform(transforms)
```

5. Set the dataset format according to the machine learning framework you're using.

Wrap the dataset in [`torch.utils.data.DataLoader`](https://albion.github.io/doc\_view/data.html?highlight=torch%20utils%20data%20dataloader#torch.utils.data.DataLoader). You'll also need to create a collate function to collate the samples into batches:

```
>>> from torch.utils.data import DataLoader

>>> def collate_fn(examples):
...     images = []
...     labels = []
...     for example in examples:
...         images.append((example["pixel_values"]))
...         labels.append(example["labels"])
...
...     pixel_values = torch.stack(images)
...     labels = torch.tensor(labels)
...     return {"pixel_values": pixel_values, "labels": labels}
>>> dataloader = DataLoader(dataset, collate_fn=collate_fn, batch_size=4)
```

Use the `prepare_tf_dataset` method from  Transformers to prepare the dataset to be compatible with TensorFlow, and ready to train/fine-tune a model, as it wraps a HuggingFace `Dataset` as a `tf.data.Dataset` with collation and batching, so one can pass it directly to Keras methods like `fit()` without further modification.

Before you start, make sure you have up-to-date versions of `albumentations` and `cv2` installed:

```
pip install -U albumentations opencv-python
```

```

>>> import albumentations
>>> import numpy as np

>>> transform = albumentations.Compose([
...     albumentations.RandomCrop(width=256, height=256),
...     albumentations.HorizontalFlip(p=0.5),
...     albumentations.RandomBrightnessContrast(p=0.2),
... ])

>>> def transforms(examples):
...     examples["pixel_values"] = [
...         transform(image=np.array(image))["image"] for image in examples["image"]
...     ]
...     return examples

>>> dataset.set_transform(transforms)
>>> tf_dataset = model.prepare_tf_dataset(
...     dataset,
...     batch_size=4,
...     shuffle=True,
... )

```

6. Start training with your machine learning framework! Check out the 🧑🏫 Transformers [image classification guide](#) for an end-to-end example of how to train a model on an image dataset.

## NLP

Text needs to be tokenized into individual tokens by a [tokenizer](#). For the quickstart, you'll load the [Microsoft Research Paraphrase Corpus \(MRPC\)](#) training dataset to train a model to determine whether a pair of sentences mean the same thing.


1. Load the MRPC dataset by providing the [load\\_dataset\(\)](#) function with the dataset name, dataset configuration (not all datasets will have a configuration), and dataset split:

```

>>> from datasets import load_dataset

>>> dataset = load_dataset("nyu-mll/glue", "mrpc", split="train")

```

**2.** Next, load a pretrained **BERT** model and its corresponding tokenizer from the  **Transformers** library. It is totally normal to see a warning after you load the model about some weights not being initialized. This is expected because you are loading this model checkpoint for training with another task.

```
>>> from transformers import AutoModelForSequenceClassification, AutoTokenizer

>>> model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased")
>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

===PT-TF-SPLIT===

>>> from transformers import TFAutoModelForSequenceClassification, AutoTokenizer

>>> model = TFAutoModelForSequenceClassification.from_pretrained("bert-base-uncased")
>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

**3.** Create a function to tokenize the dataset, and you should also truncate and pad the text into tidy rectangular tensors. The tokenizer generates three new columns in the dataset:

Use the `map()` function to speed up processing by applying your tokenization function to batches of examples in the dataset:

[illegible]

4. Rename the `label` column to `labels`, which is the expected input name in `BertForSequenceClassification`:



```
>>> dataset = dataset.map(lambda examples: {"labels": examples["label"]}, batched=True)
```

5. Set the dataset format according to the machine learning framework you're using.

Use the `[with_format()](/docs/datasets/v4.2.0/en/package_reference/main_classes#datasets.Dataset.with_format)` function to set the dataset format to `torch` and specify the columns you want to format. This function applies formatting on-the-fly. After converting to PyTorch tensors, wrap the dataset in `[torch.utils.data.DataLoader](https://albion.github.io/doc_view/data.html?highlight=torch%20utils%20data%20dataloader#torch.utils.data.DataLoader)`:

```
>>> import torch

>>> dataset = dataset.select_columns(["input_ids", "token_type_ids", "attention_mask", "labels"])
>>> dataset = dataset.with_format(type="torch")
>>> dataloader = torch.utils.data.DataLoader(dataset, batch_size=32)
```

Use the `prepare_tf_dataset` method from 🤗 Transformers to prepare the dataset to be compatible with TensorFlow, and ready to train/fine-tune a model, as it wraps a HuggingFace `Dataset` as a `tf.data.Dataset` with collation and batching, so one can pass it directly to Keras methods like `fit()` without further modification.

```
>>> import tensorflow as tf

>>> tf_dataset = model.prepare_tf_dataset(
...     dataset,
...     batch_size=4,
...     shuffle=True,
... )
```

6. Start training with your machine learning framework! Check out the 🤗 Transformers [text classification guide](#) for an end-to-end example of how to train a model on a text dataset.

# What's next?

This completes the 🧑🏻 Datasets quickstart! You can load any text, audio, or image dataset with a single function and get it ready for your model to train on.

For your next steps, take a look at our [How-to guides](#) and learn how to do more specific things like loading different dataset formats, aligning labels, and streaming large datasets. If you're interested in learning more about 🧑🏻 Datasets core concepts, grab a cup of coffee and read our [Conceptual Guides](#)!