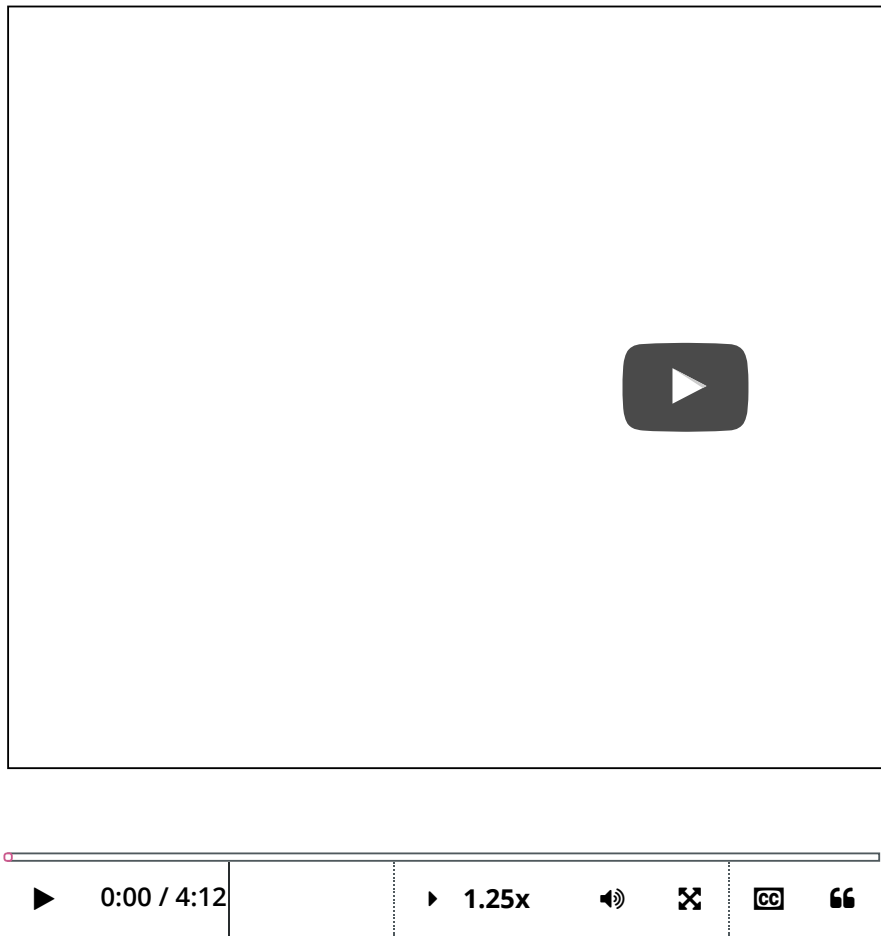
Video

Bookmark this page

Wrangling Your Data



Start of transcript. Skip to the end.

It's not uncommon to encounter data sets that

are missing some values.

The information may have been redacted.

Maybe it wasn't collected due to human error.

There may have been a mechanical failure or

it may really never have even existed in the first place.

Regardless of the cause, you

Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

It might not have been collected by the person placed in charge of doing so. There might have been a mechanical failure at the point of the sensor. Perhaps the dog ate it. Or maybe it never really even existed in the first place! Whatever the cause, it's not uncommon for datasets to come with some missing data.

When you are working with large datasets, it would be great if every sample had measurements recorded for each feature. But in reality, this almost never happens. In fact, you might not even find a single sample free of missing data. Annoying as this is, simply ignoring missing data usually isn't an option, as it can wreck havoc if not handled properly during your analysis. If not accounted for, missing data might lead you to erroneous conclusions about your samples by resulting in incorrect sums and means, and even by skewing distributions.

Pandas represents missing data internally using Numpy's `np.nan`. Had Python's `None` been used, there would be ambiguous collision cases when you actually *wished* to store `None` and could no longer differentiate that and a missing record. Pandas provides you with a few basic methods for mitigating missing data, which work on both series and dataframe objects.

Any time a nan is encountered, replace it with a scalar value:

```
df.my_feature.fillna( df.my_feature.mean() )
df.fillna(0)
```

When a nan is encountered, replace it with the immediate, previous, non-nan value. Be mindful about which axis you perform this on. You also have the abilities to specify an optional limit of how far you want the fill to go, or if you'd like to run the fill in reverse (bfill):

```
df.fillna(method='ffill')  # fill the values forward
df.fillna(method='bfill')  # fill the values in reverse
df.fillna(limit=5)
```

Fill out nans by interpolating over them with the non-nan values that come immediately before and after. You can select the interpolation method you'd like to use, such as nearest, cubic, spline and more. If your nans occur at the start or end of your list, interpolation will not be able to help you:

```
df.interpolate(method='polynomial', order=2)
```


English ▾