Pandas: 6th lesson – Data Types & Missing Values

Dtypes:

* The data type for a column in a DataFrame or a Series is known as the dtype. You can use the dtype property to grab the type of a specific column. For instance, we can get the dtype of the price column in the reviews DataFrame:

import pandas as pd

reviews = pd.read\_csv("../input/wine-reviews/winemag-data-130k-v2.csv", index\_col=0)

pd.set\_option('display.max\_rows', 5)

reviews.price.dtype

dtype('float64')

* Alternatively, the dtypes property returns the dtype of every column in the DataFrame:

reviews.dtypes

country object

description object

...

variety object

winery object

Length: 13, dtype: object

* Data types tell us something about how pandas is storing the data internally. float64 means that it's using a 64-bit floating point number; int64 means a similarly sized integer instead, and so on. One peculiarity to keep in mind (and on display very clearly here) is that columns consisting entirely of strings do not get their own type; they are instead given the object type. It's possible to convert a column of one type into another wherever such a conversion makes sense by using the astype() function. For example, we may transform the points column from its existing int64 data type into a float64 data type:

reviews.points.astype('float64')

0 87.0

1 87.0

...

129969 90.0

129970 90.0

Name: points, Length: 129971, dtype: float64

* A DataFrame or Series index has its own dtype, too:

reviews.index.dtype

dtype('int64')

Pandas also supports more exotic data types, such as categorical data and time series data. Because these data types are more rarely used, we will omit them until a much later section of this tutorial.

Missing values:

* Entries missing values are given the value NaN, short for "Not a Number". For technical reasons these NaN values are always of the float64 dtype.
* Pandas provides some methods specific to missing data. To select NaN entries you can use pd.isnull() (or its companion pd.notnull()). This is meant to be used thusly:

reviews[pd.isnull(reviews.country)]

* Replacing missing values is a common operation. Pandas provides a really handy method for this problem: fillna(). fillna() provides a few different strategies for mitigating such data. For example, we can simply replace each NaN with an "Unknown":

reviews.region\_2.fillna("Unknown")

0 Unknown

1 Unknown

...

129969 Unknown

129970 Unknown

Name: region\_2, Length: 129971, dtype: object

Or we could fill each missing value with the first non-null value that appears sometime after the given record in the database. This is known as the backfill strategy.

* Alternatively, we may have a non-null value that we would like to replace. For example, suppose that since this dataset was published, reviewer Kerin O'Keefe has changed her Twitter handle from @kerinokeefe to @kerino. One way to reflect this in the dataset is using the replace() method:

reviews.taster\_twitter\_handle.replace("@kerinokeefe", "@kerino")

0 @kerino

1 @vossroger

...

129969 @vossroger

129970 @vossroger

Name: taster\_twitter\_handle, Length: 129971, dtype: object

The replace() method is worth mentioning here because it's handy for replacing missing data which is given some kind of sentinel value in the dataset: things like "Unknown", "Undisclosed", "Invalid", and so on.