Introduction

In this micro-course, you'll learn all about **pandas**, the most popular Python library for data analysis.

Along the way, you'll complete several hands-on exercises with real-world data. We recommend that you work on the exercises while reading the corresponding tutorials.

To start the first exercise, please click here.

In this tutorial, you will learn how to create your own data, along with how to work with data that already exists.

Getting started

To use pandas, you'll typically start with the following line of code.

```
import pandas as pd
```

Creating data

There are two core objects in pandas: the **DataFrame** and the **Series**.

DataFrame

A DataFrame is a table. It contains an array of individual *entries*, each of which has a certain *value*. Each entry corresponds to a row (or *record*) and a *column*.

For example, consider the following simple DataFrame:

```
pd.DataFrame({'Yes': [50, 21], 'No': [131, 2]})
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Yes	No
0	50	131
1	21	2

In this example, the "0, No" entry has the value of 131. The "0, Yes" entry has a value of 50, and so on.

DataFrame entries are not limited to integers. For instance, here's a DataFrame whose values are strings:

```
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']})
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Bob	Sue
0	I liked it.	Pretty good.
1	It was awful.	Bland.

We are using the `pd.DataFrame()` constructor to generate these DataFrame objects. The syntax for declaring a new one is a dictionary whose keys are the column names (`Bob` and `Sue` in this example), and whose values are a list of entries. This is the standard way of constructing a new DataFrame, and the one you are most likely to encounter.

The dictionary-list constructor assigns values to the *column labels*, but just uses an ascending count from 0 (0, 1, 2, 3, ...) for the *row labels*. Sometimes this is OK, but oftentimes we will want to assign these labels ourselves.

The list of row labels used in a DataFrame is known as an **Index**. We can assign values to it by using an `index` parameter in our constructor:

```
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],
              'Sue': ['Pretty good.', 'Bland.']},
             index=['Product A', 'Product B'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Bob	Sue
Product A	I liked it.	Pretty good.
Product B	It was awful.	Bland.

Series

A Series, by contrast, is a sequence of data values. If a DataFrame is a table, a Series is a list. And in fact you can create one with nothing more than a list:

```
pd.Series([1, 2, 3, 4, 5])
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

A Series is, in essence, a single column of a DataFrame. So you can assign row labels to the Series the same way as before, using an `index` parameter. However, a Series does not have a column name, it only has one overall `name`:

```
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], name='Product A')
```

```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```

The Series and the DataFrame are intimately related. It's helpful to think of a DataFrame as actually being just a bunch of Series "glued together". We'll see more of this in the next section of this tutorial.

Reading data files

Being able to create a DataFrame or Series by hand is handy. But, most of the time, we won't actually be creating our own data by hand. Instead, we'll be working with data that already exists.

Data can be stored in any of a number of different forms and formats. By far the most basic of these is the humble CSV file. When you open a CSV file you get something that looks like this:

```
Product A,Product B,Product C,
30,21,9,
35,34,1,
41,11,11
```

So a CSV file is a table of values separated by commas. Hence the name: "Comma-Separated Values", or CSV.

Let's now set aside our toy datasets and see what a real dataset looks like when we read it into a DataFrame. We'll use the `pd.read_csv()` function to read the data into a DataFrame. This goes thusly:

```
wine_reviews = pd.read_csv("../input/wine-reviews/winemag-data-130k-v2.csv")
```

We can use the `shape` attribute to check how large the resulting DataFrame is:

```
wine_reviews.shape
```

```
(129971, 14)
```

So our new DataFrame has 130,000 records split across 14 different columns. That's almost 2 million entries!

We can examine the contents of the resultant DataFrame using the `head()` command, which grabs the first five rows:

```
wine_reviews.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Unnamed: 0	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_ha
0	0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe
1	1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger
2	2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine
3	3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN
4	4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine

The `pd.read_csv()` function is well-endowed, with over 30 optional parameters you can specify. For example, you can see in this dataset that the CSV file has a built-in index, which pandas did not pick up on automatically. To make pandas use that column for the index (instead of creating a new one from scratch), we can specify an `index_col`.

```
wine_reviews = pd.read_csv("../input/wine-reviews/winemag-data-130k-v2.csv", index_col=0)
wine_reviews.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 V Bianco (Etna)
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta Avidag 2011 Avidag Red (Douro
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainsto 2013 P Gris (Willam Valley)
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN	St. Julia 2013 Reserv Late Harves Riesling
4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Sweet Cheeks 2012 Vintner Reserv Wild Child...

Your turn

If you haven't started the exercise, you can **get started here**.

Have questions or comments? Visit the course discussion forum to chat with other learners.