Tom Jeon
May 9th, 2018

MUST READ PANDAS +1

Pandas Tutorial: Importing Data with read_csv()

Importing data is the first step in any data science project. Learn why today's data scientists prefer pandas' read_csv() function to do this.

Pandas Tutorial: Importing Data with read_csv()

The first step to any data science project is to import your data. Often, you'll work with data in Comma Separated Value (CSV) files and run into problems at the very start of your workflow. In this tutorial, you'll see how you can use the `read_csv()` function from `pandas` to deal with common problems when importing data and see why loading CSV files specifically with `pandas` has become standard practice for working data scientists today.

The filesystem

Before you can use `pandas` to import your data, you need to know where your data is in your filesystem and what your current working directory is. You'll see why this is important very soon, but let's review some basic concepts:

Everything on the computer is stored in the filesystem. "Directories" is just another word for "folders", and the "working directory" is simply the folder you're currently in. The Introduction to Shell for Data Science course on DataCamp will give you a full, hands-on experience with its utility, but here are some basic Shell commands to navigate your way in

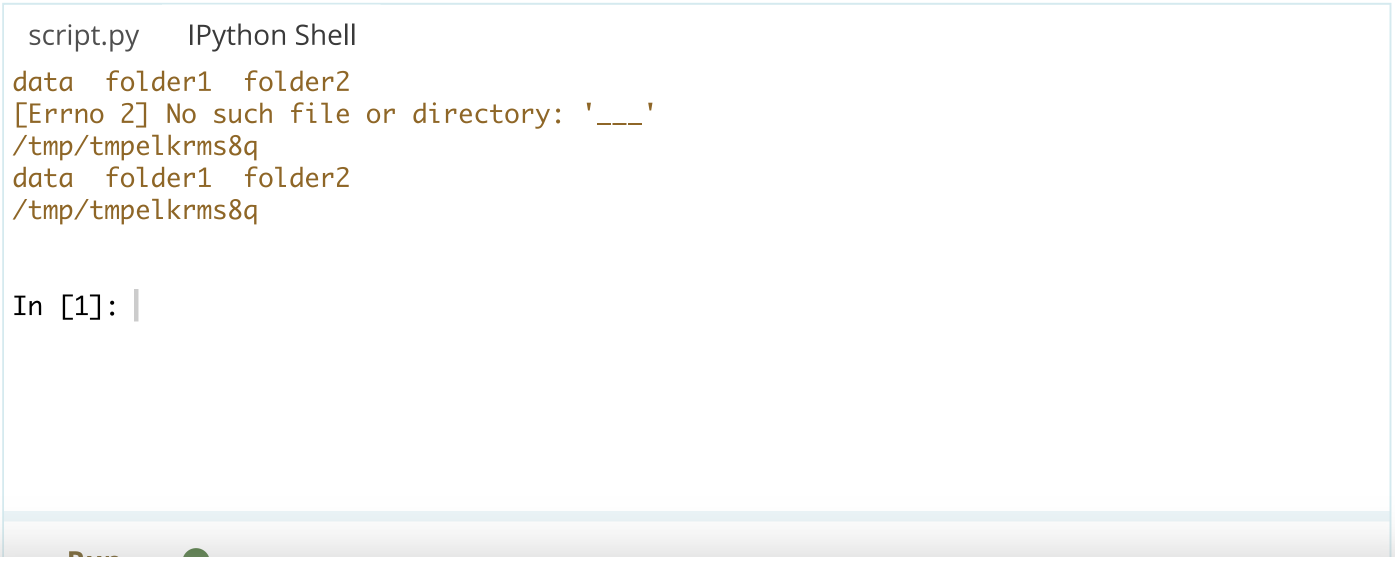
Want to leave a comment?

- The `cd` command followed by:
  - the name of a sub-directory allows you to change your working directory to the sub-directory you specify.
  - `..` allows you to navigate back to the parent directory of your current working directory.
- The `pwd` command prints the path of your current working directory.

IPython allows you to execute Shell commands directly from the IPython console via its magic commands. Here are the ones that correspond to the commands you saw above:

- `! ls` in IPython is the same as `ls` in the command line.
- `%cd` in IPython is the same as `cd` in the command line.
- `! pwd` in IPython is the same as `pwd` in the command line. The working directory is also printed after changing into it in IPython, which isn't the case in the command line.

In your filesystem, there's a file called `cereal.csv` that contains nutrition data on 80 cereals. Enter the magic commands one-by-one in the IPython Shell, and see if you can locate the dataset!



```
data  folder1  folder2
[Errno 2] No such file or directory: '___'
/tmp/tmpelkrms8q
data  folder1  folder2
/tmp/tmpelkrms8q


In [1]:
```

Want to leave a comment?

Now that you know what your current working directory is and where the dataset is in your filesystem, you can specify the file path to it. You're now ready to import the CSV file into Python using `read_csv()` from `pandas` :

```
import pandas as pd
cereal_df = pd.read_csv("/tmp/tmp07wuam09/data/cereal.csv")
cereal_df2 = pd.read_csv("data/cereal.csv")

# Are they the same?
print(pd.DataFrame.equals(cereal_df, cereal_df2))
```

```
True
```

As you can see in the code chunk above, the file path is the main argument to `read_csv()` and it was specified in two ways. You can use the full file path which is prefixed by a `/` and includes the working directory in the specification, or use the relative file path which doesn't. The `read_csv()` function is smart enough to decipher whether it's working with full or relative file paths and convert your flat file as a DataFrame without a problem. (**Note**: the environment for every DataCamp session is temporary, so the working directory you saw in the previous section may not be identical to the one you see in the code chunk above.)

Continue on and see how else `pandas` makes importing CSV files easier. Let's use some of the function's customizable options, particularly for the way it deals with headers, incorrect data types, and missing data.

Dealing with headers

Headers refer to the column names of your dataset. For some datasets you might encounter, the headers may be completely missing, partially missing, or they might exist, but you may want to rename them. How can you deal with such issues effectively with `pandas` ?

Let's take a closer look at your data:

Want to leave a comment?

```
print(df.head(5))
```

```
                          X.1      X.2      X.3       X.4      X.5      X.6  \
0                        name      mfr     type  calories  protein      fat
1                   100% Bran        N        C        70        4        1
2           100% Natural Bran        Q  no info       120        3        5
3                    All-Bran  no info        C        70        4        1
4   All-Bran with Extra Fiber        K        C        50        4  no info

      X.7    X.8      X.9    X.10     X.11      X.12   X.13     X.14  X.15  \
0  sodium  fiber    carbo  sugars   potass  vitamins  shelf   weight  cups
1       .     10  no info       6      280        25      3        1  0.33
2      15      2        8       8      135         0      .        1     1
3     260      9        7       5  no info        25      3        1  0.33
4     140     14        8       0      330        25      3  no info   0.5

        X.16
0     rating
1  68.402973
2    no info
3  59.425505
4  93.704912
```

It seems like more sensible columns names would be `name` , `mfr` , ..., `rating` , but they're incorrectly imported as the first observation in the dataset! Let's try to have these as the headers. The `read_csv()` function has an argument called `skiprows` that allows you to specify the number of lines to skip at the start of the file. In this case, you want to skip the first line, so let's try importing your CSV file with `skiprows` set equal to 1:

```
df = pd.read_csv("data/cereal.csv", skiprows = 1)
print(df.head(5))
```

Want to leave a comment?

```
3  All-Bran with Extra Fiber        K        C       50       4  no info
4              Almond Delight        R        C      110       2        2

  sodium  fiber    carbo  sugars   potass vitamins shelf   weight  cups  \
0      .    10.0  no info       6      280       25     3        1  0.33
1     15     2.0        8       8      135        0     .        1  1.00
2    260     9.0        7       5  no info       25     3        1  0.33
3    140    14.0        8       0      330       25     3  no info  0.50
4    200     1.0       14       8       -1        .     3        1  0.75

      rating
0  68.402973
1    no info
2  59.425505
3  93.704912
4  34.384843
```

Nice!

Even when you don't specify the headers, the `read_csv()` function correctly infers that the first observation contains the headers for the dataset. Not only that, `read_csv()` can infer the data types for each column of your dataset as well. You can see below the `calories` column is an integer column, whereas the `fiber` column is a float column:

```
print(df['calories'].dtypes)
print(df['fiber'].dtypes)
```

```
int64
float64
```

Dealing with missing values and incorrect data types

Want to leave a comment?

```
print(df['fat'].dtypes)
```

```
object
```

When the column's data type is an `object`, doing simple arithmetic results in unexpected results. This sort of behavior can be problematic when doing all sorts of tasks—visualizing distributions, finding outliers, training models—because you expect Python to treat numbers as numbers.

Run the code below to see an example of this. Your dataset has been loaded as `df`.

script.py IPython Shell

```python
# Print the first value in the `fat` column
print(df['fat'][0])

# Print the second value in the `fat` column
print(df['fat'][1])

# Print the sum
print(df['fat'][0] + df['fat'][1])
```

Run

But 1 + 5 is not 15!

Ideally, the `fat` column should be treated as type `int64` or `float64`, and missing data should be encoded as `NaN` so that you can apply statistics in a missing-value-friendly manner. Instead of parsing through each column and replacing `'no info'` and `'.'` with `NaN` values after the dataset is loaded, you can use the `na_values` argument to account for

Want to leave a comment?

```
print(df.head(5))
```

```
                        name  mfr type  calories  protein  fat  sodium  fiber  \
0                  100% Bran    N    C        70      4.0  1.0     NaN   10.0
1          100% Natural Bran    Q  NaN       120      3.0  5.0    15.0    2.0
2                   All-Bran  NaN    C        70      4.0  1.0   260.0    9.0
3  All-Bran with Extra Fiber    K    C        50      4.0  NaN   140.0   14.0
4             Almond Delight    R    C       110      2.0  2.0   200.0    1.0

   carbo  sugars  potass  vitamins  shelf  weight  cups     rating
0    NaN       6   280.0      25.0    3.0     1.0  0.33  68.402973
1    8.0       8   135.0       0.0    NaN     1.0  1.00        NaN
2    7.0       5     NaN      25.0    3.0     1.0  0.33  59.425505
3    8.0       0   330.0      25.0    3.0     NaN  0.50  93.704912
4   14.0       8    -1.0       NaN    3.0     1.0  0.75  34.384843
```

Now try the same arithmetic you saw a moment ago:

script.py   IPython Shell

```python
# Print the first value in the `fat` column
print(df['fat'][0])

# Print the second value in the `fat` column
print(df['fat'][1])

# Print the sum
print(df['fat'][0] + df['fat'][1])
```

Run

Awesome. 1 + 5 is indeed 6. The values in the `fat` column are now treated as numerics.

Want to leave a comment?

- Located the CSV file you want to import from your filesystem.
- Corrected the headers of your dataset.
- Dealt with missing values so that they're encoded properly as `NaN` s.
- Corrected data types for every column in your dataset.
- Converted a CSV file to a Pandas DataFrame (see why that's important in this Pandas tutorial).

Final thoughts

Although the CSV file is one of the most common formats for storing data, there are other file types that the modern-day data scientist must be familiar with. You now have a good sense of how useful `pandas` is when importing the CSV file, and conveniently, `pandas` offers other similar and equally handy functions to import Excel, SAS, and Stata files to name a few.

Yet, due to the active community in open source software, there is constant activity in file formats and ways to import data. Lots of useful, high quality datasets are hosted on the web and accessed through APIs, for example. If you're curious and want to know the state of the art, DataCamp's Importing Data in Python (Part 1) and Importing Data in Python (Part 2) courses will teach you all the best practices.

Happy Learning!

▲ 58

11

COMMENTS

Want to leave a comment?

▲ 12 REPLY

Gonzalo Maldonado
04/06/2018 06:08 PM
great! all that i was looking for :)

▲ 5 REPLY

Vo Hac
07/07/2018 08:41 PM
Thank you :)

▲ 3 REPLY

Antonio Cachuan
31/07/2018 11:30 AM
Useful article

▲ 3 REPLY

Amit Bhonsle
27/10/2018 06:35 AM
I have a CSV file for word bank population data, I am trying tor read the csv file using pandas read_csv method and getting this error: ' UnicodeDecodeError: 'utf-8' codec can't decode byte 0x92 in position 18: invalid start byte'. what is it?

▲ 3 REPLY

Tom Jeon
03/11/2018 09:38 PM
Hey Amit, the `encoding` argument to the `read_csv()` function may be what you need. Check out this post if you haven't already!

▲ 2 REPLY

Want to leave a comment?

crazy. Thank you!

Also anyone using Mojave on Mac may find it's harder to copy a file's path. Use option + command + c to grab it.

▲ 2 REPLY

Mohammad Zafari
14/11/2018 07:57 AM
What if the header is multiple lines and you like to keep all the info as header?

something like this:

name , type , fat , fat, sodium

 , ,type1, type2,

100% Bran, C, 1, 1.5, 10

All-Bran, C, 2, 1, 12

▲ 1 REPLY

Neha D Shakya
07/12/2018 05:16 PM
I keep getting this error

AttributeError: module 'pandas' has no attribute 'read_cvs'

I have tried reinstalling pandas but I still get this error

▲ 3 REPLY

---

bruno giscard siewe
19/01/2019 07:17 AM
yes

▲ 1 REPLY

Want to leave a comment?

▲ 1 REPLY

Want to leave a comment?