Aleksey Bilogur

Creating, reading, and writing reference

Notebook

Creating, reading, and writing reference

This is the reference component to the "Creating, reading, and writing" section of the tutorial. For the workbook section, click here.

The very first step in any data analytics project will probably reading the data out of a file somewhere, so it makes sense that that's the first thing we'd need to cover. In this section, we'll look at exercises on creating `pandas` `Series` and `DataFrame` objects, both by hand and by reading data from disc.

The IO Tools section of the official `pandas` docs provides a comprehensive overview on this subject.

In [1]:
```
import pandas as pd
```

Creating data

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

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

For example, consider the following simple `DataFrame`:

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

Out[2]:

	No	Yes
0	131	50
1	2	21

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 `str` strings:

In [3]:
```
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']})
```

Out[3]:

	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 likliest 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:

In [4]:
```
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],
             'Sue': ['Pretty good.', 'Bland.']},
            index=['Product A', 'Product B'])
```

Out[4]:

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

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:

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

Out[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 column values to the `Series` the same way as before, using an `index` parameter. However, a `Series` do not have a column name, it only has one overall `name`:

In [6]:
```
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], name='Product A')
```

Out[6]:
```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```

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

Reading common file formats

Being able to create a `DataFrame` and `Series` by hand is handy. But, most of the time, we won't actually be creating our own data by hand, 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:

```
csv
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-seperated 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 `read_csv` function to read the data into a `DataFrame`. This goes thusly:

In [7]:
```
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:

In [8]:
```
wine_reviews.shape
```

Out[8]:
```
(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:

In [9]:
```
wine_reviews.head()
```

Out[9]:

	Unnamed: 0	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_
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 `pandas` `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 an in-built 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 may specify and use an `index_col`.

In [10]:
```
wine_reviews = pd.read_csv("../input/wine-reviews/winemag-data-130k-v2.csv", index_col=0)
wine_reviews.head()
```

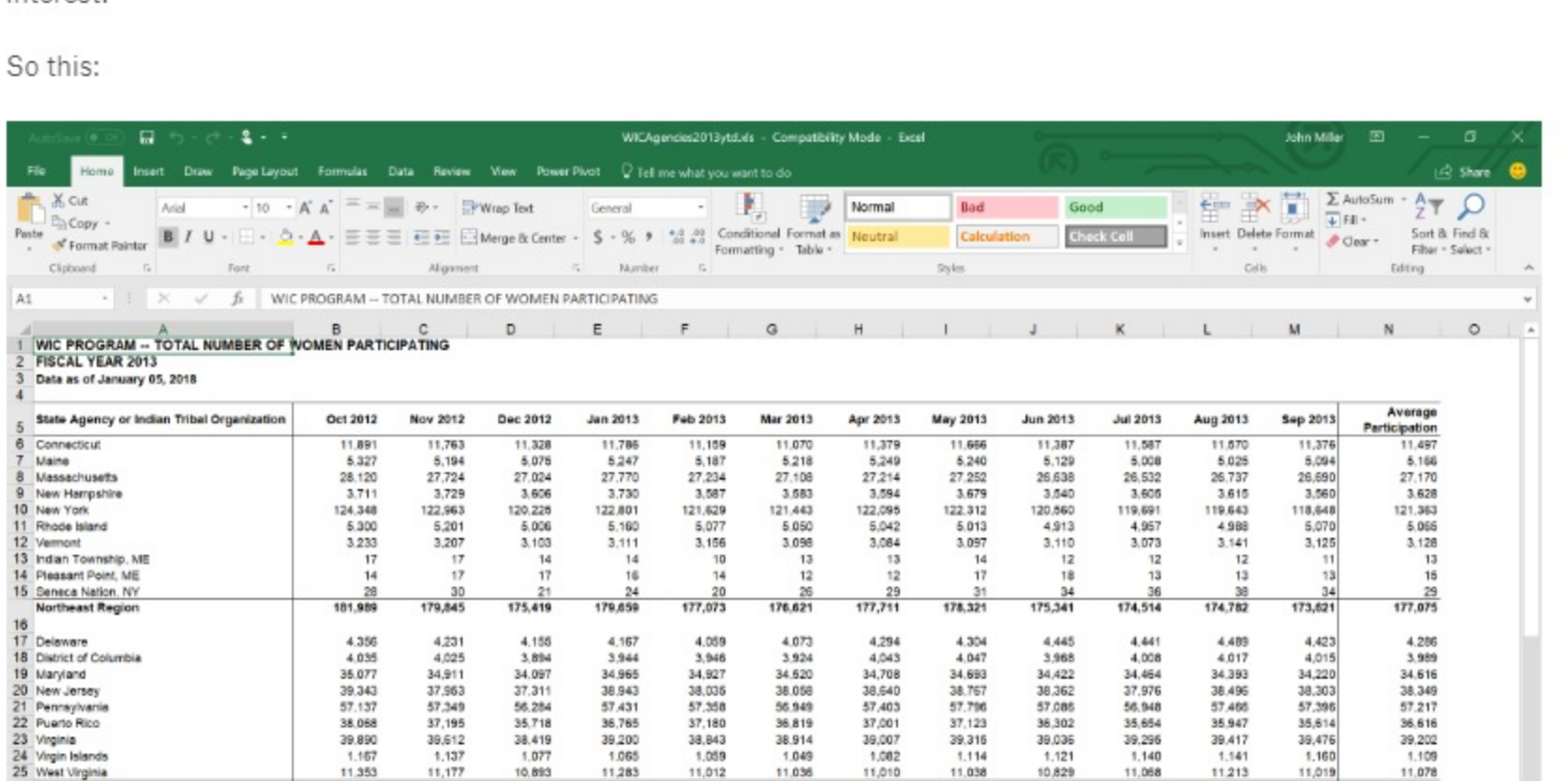
Out[10]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	titl
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nic 20 Bia (Et
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Qu Avi 20 Avi Re (Do
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Ra 20 Gri (W Val
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN	St. 20 Re Lat Ha Rie
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	Sw Ch 20 Vir Re Wi Ch

Let's look at a few more datatypes you're likely to encounter.

First up, the venerable Excel spreadsheet. An Excel file (`XLS` or `XLST`) organizes itself as a sequence of named sheets. Each sheet is basically a table. So to load the data into `pandas` we need one additional parameter: the name of the sheet of interest.

So this:



Becomes this:

In [11]:
```
wic = pd.read_excel("../input/publicassistance/xls_files_all/WICAgencies2013ytd.xls",
                    sheet_name='Total Women')
wic.head()
```

Out[11]:

	WIC PROGRAM -- TOTAL NUMBER OF WOMEN PARTICIPATING	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	FISCAL YEAR 2013	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Data as of January 05, 2018	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	State Agency or Indian Tribal Organization	2012-10-01 00:00:00	2012-11-01 00:00:00	2012-12-01 00:00:00	2013-01-01 00:00:00	2013-02-01 00:00:00	2013-03-01 00:00:00	2013-04-01 00:00:00	2013-05-01 00:00:00	2013-06-01 00:00:00
4	Connecticut	11891	11763	11328	11786	11159	11070	11379	11666	11387

As you can see in this example, Excel files are often not formatted as well as CSV files are. Spreadsheets allow (and encourage) creating notes and fields which are human-readable, but not machine-readable.

So before we can use this particular dataset, we will need to clean it up a bit. We will see how to do so in the next section.

For now, let's move on to another common data format: SQL files.

SQL databases are where most of the data on the web ultimately gets stored. They can be used to store data on things as simple as recipes to things as complicated as "almost everything on the Kaggle website".

Connecting to a SQL database requires a lot more thought than reading from an Excel file. For one, you need to create a **connector**, something that will handle siphoning data from the database.

`pandas` won't do this for you automatically because there are many, many different types of SQL databases out there, each with its own connector. So for a SQLite database (the only kind supported on Kaggle), you would need to first do the following (using the `sqlite3` library that comes with Python):

In [12]:
```
import sqlite3
conn = sqlite3.connect("../input/188-million-us-wildfires/FPA_FOD_20170508.sqlite")
```

The other thing you need to do is write a SQL statement. Internally, SQL databases all operate very differently. Externally, however, they all provide the same API, the "Structured Query Language" (or...SQL...for short).

We (very briefly) need to use SQL to load data into

For the purposes of analysis however we can usually just think of a SQL database as a set of tables with names, and SQL as a minor inconvenience in getting that data out of said tables.

So, without further ado, here is all the SQL you have to know to get the data out of `SQLite` and into `pandas`:

In [13]:
```
fires = pd.read_sql_query("SELECT * FROM fires", conn)
```

Every SQL statement beings with `SELECT`. The asterisk (`*`) is a wildcard character, meaning "everything", and `FROM fires` tells the database we want only the data from the `fires` table specifically.

And, out the other end, data:

In [14]:
```
fires.head()
```

Out[14]:

	OBJECTID	FOD_ID	FPA_ID	SOURCE_SYSTEM_TYPE	SOURCE_SYSTEM	NWCG_REPORTING_AGENCY	NWCG_REPORTIN
0	1	1	FS-1418826	FED	FS-FIRESTAT	FS	USCAPNF
1	2	2	FS-1418827	FED	FS-FIRESTAT	FS	USCAENF
2	3	3	FS-1418835	FED	FS-FIRESTAT	FS	USCAENF
3	4	4	FS-1418845	FED	FS-FIRESTAT	FS	USCAENF
4	5	5	FS-1418847	FED	FS-FIRESTAT	FS	USCAENF

5 rows × 39 columns

Writing common file formats

Writing data to a file is usually easier than reading it out of one, because `pandas` handles the nuisance of conversions for you.

We'll start with CSV files again. The opposite of `read_csv`, which reads our data, is `to_csv`, which writes it. With CSV files it's dead simple:

In [15]:
```
wine_reviews.head().to_csv("wine_reviews.csv")
```

To write an Excel file back you need `to_excel` and the `sheet_name` again:

In [16]:
```
wic.to_excel('wic.xlsx', sheet_name='Total Women')
```

And finally, to output to a SQL database, supply the name of the table in the database we want to throw the data into, and a connector:

In [17]:
```
conn = sqlite3.connect("fires.sqlite")
fires.head(10).to_sql("fires", conn)
```

Painless!

Comments (32) All Comments Sort by Hotness

Click here to enter a comment...

Jane Doe • Posted on Latest Version • 12 days ago • Options • Reply 1

Hello, Is anyone else always getting a False when running the check_qN() function to check answers? It appears that whatever I type it results in False, even if my answer is the same as what print(answer_qN()) gives. It's the same for exercise 1 and 2.

Abhik Patel • Posted on Latest Version • 7 days ago • Options • Reply 0

Can anyone help answer this?

Javed Halani • Posted on Latest Version • 7 days ago • Options • Reply 0

Are you executing your program and then running a check ?

doodhwala • Posted on Latest Version • 6 days ago • Options • Reply 0

It worked fine for me. You could write code like this:

```
df = pd.DataFrame()
check_q1(df)
```

Liz Ravenwood • Posted on Latest Version • 2 months ago • Options • Reply 1

Thanks. I appreciate this. I was a little confused with the exercise because the text said, "write to a data frame that looks like this" when I didn't see a data frame. Also, there were embedded png file paths that didn't show images, so that confused me. I'm soooo new that I didn't know I needed to click inside the box to even get it to activate. HA! This is cool though. I am having fun.

vigilance • Posted on Version 7 • 3 months ago • Options • Reply 2

hello, First, great thanks for this handy tutorial. Second thing that I want to mention, is that in the "**Writing common file formats**" is that in this sentence We'll start with CSV files again. The opposite of `from_csv`, which reads our data, is to_csv, which writes it. With CSV files the from_csv function shouldn't be `read_csv`

thanks again

Aleksey Bilogur Kernel Author • Posted on Version 7 • 3 months ago • Options • Reply 0

Fixed. Thanks for reporting!

GSD • Posted on Version 6 • 3 months ago • Options • Reply 0

Thanks for the share Aleksey ..You had not mentioned how to view the various tables present inside the db..I did a quick google check and figured out the solution..Thanks again...Here are my solutions : https://www.kaggle.com/gsdeepakkumar/creating-reading-writing-data-pandas-exercise

Alex Achterberg • Posted on Version 6 • 3 months ago • Options • Reply 0

Hi Deepak Kumar, it wasn't explicitly mentioned but in any notebook you can see the tables detail in a sqlite database browsing at the "Data" tab. Regards !

Anurag Nandigama • Posted on Version 6 • 3 months ago • Options • Reply 0

In terminal 11, while reading from the excel sheet, the specified keyword 'sheetname' is warned as deprecated. Please look here: /opt/conda/lib/python3.6/site-packages/pandas/util/_decorators.py:118: FutureWarning: The `sheetname` keyword is deprecated, use `sheet_name` instead return func(*args, **kwargs)

Though it doesn't break the code now, just a note!

Aleksey Bilogur Kernel Author • Posted on Version 7 • 3 months ago • Options • Reply 0

I've changed the parameter to `sheet_name`. Thanks!

Yash Shevde • Posted on Version 6 • 3 months ago • Options • Reply 0

the pd.read_excel parameter of 'sheetname' is deprecated. Please replace with 'sheet_name'

Paul Green • Posted on Version 7 • 3 months ago • Options • Reply 0

Hi, I very much appreciate the tutorial and came here from email prompt but int he early tasks where you create a dataframe using the dict I get the following message and cannot see what I am doing wrong here. Thanks

TypeError Traceback (most recent call last) in () ----> 1 pd.DataFrame({'Apples':[30], 'Bananas':[21]})

TypeError: 'dict' object is not callable

Aleksey Bilogur Kernel Author • Posted on Version 7 • 3 months ago • Options • Reply 0

I do not see what you are doing wrong here either. Are you sure that's the formatting you're using? Usually you get that error when you try to do e.g. `{1: 2}()`.

Karthik K • Posted on Version 7 • 3 months ago • Options • Reply 0

Hai, very nice tutorial for beginners like me and straight forward type. I got to know about read from excel, write to csv, excel files . Thanks guys

AntonyDS • Posted on Version 7 • 3 months ago • Options • Reply 0

Hi guys. Excellent job, top marks for clarity and simplicity. Thanks a ton.

Rocky • Posted on Version 7 • 3 months ago • Options • Reply 0

Think you!(•´▽`•)

hossein bahrami • Posted on Version 8 • 2 months ago • Options • Reply 0

Thanks very much.

Yevhen Diachenko • Posted on Latest Version • 2 months ago • Options • Reply 0

Thank you, great tutorial!

MAmidon • Posted on Latest Version • 2 months ago • Options • Reply 0

Well written. Very concise. Easy to understand examples.

Tomasz Sikora • Posted on Latest Version • 2 months ago • Options • Reply 0

Thank you. Wasn't expecting Pandas to handle SQL too.

Wei Shaojie • Posted on Latest Version • 2 months ago • Options • Reply 0

helpful

Prashant Chaturvedi • Posted on Latest Version • 2 months ago • Options • Reply 0

Perfect tutorial for beginners, but SQL part is a little bit tough.what should I refer to SQL part? Thanks.

Aleksey Bilogur Kernel Author • Posted on Latest Version • 2 months ago • Options • Reply 0

There's a `Learn` track for SQL (here), so you could do that.

LALIT MOHAN YADAV • Posted on Latest Version • a month ago • Options • Reply 0

Thanks for this great handy tutorial

a month ago

This Comment was deleted.

Andrii Trokoz • Posted on Latest Version • a month ago • Options • Reply 0

invaluable tutorial! thanks so much!!

Akshata Chalke • Posted on Latest Version • a month ago • Options • Reply 0

Thanks a lot for this tutorial.

Deokjin Jeong • Posted on Latest Version • 19 days ago • Options • Reply 0

helpful! thanks :D

nanmu42 • Posted on Latest Version • 17 days ago • Options • Reply 0

Thanks for this wonderful course! `:-D`

Harry Morris • Posted on Latest Version • 11 days ago • Options • Reply 0

Really useful introduction exercise for learning the basics of reading and writing different datatypes using pandas. Really clear and concise instructions for a beginner!

Josh Hamilton • Posted on Latest Version • 5 days ago • Options • Reply 0

Thanks for the post, it was very useful!

But please, if you would, look over your Notebook again; there are a ton of typos and broken sentences.