**Introduction to Pandas**

**How to load and save .csv files, series and dataframe variable types**

Pandas is one of the most popular Python libraries for Data Science and Analytics. In this pandas worksheet series, you will learn the most important (that is, the most often used) things that you have to know as an Analyst or a Data Scientist. Before you can use pandas in Jupyter notebook. You need to install Pandas libraries with the following command (in the Jupyter notebook)

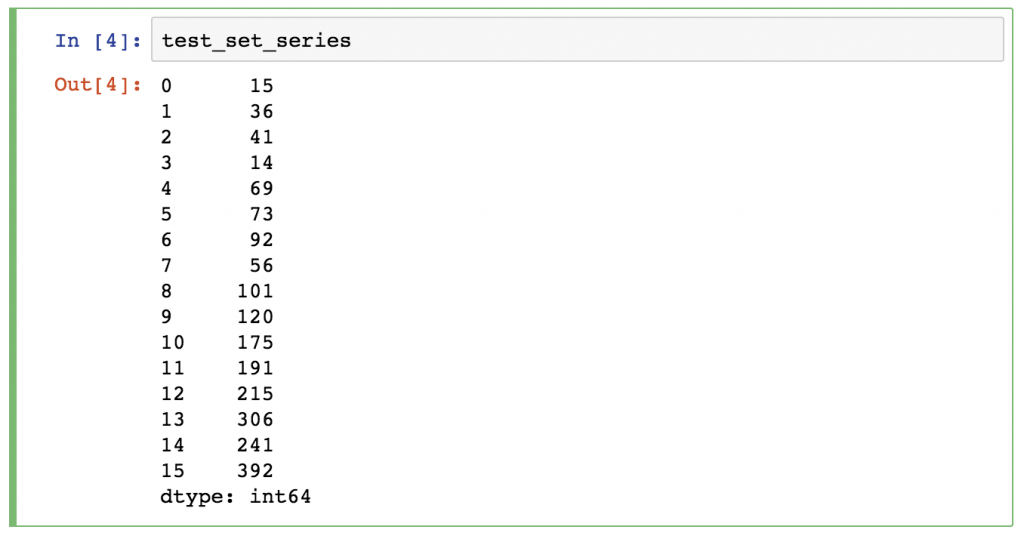
1. **pip install pandas**

After the installation is completed, the next question is how to open data files in pandas.

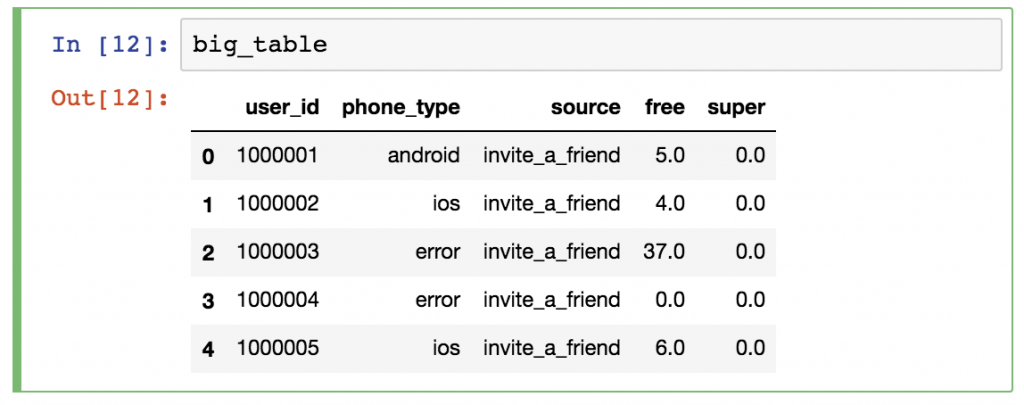
You might have your data in .csv files or SQL tables. Maybe Excel files. Or .tsv files. Or something else. But the goal is the same in all cases. If you want to analyze that data using pandas, the first step will be to read it into a data structure that’s compatible with pandas. Let’s firstly understand Pandas data structures.

|  |
| --- |
| **Pandas Data Structures**  There are two types of data structures in pandas: **Series** and **Dataframes**.  **Series:** a pandas Series is a one dimensional data structure (“a one dimensional ndarray”) that can store values — and for every value it holds a unique index, too.  **Dataframe:** a pandas dataframe is a two (or more) dimensional data structure – basically a table with rows and columns. The columns have names and the rows have indexes. |

An example of Pandas series is given below.



An example of Pandas Dataframe is given below.



We will focus mostly on Dataframes. The reason is simple: most of the analytical methods I will talk about will make more sense in a 2D datatable than in a 1D array.

Okay, time to put things into practice! Let’s load a .csv data file into pandas!  
There is a function for it, called read\_csv().

Start with a simple demo data set, called zoo! This time – for the sake of practicing – you will create a .csv file for yourself! Here’s the raw data:

animal,uniq\_id,water\_need

elephant,1001,500

elephant,1002,600

elephant,1003,550

tiger,1004,300

tiger,1005,320

tiger,1006,330

tiger,1007,290

tiger,1008,310

zebra,1009,200

zebra,1010,220

zebra,1011,240

zebra,1012,230

zebra,1013,220

zebra,1014,100

zebra,1015,80

lion,1016,420

lion,1017,600

lion,1018,500

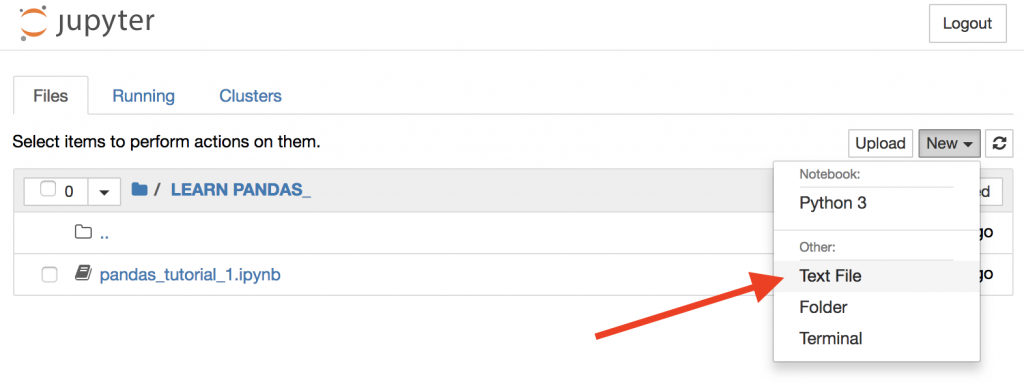
lion,1019,390

kangaroo,1020,410

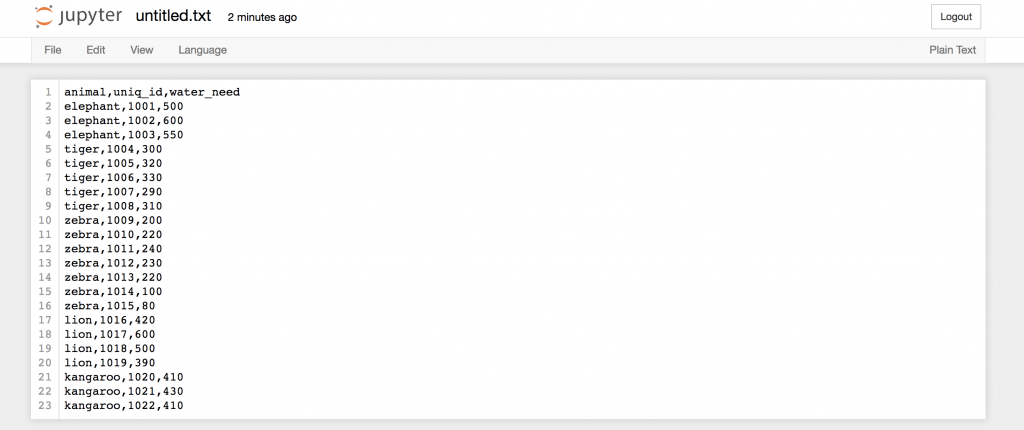
kangaroo,1021,430

kangaroo,1022,410

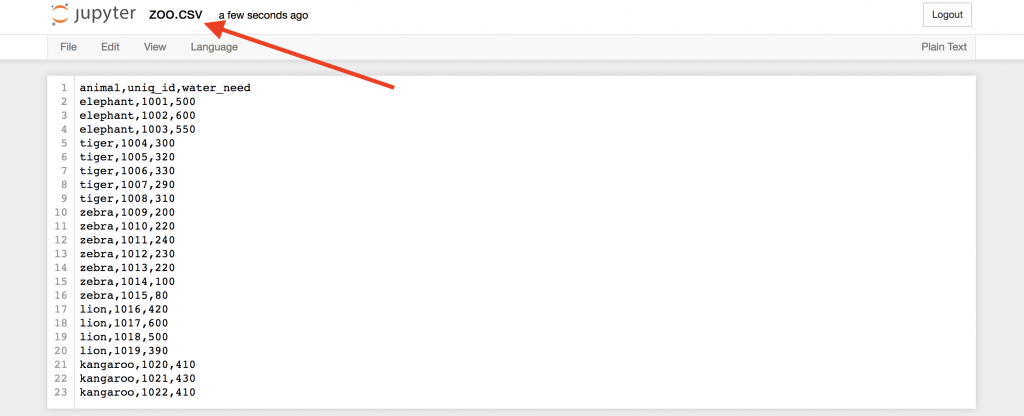
Go back to your Jupyter Home tab and create a new text file…



…then copy-paste the above zoo data into this text file…



… and then rename this text file to zoo.csv!



Okay, this is our first .csv file.  
Now, go back to your Jupyter Notebook and open this freshly created .csv file in it!

Again, the function that you have to use is: read\_csv() Prior to loading .csv file, you need to import two necessary libraries as follows:



Type this to a new cell:

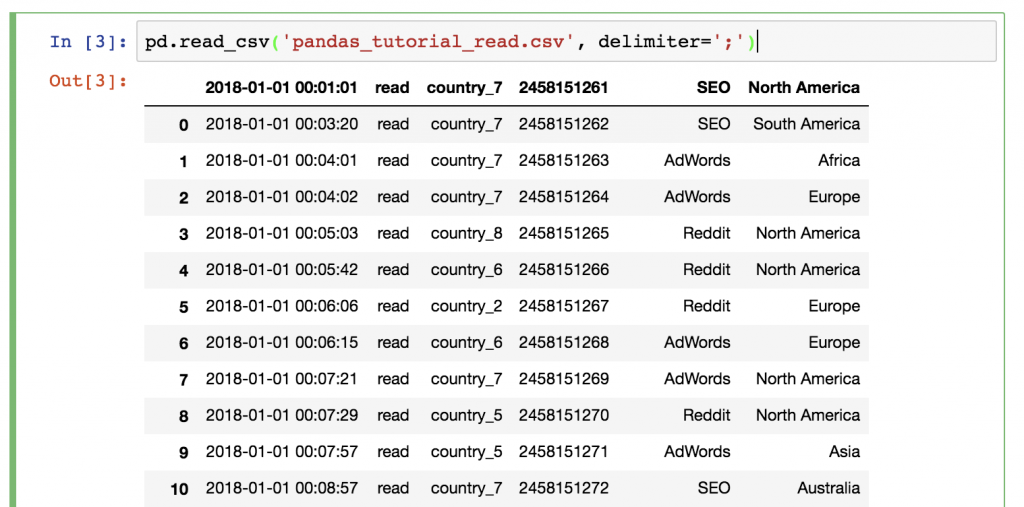
pd.read\_csv('zoo.csv', delimiter = ',')



And there you go! This is the zoo.csv data file, brought to pandas. This nice 2D table? Well, this is a pandas dataframe. The numbers on the left are the indexes. And the column names on the top are picked up from the first row of our zoo.csv file.

From the above exercises, you have learned how to create your own 2D data and saved them into .csv file. To be honest, though, you will probably never create a .csv data file for yourself, like we just did… you will use pre-existing data files.

1. **Visit CS1201 portal, download pandas\_tutorial\_read.csv and save the file in the same folder that stores your .ipynb file.**
2. **Load data from pandas\_tutorial\_read.csv using pd.read\_csv() function. The following result is expected.**



Does something feel off? Yes, this time we didn’t have a header in our .csv file, so we have to set it up manually! (since we may not want to modify the original .csv file) Add the names parameter to your function!

The following names parameter can be added while we call pd.read\_csv() function.

pd.read\_csv('pandas\_tutorial\_read.csv', delimiter=';', names = ['my\_datetime', 'event', 'country', 'user\_id', 'source', 'topic'])

The most basic method is to print your whole data frame to your screen. Of course, you don’t have to run the pd.read\_csv() function **again** and **again** and **again**. Just store its output the first time you run it!

[Note: pd is the loaded Pandas libraries and .read\_csv() is one of the functions in the libraries.]

article\_read = pd.read\_csv('pandas\_tutorial\_read.csv', delimiter=';', names = ['my\_datetime', 'event', 'country', 'user\_id', 'source', 'topic'])

After that, you can call this article\_read **(a variable name with Dataframe type)** value anytime to print your Dataframe!

1. **Use the above function to load the data from .csv file into article\_read. Then type article\_read in the next cell to observe data on the screen.**

Sometimes, it’s handy not to print the whole dataframe and flood your screen with data. When a few rows is enough, you can print only the first 5 lines – by typing: You can specify a number of lines to be shown by adding parameter head = 10 (10 rows) in the following function.

article\_read.head()

1. **Show the data for the first 5, 10 and 15 rows.**

You can show a few last rows using article\_read.tail() or show a few random rows using article\_read.sample(5)

1. **Try article\_read.tail() and article\_read.sample() to observe the results.**

**Select specific columns of your dataframe**

This one is a bit tricky! Let’s say you want to print the ‘country’ and the ‘user\_id’ columns only. You should use this syntax:

article\_read[['country', 'user\_id']]

**Any guesses why we have to use double bracket frames? It seems a bit over-complicated,** but maybe this will help you remember: the outer bracket frames tell pandas that you want to select columns, and the inner brackets are for the list (*remember? Python lists go between bracket frames*) of the column names.

By the way, if you change the order of the column names, the order of the returned columns will change, too:

1. **Try to show data from other columns (also try with different orders)**

*Note: Sometimes (especially in predictive analytics projects), you want to get Series objects instead of dataframes. You can get a Series using any of these two syntaxes (and selecting only one column):*

article\_read.user\_id  
article\_read['user\_id']

|  |
| --- |
| **How to filter for specific values in your dataframe** |

If the previous one was a bit tricky, this one will be really tricky!

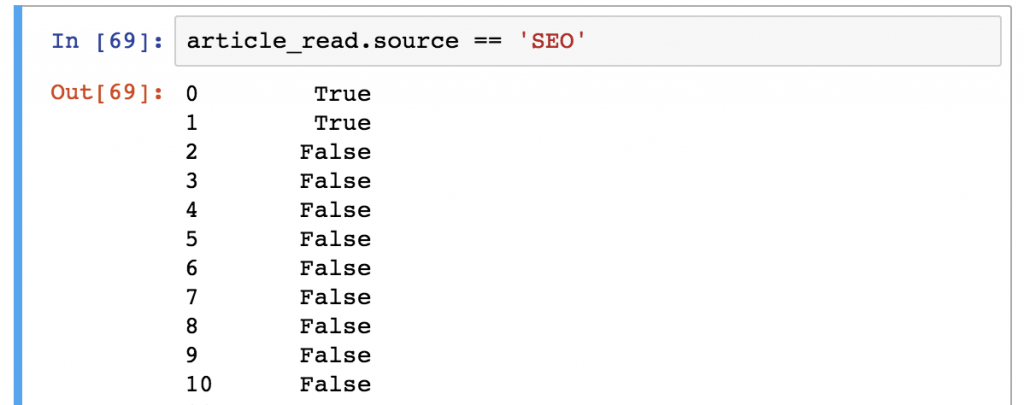
Let’s say, you want to see a list of only the users who came from the ‘SEO’ source. In this case you have to filter for the ‘SEO’ value in the ‘source’ column:

article\_read[article\_read.source == 'SEO']

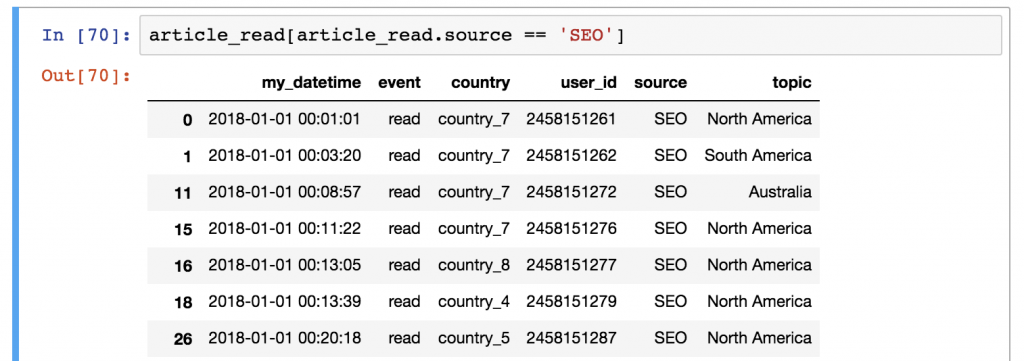
It’s worth it to understand how pandas thinks about data filtering: If we only typ

article\_read.source == 'SEO'

**STEP 1)** First, between the bracket frames it evaluates every line: is the article\_read.source column’s value 'SEO' or not? The results are boolean values (True or False).



**STEP 2)** Then from the article\_read table, it prints every row where this value is True and doesn’t print any row where it’s False as shown below.



1. **Show all records where source is from Reddit.**
2. **Show all records where country is Europe.**

|  |
| --- |
| **Functions in Pandas can be used after each other** |

It’s very important to understand that Pandas’s logic is very linear (So if you apply a function, you can always apply another one on it. In this case, the input of the latter function will always be the output of the previous function.

E.g. combine these two selection methods:

article\_read.head()[['country', 'user\_id']]

This line first selects the first 5 rows of our data set. And then it takes only the ‘country’ and the ‘user\_id’ columns.

Could you get the same result with a different chain of functions? Of course you can:

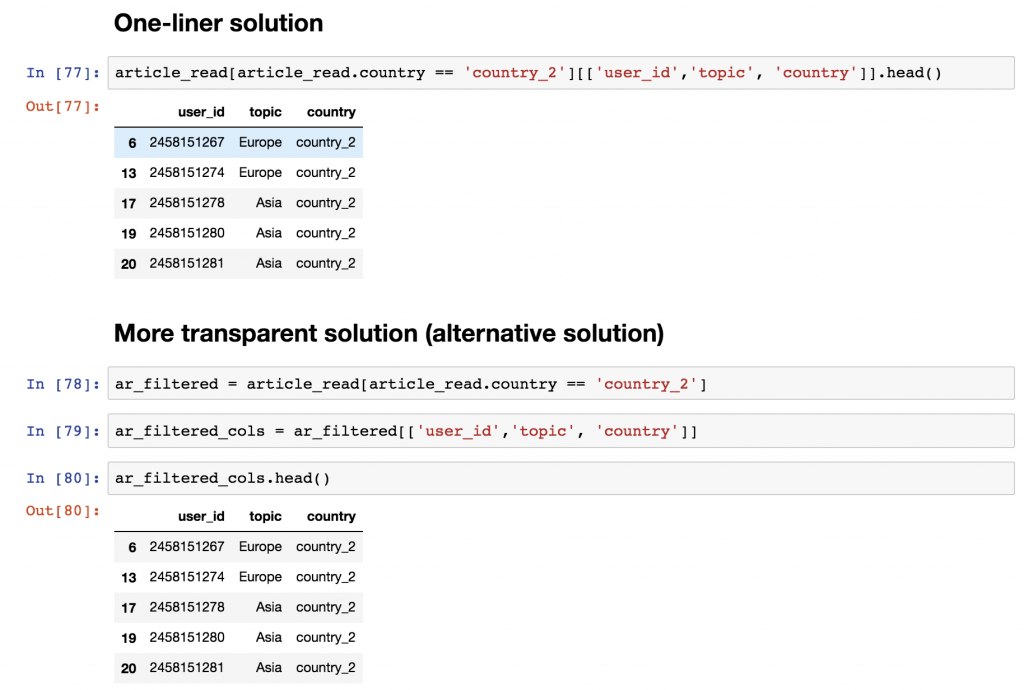
article\_read[['country', 'user\_id']].head()

In this version, you select the columns first, then take the first five rows. The result is the same – the order of the functions (and the execution) is different.

1. **Try the above two Pandas codes to see the results.**
2. **Show the first five records of user\_id, country and topic for the users who are from ‘country\_2’. (Hint: you need to filter for ‘country\_2’ and select only user\_id, country, topic, and then call .head() )**

**[The solution is available on the next page. There are two alternatives.]**

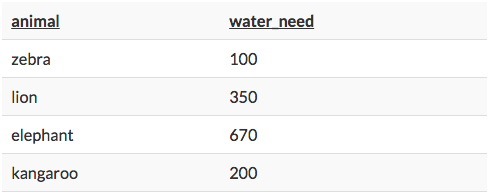
1. **Show the first ten records of user\_id source topic where source is from Reddit and topic is ‘Asia’.**



**Data Aggregation and Grouping in Pandas**

In this section, you will be introduced to aggregation (such as min, max, sum, count, etc.) and grouping. Both are very commonly used methods in analytics and data science projects – so make sure you go through every detail in this article!

Data aggregation is the process of turning the values of a dataset (or a subset of it) into one single value. Let me make this clear! If you have a dataframe like…



…then a simple aggregation method is to calculate the summary of the water\_needs, which is 100 + 350 + 670 + 200 = 1320. Or a different aggregation method would be to count the number of the animals, which is 4. So the theory is not too complicated. Let’s see the rest in practice…

Let’s store this dataframe into a variable called zoo.

zoo = pd.read\_csv('zoo.csv', delimiter = ',')

Okay, let’s do five things with this data:

1. Let’s count the number of rows (the number of animals) in zoo!
2. Let’s calculate the total water\_need of the animals!
3. Let’s find out which is the smallest water\_need value!
4. And then the greatest water\_need value!
5. And eventually the average water\_need!

Counting the number of the animals is as easy as applying a count function on the zoo dataframe using zoo.count(). Actually, the .count() function counts the number of values in each column. In the case of the zoo dataset, there were 3 columns, and each of them had 22 values in it.



1. **Show the count value of animal column only.**
2. **Show the count value of animal and water\_need columns.**

Following the same logic, you can easily sum the values in the water\_need column by typing:

zoo.water\_need.sum()

1. **Try zoo.water\_need.sum()**
2. **Find the sum for animal column.**

**What’s the smallest value in the water\_need column? Guess:**

1. **Find max and min values of water\_need.**

**Find what is the difference between Median and Mean. I will randomly come to you for an answer.**

In Pandas, you can find Mean and Median using .mean() and .median(), respectively.

1. **Find mean and median values of water\_need.**

**[The correct mean and median values are 347.7272 and 325.0, respectively.]**