

Data Wrangling: Making data useful for analysis

This tutorial will demonstrate a way to load data from an uncommonly formatted file and change the data to a more common and usable format.

The tutorial will use .CSV files as format for the raw input data and Pandas Data Frames as example of output data. The goal is to take the input data format and reorganize the data into a [Tidy Data Format \(https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html\)](https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html).

Learning outcomes

- Understand the importance of data preparation
- Read and Write files from disk into a Jupyter Notebook
- Manipulate Pandas Data Frames

The best rat lab

We are dealing with a lab that studies animal behavior. This is one of the best labs.

Generally in animal behavior labs rats are studied under different conditions that might vary along several dimensions, for example, cognitive, social, emotional, biological (drugs) or genetic.

The lab has developed a new strain of rats that seem to have interesting properties. The new mutant mouse is [stronger, better, faster \(https://www.youtube.com/watch?v=yydNF8tuVmU\)](https://www.youtube.com/watch?v=yydNF8tuVmU) than the wild type mouse.

A new Research Assistant has spent the 2022 Spring Break in the lab measuring the fascinating behavioral abilities of the super rat. We are lucky because we have been handed the amazing data from this new rat strain.

Here after we will load the data and organize it into a more convenient format for plotting and further analysis.

The data files

The data have been handed to us in a .CSV file. [The CSV file format \(https://en.wikipedia.org/wiki/Comma-separated_values\)](https://en.wikipedia.org/wiki/Comma-separated_values) is a very common format used in science and engineering to handle small datasets consisting of numerical recordings of events in time, space or other types of measurements.

CSV stands for Comma Separated Values. The commas separate the meaningful units of the data, these can be labels or actual numerical data. Data can be organized in rows or columns or both.

For example, let look at the anatomy of the dataset we were handed has the following structure (yes, OK we are having a little bit ahead of us, as we have not loaded the data yet):

male wild type	female wild type	male mutant	female mutant
10	5	4	10
23	4	22	33
22	23	5	33
11	25	5	4

This is not a convenient format because the labels of interest (male/female, wild type/mutant) have been mixed up and their respective data separated. A more generally used and convenient organization of the same data would be the following:

Data	Sex	Strain
5	M	Mutant
6	M	Mutant
7	M	Mutant

4	M	Mutant
4	F	Mutant
7	F	Mutant
8	F	Mutant
4	F	Mutant
11	M	Wild Type
10	M	Wild Type
8	M	Wild Type
10	M	Wild Type
11	F	Wild Type
10	F	Wild Type
8	F	Wild Type
9	F	Wild Type

This format would allow making analyses that group the data based on the strain of the rats and their sex.

We made a video to show visually the type of data reorganization we will perform with the code hereafter.

The screenshot shows a Google Sheets spreadsheet titled "TheBestLab". The spreadsheet contains a table with the following data:

	Male Wild Type	Female Wild Type	Male Mutant	Female Mutant	Data	Mouse Type	Mouse Sex
2	5	5	0	0	5	WT	Male
3	6	4	9	7	6	WT	Male
4	6	4	7	9	6	WT	Male
5	7	5	0	7	7	WT	Male
6	5	5	10	9	5	WT	Male
7	4	6	0	8	4	WT	Male
8	5	5	8	10	5	WT	Male
9	6	5	7	8	6	WT	Male

(<https://www.youtube.com/watch?v=qR8rL0GiOKE>).

A peak at the dataset

Before we start coding, let's take a look at a snapshot of the dataset we are going to load. The numbers might be different but the basic structure of the dataset will be similar.

 jupyter 017DataFile.csv ✓ 3 minutes ago

	File	Edit	View	Language
1	Male Mutant,Female Mutant,Male Wild Type,Female Wild Type			
2	12.088281281759626,10.093227846701438,20.077613775639325,24.007070049558237			
3	11.96430123531203,9.94390446702576,19.88266430299905,23.794170284727755			
4	11.92679895116101,9.899150380016037,20.047606411305352,23.952991758451574			
5	12.104004208680951,9.93922229517685,20.145598303629615,24.04035949617375			
6	11.99792941935085,10.165117114043019,19.999009929156216,23.841910204589222			
7	12.05436961486714,10.002744031166644,19.98494666689937,23.931014020997804			
8	11.93276567809532,10.01199269096689,20.044155890456686,23.9272371774385			
9	12.052543874996877,10.029897561945388,19.811331106317244,23.933809411203825			
10	12.02409150458776,10.082685876670725,20.061199732551927,24.10531299439611			
11	11.980865964767057,9.979996961495994,19.96386320336739,23.963402675646204			
12				

The data set has 10 rows of data and one of headers (labels for the various types of rats). The headers are on top of the file and the numerical data follows that. In our example, the numerical data represent reaction times. The rats had to run in a maze and the time it took them to start the maze have been recorded in the data.

All elements are separated by commas. Well indeed this is a CSV file! Good.

Loading the dataset using Padas

We have used Pandas in the past. Pandas is one of the most used libraries for data science and more generally science.

Pandas uses DataFrames. Hereafter, we will load the dataset into a Pandas DataFrame.

```
In [1]: 1 # We import Padas
        2 import pandas as pd
```

Pandas has a dedicated CSV reader. We will use the reader to load the CSV file into a DataFrame (df). Pandas will recognize the commas in the CSV (comma-separated file) and map the elements of the file into files of the data frame.

```
In [3]: 1 df = pd.read_csv('datasets/017DataFile.csv')
```

Note. The above line will only work if the dataset is saved inside a folder called 'datasets', saved inside the current folder used to launch this Jupyter Notebook. Please make sure to create that folder (`mkdir`) and move (`mv`) the data file downloaded for this tutorial inside that folder.

Complete the following exercise.

- Identify other readers Pandas provide. Use the cell below to report 2-3 readers (write them down) and explain the type of file they each can read. (Hint. You can use `TAB` to find all the other methods that start with 'read'.)

Type *Markdown* and LaTeX: α^2

After loading the dataset in a Pandas Data Frame, we can now take a look at the first five columns of the data frame by using the method `head` . Head returns the beginning of a data frame.

In [4]:

```
1 df.head()
```

Out [4]:

	Male Mutant	Female Mutant	Male Wild Type	Female Wild Type
0	10.485451	8.250013	20.127063	25.946384
1	11.747948	8.453839	20.068147	23.464870
2	13.412580	9.706605	21.215148	22.989480
3	12.910095	9.522116	20.706416	25.324376
4	10.367770	8.583212	18.074795	22.607487

We can also use tail to look at the last 5 rows of the data frame.

In [5]:

```
1 df.tail()
```

Out [5]:

	Male Mutant	Female Mutant	Male Wild Type	Female Wild Type
5	11.698422	9.835002	20.367624	23.052187
6	11.583153	10.532096	20.152521	25.369037
7	11.447349	9.394166	19.392476	23.372709
8	10.852276	8.739473	18.524341	25.215646
9	11.285897	10.892394	20.325026	24.990505

We can get some basic statistics about the dataset. The `describe` method of the pandas data frame will return the count, the mean the standard deviation, the min value, the the 25th, 50th and 75th percentile and the max value.

A good set of stats useful for many things.

In [6]: `1 df.describe()`

Out [6]:

	Male Mutant	Female Mutant	Male Wild Type	Female Wild Type
count	10.000000	10.000000	10.000000	10.000000
mean	11.579094	9.390892	19.895356	24.233268
std	0.970163	0.889904	0.966323	1.241408
min	10.367770	8.250013	18.074795	22.607487
25%	10.960681	8.622278	19.561394	23.132318
50%	11.515251	9.458141	20.139792	24.227687
75%	11.735566	9.802902	20.356975	25.297194
max	13.412580	10.892394	21.215148	25.946384

Complete the following exercise.

- Test whether the method `describe` works only on `DataFrames` or also on `Series`. To do so use the cell below to create a Pandas `Series` (it is not important what the series contains), and test with code whether the method works also for a `Series`.


```
In [9]: 1 month_2022 = {'January':31,
2           'February': 28,
3           'March': 31,
4           'April': 30,
5           'May': 31,
6           'June':30,
7           'July':31,
8           'August':31,
9           'September':30,
10          'October':31,
11          'November':30,
12          'December':31
13        }
14
15 month_2022_Series = pd.Series(month_2022)
16
17 month_2022_Series.index
18
19 month_2022_Series['January':'December']
```

```
Out[9]: January      31
February    28
March       31
April       30
May         31
June        30
July        31
August      31
September   30
October     31
November    30
December    31
dtype: int64
```

```
In [11]: 1 month_2022_Series.describe()
```

```
Out[11]: count    12.000000  
mean      30.416667  
std        0.900337  
min       28.000000  
25%       30.000000  
50%       31.000000  
75%       31.000000  
max       31.000000  
dtype: float64
```

- Does the method `describe` take inputs? Use the cell below describe any additional inputs `describe` might take (if any). Explain what the input do/mean (if they exist).

The method `describe` does work. The `describe` method also works with the `pandas.series` because we can still get the values such as *count*, *mean*, *std*, etc.

Reorganizing the dataset

As we described above instead of one column per type of rat (male mutant, female mutant, male wild type, female wild type) we would like to have all the data in the first column and the labels (male or female, wild type or mutant) in the second and third column respectively.

There are multiples ways to reorganize the dataset. We will use what is called slicing.

We will address each column of the dataset using the header of the column (the label). For example, below we address the first column of the data frame:

```
In [12]: 1 df['Male Mutant']
```

```
Out[12]: 0    10.485451  
1    11.747948  
2    13.412580  
3    12.910095  
4    10.367770  
5    11.698422  
6    11.583153  
7    11.447349  
8    10.852276  
9    11.285897  
Name: Male Mutant, dtype: float64
```

Complete the following exercise.

- Use the cell below to show the third column of the dataset.

```
In [13]: 1 df['Male Wild Type']
```

```
Out[13]: 0    20.127063  
1    20.068147  
2    21.215148  
3    20.706416  
4    18.074795  
5    20.367624  
6    20.152521  
7    19.392476  
8    18.524341  
9    20.325026  
Name: Male Wild Type, dtype: float64
```

Our goal is to stack the data into the first column of a new data frame. After that, we will want to add the labels into the second and third columns of the new data frame.

Let's start by creating a new data frame. We have chose to start withg an empty data frame, create a column at the time and add the columns to the data frame as the data foreach column is created.

(This could have been done in at least a few different ways.)

```
In [14]: 1 new_df = pd.DataFrame( [])
```

```
In [15]: 1 new_df
```

```
Out[15]: —
```

Next we want to stack the data from all the conditions of the original data frame into a single column data frame.

We can do this by first initializng a data frame and after that stacking the data using the Pandas' command `concat` which stands for concatenation.

```
In [16]: 1 data = pd.DataFrame()  
2 data = pd.concat([df['Male Mutant'],  
3                  df['Female Mutant'],  
4                  df['Male Wild Type'],  
5                  df['Female Wild Type']])
```

Complete the following exercise.

- Use the cell below to describe what the method `concat` does to data frames.

The method `concat` combine selected data into one long data set.

Let's take a look at the data frame containing the data.

```
In [17]:
```

1	data
---	------

```
Out[17]: 0    10.485451
          1    11.747948
          2    13.412580
          3    12.910095
          4    10.367770
          5    11.698422
          6    11.583153
          7    11.447349
          8    10.852276
          9    11.285897
          0     8.250013
          1     8.453839
          2     9.706605
          3     9.522116
          4     8.583212
          5     9.835002
          6    10.532096
          7     9.394166
          8     8.739473
          9    10.892394
          0    20.127063
          1    20.068147
          2    21.215148
          3    20.706416
          4    18.074795
          5    20.367624
          6    20.152521
          7    19.392476
          8    18.524341
          9    20.325026
          0    25.946384
          1    23.464870
          2    22.989480
          3    25.324376
          4    22.607487
          5    23.052187
          6    25.369037
          7    23.372709
```

```
8    25.215646
9    24.990505
dtype: float64
```

Let's check its shape. it should be four times the size of each column of data (10), so 40.

```
In [18]: 1 data.shape
```

```
Out[18]: (40,)
```

The above seems to look all good except that if we notice the index of the individual data frames going from 0-9 have been kept in the new concatenated data frame. This is because by default Pandas will concatenate by and maintain the individual indices of the concatenated data frames.

To solve this, we will do the concatenation and set the variable `ignore_index` equal `True` that will tell Pandas to recast the indices instead of keeping the one of the original data frames concatenated.

```
In [19]: 1 data = pd.DataFrame()
          2 data = pd.concat([df['Male Mutant'],
          3                     df['Female Mutant'],
          4                     df['Male Wild Type'],
          5                     df['Female Wild Type']], ignore_index=True)
          6 data
```

```
Out[19]: 0    10.485451
          1    11.747948
          2    13.412580
          3    12.910095
          4    10.367770
          5    11.698422
          6    11.583153
          7    11.447349
          8    10.852276
          9    11.285897
         10     8.250013
         11     8.453839
         12     9.706605
         13     9.522116
```

```
14      8.583212
15      9.835002
16     10.532096
17      9.394166
18      8.739473
19     10.892394
20     20.127063
21     20.068147
22     21.215148
23     20.706416
24     18.074795
25     20.367624
26     20.152521
27     19.392476
28     18.524341
29     20.325026
30     25.946384
31     23.464870
32     22.989480
33     25.324376
34     22.607487
35     23.052187
36     25.369037
37     23.372709
38     25.215646
39     24.990505
dtype: float64
```

Complete the following exercise.

- Use the cell below to describe what the `ignore_index=True` parameter does and what is the utility of such operation.

The `ignore_index=True` simply ignore the original index and put the new index into the selected data.

Now the indices are sequential 0-39. Great!

Next, we will want to add the data just created into the first column of the data frame we want to create.

```
In [22]: 1 new_df['Data'] = data
          2 # file name ['new column'] = data set
```

We have now populated the first column of the new data frame with the data. Each entry comes from the original data frame loaded from the CSV. The data for the 4 different conditions are stancked so that the data for the 'Male Mutant' is first, that for the 'Female Mutant' is second, that for the 'Male Wild Type' third, and that for the 'Female Wild Type' last.

```
In [21]: 1 new_df
```

Out[21]:

	Data
0	10.485451
1	11.747948
2	13.412580
3	12.910095
4	10.367770
5	11.698422
6	11.583153
7	11.447349
8	10.852276
9	11.285897
10	8.250013
11	8.453839
12	9.706605
13	9.522116
14	8.583212
15	9.835002
16	10.532096

17	9.394166
18	8.739473
19	10.892394
20	20.127063
21	20.068147
22	21.215148
23	20.706416
24	18.074795
25	20.367624
26	20.152521
27	19.392476
28	18.524341
29	20.325026
30	25.946384
31	23.464870
32	22.989480
33	25.324376
34	22.607487
35	23.052187
36	25.369037
37	23.372709
38	25.215646
39	24.990505

After adding the first column of data we will want to add the two missing columns defining the labels for each data point.

This step will create a column for the sex and one for the strain

Sex	Strain
M	Mutant
M	Mutant
M	Mutant
M	Mutant
F	Mutant
F	Mutant
F	Mutant
F	Mutant
M	Wild Type
M	Wild Type
M	Wild Type
M	Wild Type
F	Wild Type
F	Wild Type
F	Wild Type
F	Wild Type

Because the numerosity of each sample is known and equal across samples (10). We can use a simple assignment method to create what we need.

The first 10 elements of the Sex columns will be Males (coded as *M*), the next Females (coded as *F*), etc.

In [23]:

```
1 sex = pd.DataFrame( ['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',  
2                       'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F',  
3                       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',  
4                       'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F'])  
5 sex
```

Out[23]:

	0
0	M
1	M
2	M
3	M
4	M
5	M
6	M
7	M
8	M
9	M
10	F
11	F
12	F
13	F
14	F
15	F
16	F
17	F
18	F
19	F
20	M

21 M
22 M
23 M
24 M
25 M
26 M
27 M
28 M
29 M
30 F
31 F
32 F
33 F
34 F
35 F
36 F
37 F
38 F
39 F

We can now add a column to our data frame and the coding for sex. We can do this operation because the number of rows is the same between the two data frames.

In [24]:

```
1 new_df['Sex'] = sex
2 new_df
```

Out[24]:

	Data	Sex
0	10.485451	M
1	11.747948	M
2	13.412580	M
3	12.910095	M
4	10.367770	M
5	11.698422	M
6	11.583153	M
7	11.447349	M
8	10.852276	M
9	11.285897	M
10	8.250013	F
11	8.453839	F
12	9.706605	F
13	9.522116	F
14	8.583212	F
15	9.835002	F
16	10.532096	F
17	9.394166	F
18	8.739473	F
19	10.892394	F
20	20.127063	M
21	20.068147	M
22	21.215148	M

```
23 20.706416 M
24 18.074795 M
25 20.367624 M
26 20.152521 M
27 19.392476 M
28 18.524341 M
29 20.325026 M
30 25.946384 F
31 23.464870 F
32 22.989480 F
33 25.324376 F
34 22.607487 F
35 23.052187 F
36 25.369037 F
37 23.372709 F
38 25.215646 F
39 24.990505 F
```

OK Great in a few commands we have used added the sex labels.

The final column to add is the rat strain. This is going to be as simple as creating the sex column.

In [25]:

```

1 strain = pd.DataFrame(['Wild Type', 'Wild Type', 'Wild Type', 'Wild Type',
2                        'Wild Type', 'Wild Type', 'Wild Type', 'Wild Type',
3                        'Wild Type', 'Wild Type',
4                        'Wild Type', 'Wild Type', 'Wild Type', 'Wild Type',
5                        'Wild Type', 'Wild Type', 'Wild Type', 'Wild Type',
6                        'Wild Type', 'Wild Type',
7                        'Mutant', 'Mutant', 'Mutant', 'Mutant', 'Mutant',
8                        'Mutant', 'Mutant', 'Mutant', 'Mutant', 'Mutant',
9                        'Mutant', 'Mutant', 'Mutant', 'Mutant', 'Mutant',
10                       'Mutant', 'Mutant', 'Mutant', 'Mutant', 'Mutant'])
11 strain

```

Out[25]:

```

0
-----
0 Wild Type
1 Wild Type
2 Wild Type
3 Wild Type
4 Wild Type
5 Wild Type
6 Wild Type
7 Wild Type
8 Wild Type
9 Wild Type
10 Wild Type
11 Wild Type
12 Wild Type
13 Wild Type
14 Wild Type
15 Wild Type
16 Wild Type

```

17 Wild Type

18 Wild Type

19 Wild Type

20 Mutant

21 Mutant

22 Mutant

23 Mutant

24 Mutant

25 Mutant

26 Mutant

27 Mutant

28 Mutant

29 Mutant

30 Mutant

31 Mutant

32 Mutant

33 Mutant

34 Mutant

35 Mutant

36 Mutant

37 Mutant

38 Mutant

39 Mutant

Complete the following exercise.

- Use the cell below to explain in your own words what the cell above does. How did we organize the labels, and why?

The cell above create a new data set call `strain` , which could be added into the `new_df` dataframe.

Finally, we will add the new column labeling the rat strain:

```
In [26]: 1 new_df['Strain'] = strain
          2 new_df
```

Out[26]:

	Data	Sex	Strain
0	10.485451	M	Wild Type
1	11.747948	M	Wild Type
2	13.412580	M	Wild Type
3	12.910095	M	Wild Type
4	10.367770	M	Wild Type
5	11.698422	M	Wild Type
6	11.583153	M	Wild Type
7	11.447349	M	Wild Type
8	10.852276	M	Wild Type
9	11.285897	M	Wild Type
10	8.250013	F	Wild Type
11	8.453839	F	Wild Type
12	9.706605	F	Wild Type
13	9.522116	F	Wild Type
14	8.583212	F	Wild Type

15	9.835002	F	Wild Type
16	10.532096	F	Wild Type
17	9.394166	F	Wild Type
18	8.739473	F	Wild Type
19	10.892394	F	Wild Type
20	20.127063	M	Mutant
21	20.068147	M	Mutant
22	21.215148	M	Mutant
23	20.706416	M	Mutant
24	18.074795	M	Mutant
25	20.367624	M	Mutant
26	20.152521	M	Mutant
27	19.392476	M	Mutant
28	18.524341	M	Mutant
29	20.325026	M	Mutant
30	25.946384	F	Mutant
31	23.464870	F	Mutant
32	22.989480	F	Mutant
33	25.324376	F	Mutant
34	22.607487	F	Mutant
35	23.052187	F	Mutant
36	25.369037	F	Mutant
37	23.372709	F	Mutant
38	25.215646	F	Mutant
39	24.990505	F	Mutant

Excellent, we are done. We have created the file we want using the original data and labels.

We can now save the file. To do that and to use compatibility with other readers of the data we will want to save the data frame in a CSV file. These files can be opened from multiple systems (MS Excell or Google Spreasheet are two examples).

To save the data frame into a CSV we will use the data frame method `.to_csv()` , this will directly save the file.

```
In [28]: 1 new_df.to_csv('datasets/tutorial017data_reorganized.csv')  
        2 #Save new data set to the folder
```

We are done! We have loaded a poorly organized datasets and used operations and methods available in `data frames` to reorganize the format of the data into a `Tidy Data` format !

Complete the following exercise.

Note, that our code above, used a lot of *hard coded* sections. This does not make the code flexible, for example if a new dataset were handed to us say with 15 data entries per condition.

The goal here is to make the code adaptable to the number of rows in the original dataset handed to us. This is helpful because in principle the number of rows might vary every day.

For example ur RA might have collected 11 trials in Day 1 but only 8 in Day 2. In this scenario every day we would need to repeat a long chunk of code the following one:

```
strain = pd.DataFrame(['Wild Type','Wild Type','Wild Type','Wild Type',  
                        'Wild Type','Wild Type','Wild Type','Wild Type',  
                        'Wild Type','Wild Type',  
                        'Wild Type','Wild Type','Wild Type','Wild Type',  
                        'Wild Type','Wild Type','Wild Type','Wild Type',  
                        'Wild Type','Wild Type',  
                        'Mutant','Mutant','Mutant','Mutant','Mutant',  
                        'Mutant','Mutant','Mutant','Mutant','Mutant',  
                        'Mutant','Mutant','Mutant','Mutant','Mutant',  
                        'Mutant','Mutant','Mutant','Mutant','Mutant'])
```

- Can you think a way to use `Loops` to avoid hard coding the above lines? For example, would it be possible to replace the block of code above using a for loop? Use the cell below to show your solution.

```
In [35]: 1 organized_data = pd.read_csv('datasets/tutorial017data_reorganized.csv')
          2 organized_data.head()
```

```
Out[35]:
```

	Unnamed: 0	Data	Sex	Strain
0	0	10.485451	M	Wild Type
1	1	11.747948	M	Wild Type
2	2	13.412580	M	Wild Type
3	3	12.910095	M	Wild Type
4	4	10.367770	M	Wild Type

```
In [41]: 1 for Strain in organized_data:
          2     exec('{ } = pd.DataFrame()'.format(Strain))
```

- Alternatively can you think ways to use Pandas' data frame methods to populate the rows of your data frame flexibly? This means, that the length of the rows can change depending on the number of trials collected. Use the cell below to show your solution.

Hint. Pandas' `series.repeat` (<https://pandas.pydata.org/docs/reference/api/pandas.Series.repeat.html>) might be a good start, if you prefer to use Pandas' instead of loops to make the code flexible that would be totally fine!


```
In [4]: 1 test_data['Type'] = pd.DataFrame(s.repeat([10,12]))
        2 test_data
```

```
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
1    Mutant    Mutant
```

```
In [6]: 1 data2 = pd.DataFrame()
        2
        3 Wild_Type = test_data[test_data.Type == 'Wild Type']
        4 Mutant_Type = test_data[test_data.Type == 'Mutant']
```

1	Wild_Type
---	-----------

Out[7]:

[illegible]

In [9]: 1 Mutant_Type

Out [9]:

	0	Type
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant
1	Mutant	Mutant

Summary

OK in this tutorial we have covered

- read/write operations for CSV files
- operations formnipulating Pandas' Data Frames (create, add data, concatenation, add columns)
- We have discussed considerations regarding data organization

Keep your file created today around, we will start from that next time!

