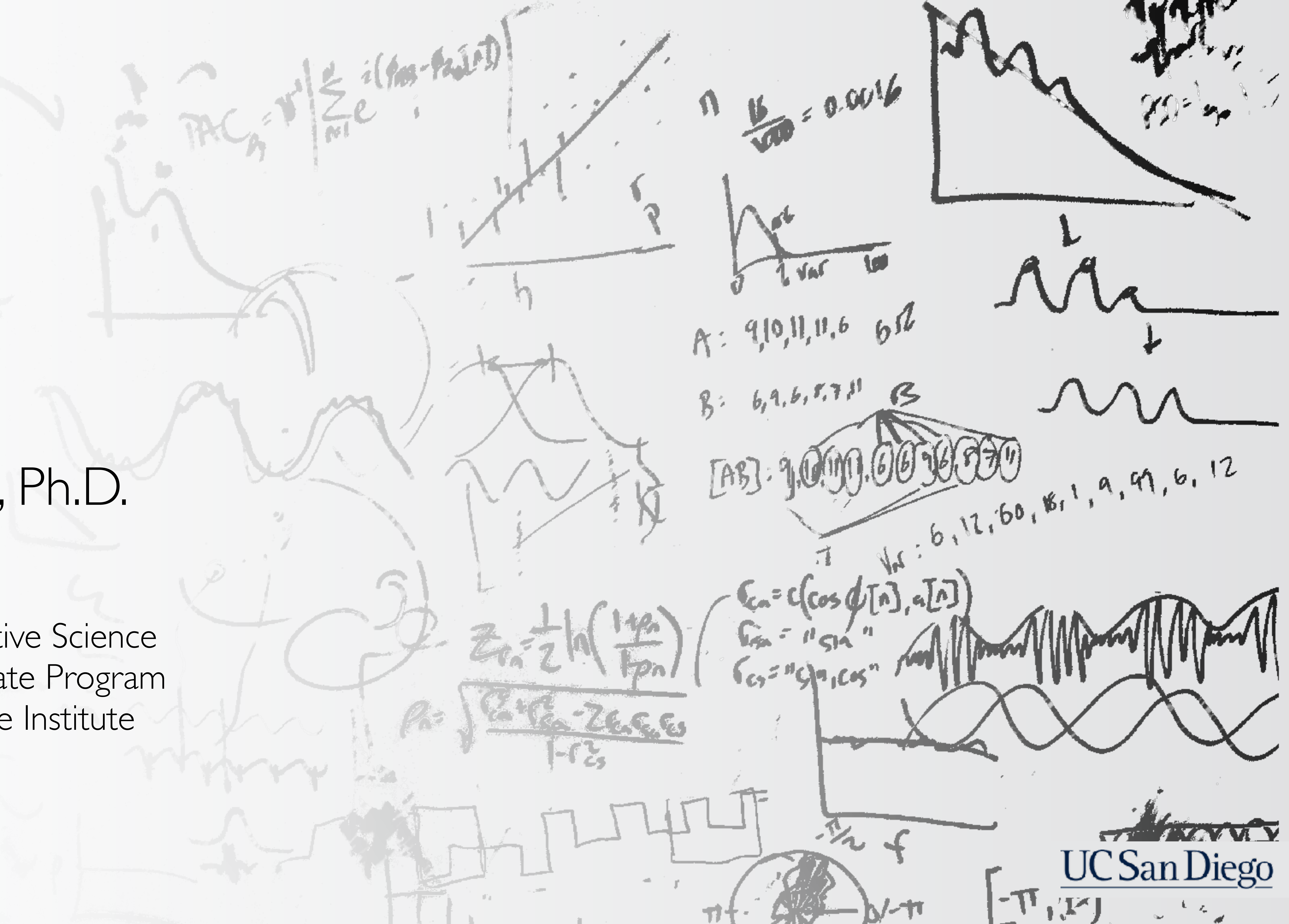


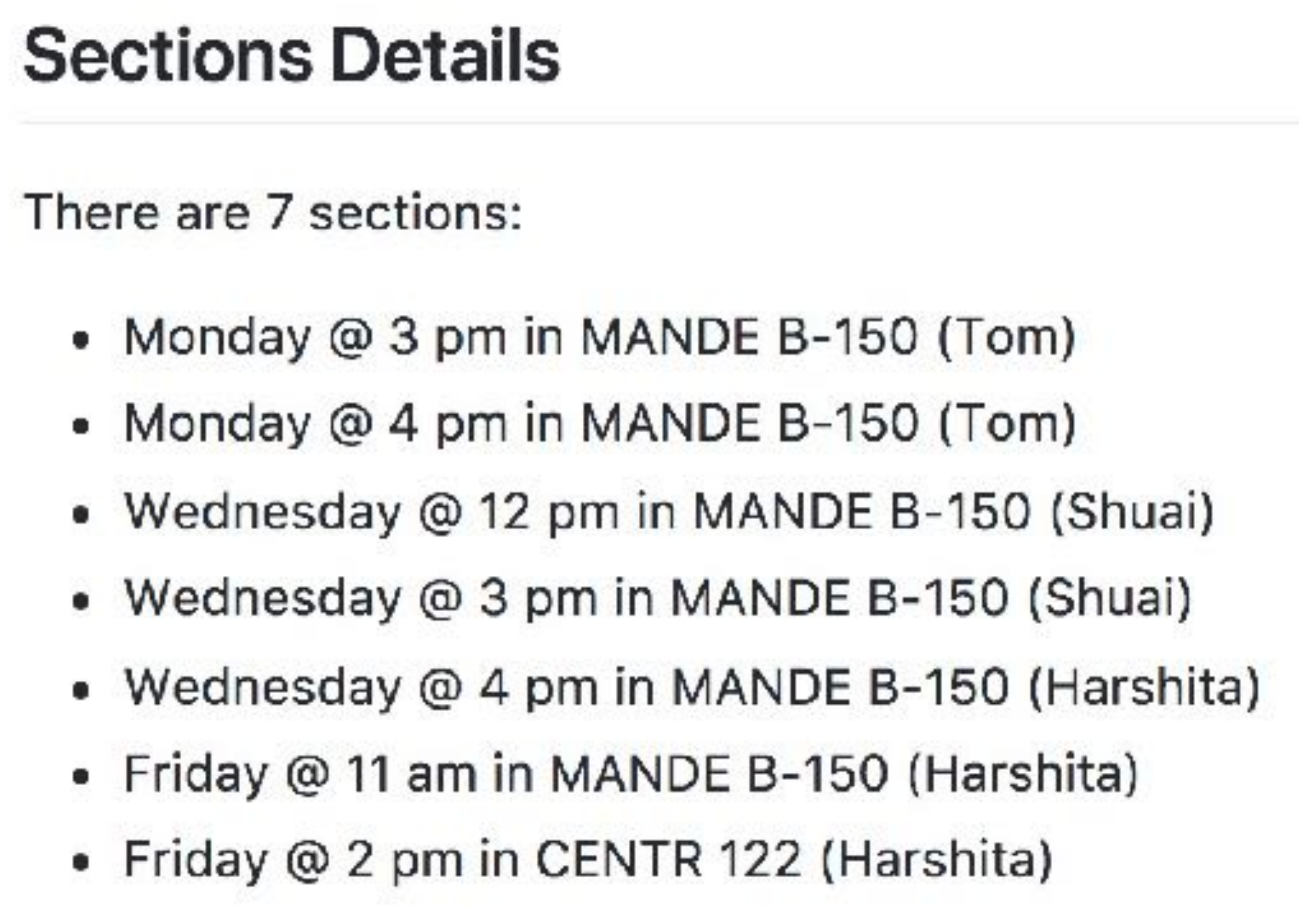
Bradley Voytek, Ph.D.

UC San Diego

Department of Cognitive Science
Neurosciences Graduate Program
Halicioglu Data Science Institute

bvoytek@ucsd.edu
@bradleyvoytek

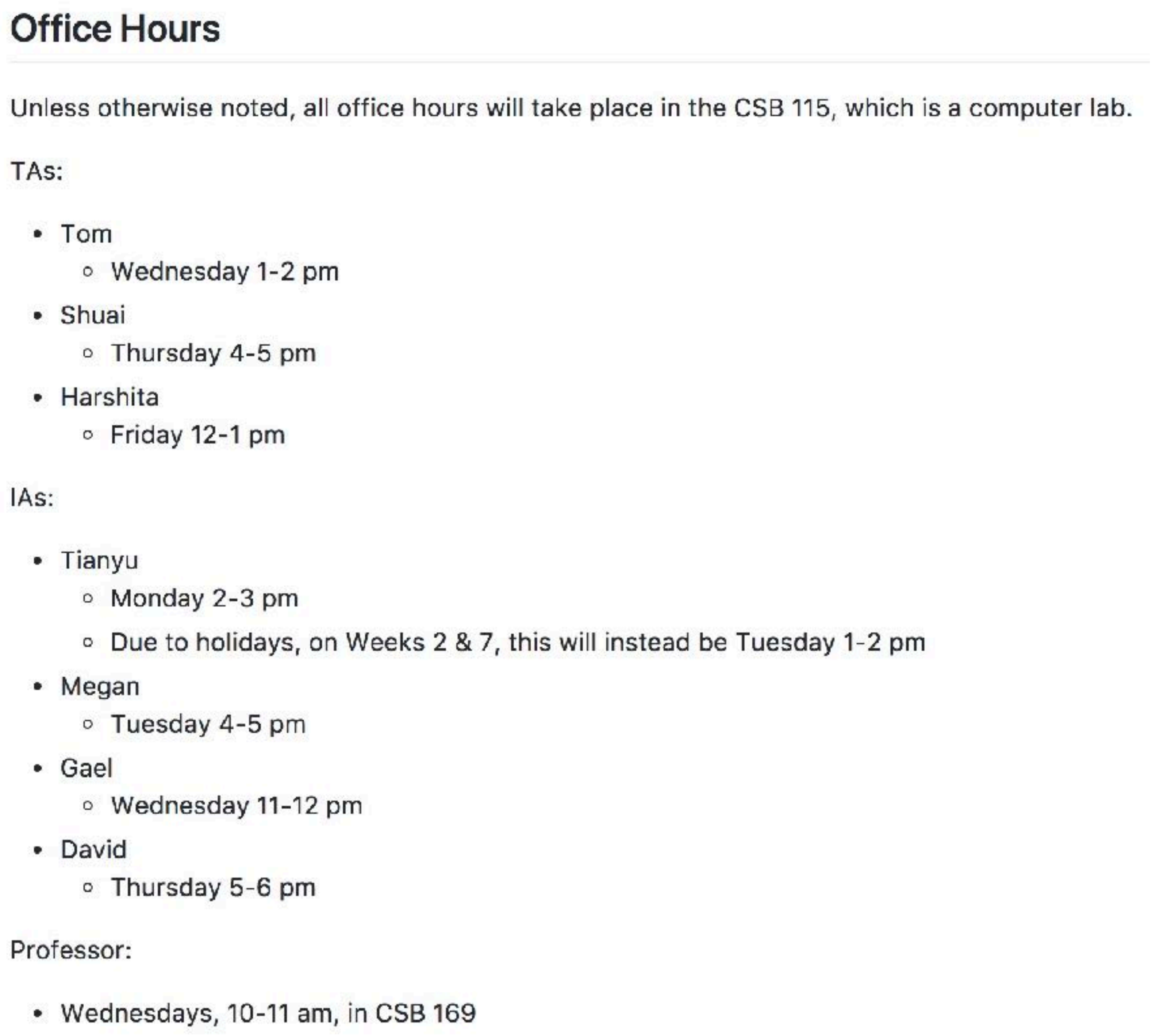
Administrative stuff



Sections Details

There are 7 sections:

- Monday @ 3 pm in MANDE B-150 (Tom)
- Monday @ 4 pm in MANDE B-150 (Tom)
- Wednesday @ 12 pm in MANDE B-150 (Shuai)
- Wednesday @ 3 pm in MANDE B-150 (Shuai)
- Wednesday @ 4 pm in MANDE B-150 (Harshita)
- Friday @ 11 am in MANDE B-150 (Harshita)
- Friday @ 2 pm in CENTR 122 (Harshita)



Office Hours

Unless otherwise noted, all office hours will take place in the CSB 115, which is a computer lab.

TAs:

- Tom
  - Wednesday 1-2 pm
- Shuai
  - Thursday 4-5 pm
- Harshita
  - Friday 12-1 pm

IAs:

- Tianyu
  - Monday 2-3 pm
  - Due to holidays, on Weeks 2 & 7, this will instead be Tuesday 1-2 pm
- Megan
  - Tuesday 4-5 pm
- Gael
  - Wednesday 11-12 pm
- David
  - Thursday 5-6 pm

Professor:

- Wednesdays, 10-11 am, in CSB 169

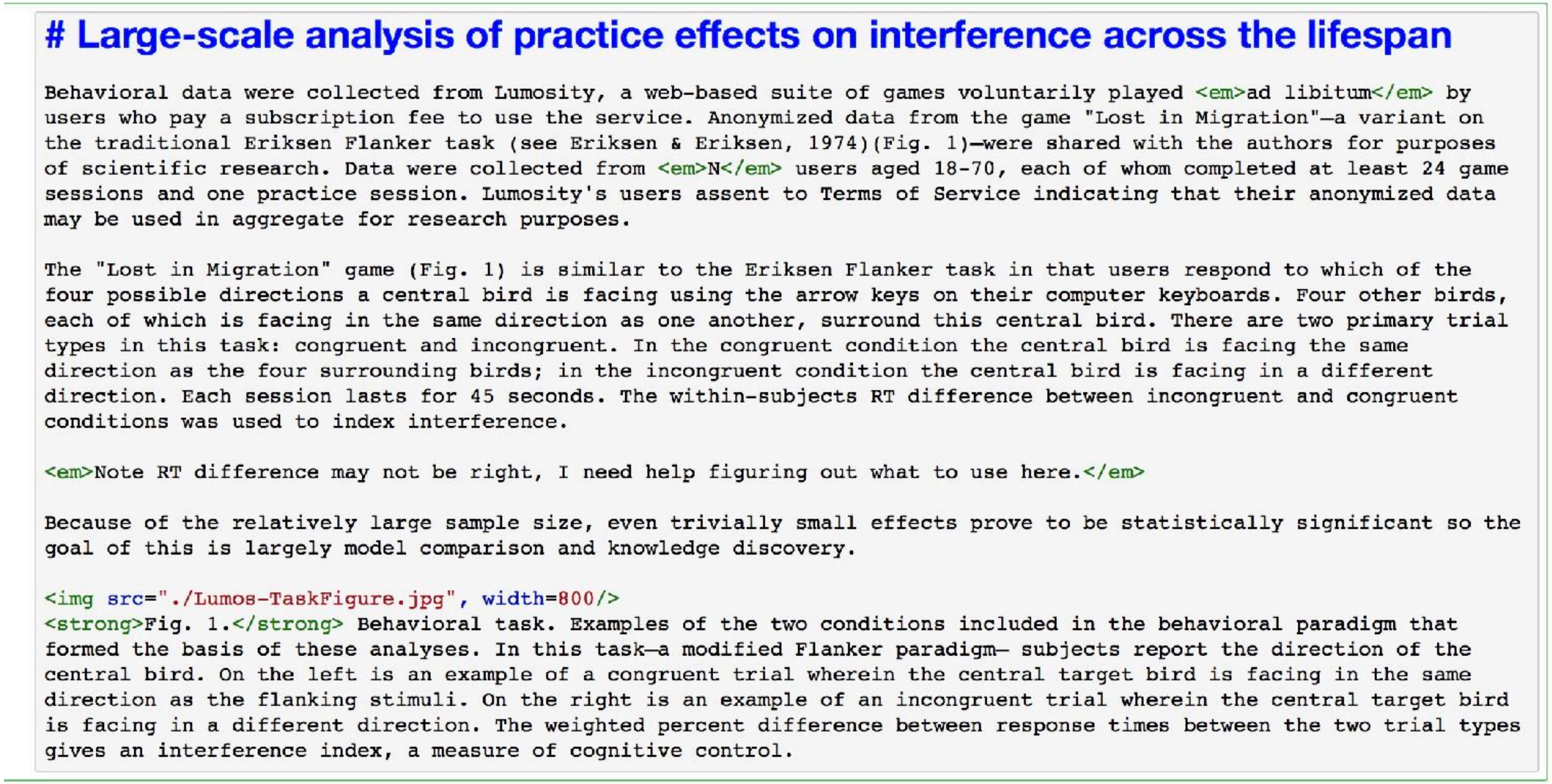
Administrative stuff

- Fun data science blogs, etc.:
  - /r/dataisbeautiful
  - Hilary Mason's blog
  - "Becoming a Data Scientist" (Data Science Renee)
  - John Myles White
  - Andrew Gelman
  - KDnuggets
  - No Free Hunch (Kaggle)

COGS 108
Data Science in Practice

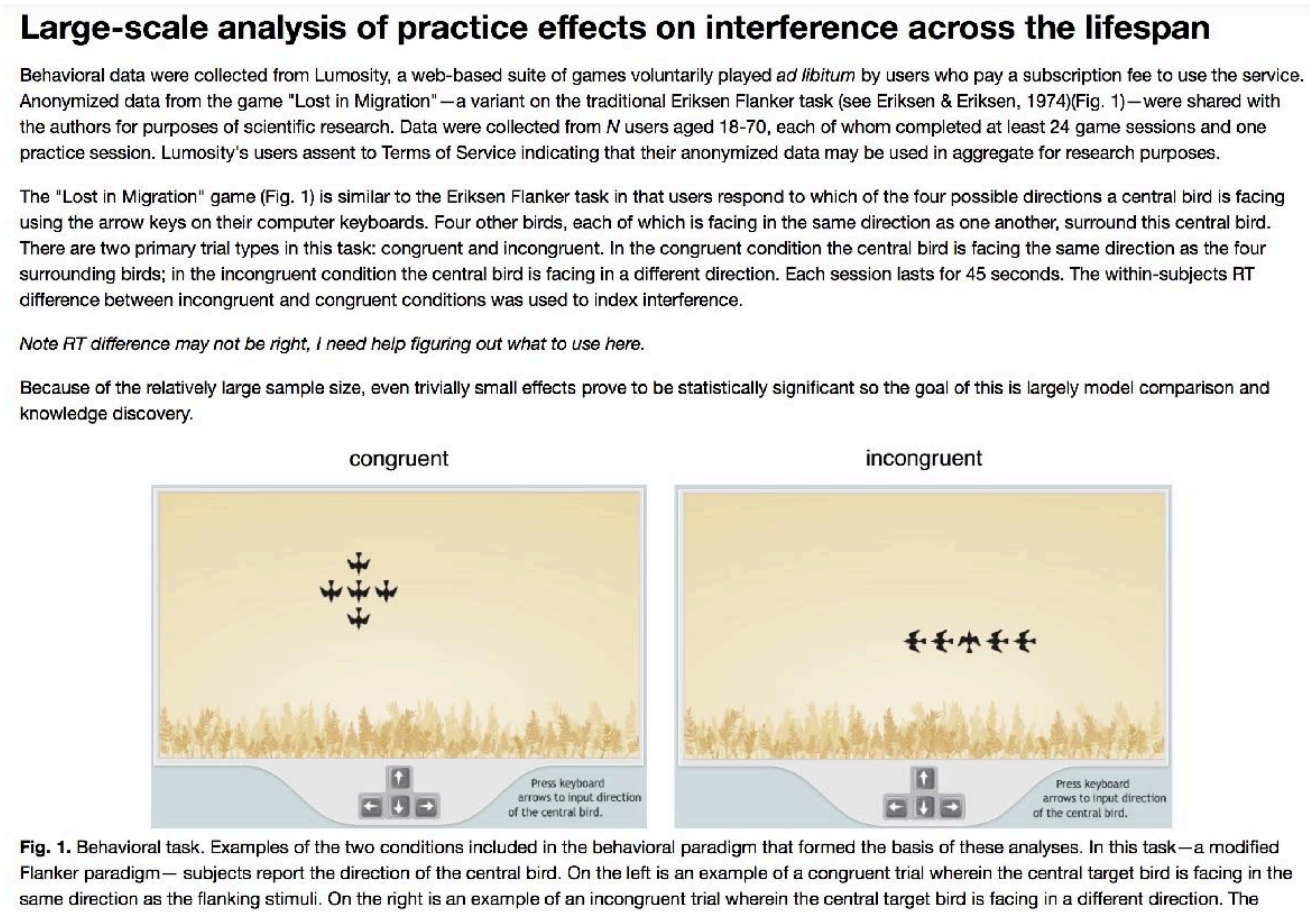
Python! (For great Data Science)

Jupyter - Markdown



```
# Large-scale analysis of practice effects on interference across the lifespan

Behavioral data were collected from Lumosity, a web-based suite of games voluntarily played <em>ad libitum</em> by
users who pay a subscription fee to use the service. Anonymized data from the game "Lost in Migration"—a variant on
the traditional Eriksen Flanker task (see Eriksen & Eriksen, 1974)(Fig. 1)—were shared with the authors for purposes
of scientific research. Data were collected from <em>N</em> users aged 18-70, each of whom completed at least 24 game
sessions and one practice session. Lumosity's users assent to Terms of Service indicating that their anonymized data
may be used in aggregate for research purposes.

The "Lost in Migration" game (Fig. 1) is similar to the Eriksen Flanker task in that users respond to which of the
four possible directions a central bird is facing using the arrow keys on their computer keyboards. Four other birds,
each of which is facing in the same direction as one another, surround this central bird. There are two primary trial
types in this task: congruent and incongruent. In the congruent condition the central bird is facing the same
direction as the four surrounding birds; in the incongruent condition the central bird is facing in a different
direction. Each session lasts for 45 seconds. The within-subjects RT difference between incongruent and congruent
conditions was used to index interference.

<em>Note RT difference may not be right, I need help figuring out what to use here.</em>

Because of the relatively large sample size, even trivially small effects prove to be statistically significant so the
goal of this is largely model comparison and knowledge discovery.

<img src="./Lumos-TaskFigure.jpg", width=800/>
<strong>Fig. 1.</strong> Behavioral task. Examples of the two conditions included in the behavioral paradigm that
formed the basis of these analyses. In this task—a modified Flanker paradigm— subjects report the direction of the
central bird. On the left is an example of a congruent trial wherein the central target bird is facing in the same
direction as the flanking stimuli. On the right is an example of an incongruent trial wherein the central target bird
is facing in a different direction. The weighted percent difference between response times between the two trial types
gives an interference index, a measure of cognitive control.
```

Jupyter - Markdown



Large-scale analysis of practice effects on interference across the lifespan

Behavioral data were collected from Lumosity, a web-based suite of games voluntarily played *ad libitum* by users who pay a subscription fee to use the service. Anonymized data from the game "Lost in Migration"—a variant on the traditional Eriksen Flanker task (see Eriksen & Eriksen, 1974)(Fig. 1)—were shared with the authors for purposes of scientific research. Data were collected from *N* users aged 18-70, each of whom completed at least 24 game sessions and one practice session. Lumosity's users assent to Terms of Service indicating that their anonymized data may be used in aggregate for research purposes.

The "Lost in Migration" game (Fig. 1) is similar to the Eriksen Flanker task in that users respond to which of the four possible directions a central bird is facing using the arrow keys on their computer keyboards. Four other birds, each of which is facing in the same direction as one another, surround this central bird. There are two primary trial types in this task: congruent and incongruent. In the congruent condition the central bird is facing the same direction as the four surrounding birds; in the incongruent condition the central bird is facing in a different direction. Each session lasts for 45 seconds. The within-subjects RT difference between incongruent and congruent conditions was used to index interference.

Note RT difference may not be right, I need help figuring out what to use here.

Because of the relatively large sample size, even trivially small effects prove to be statistically significant so the goal of this is largely model comparison and knowledge discovery.

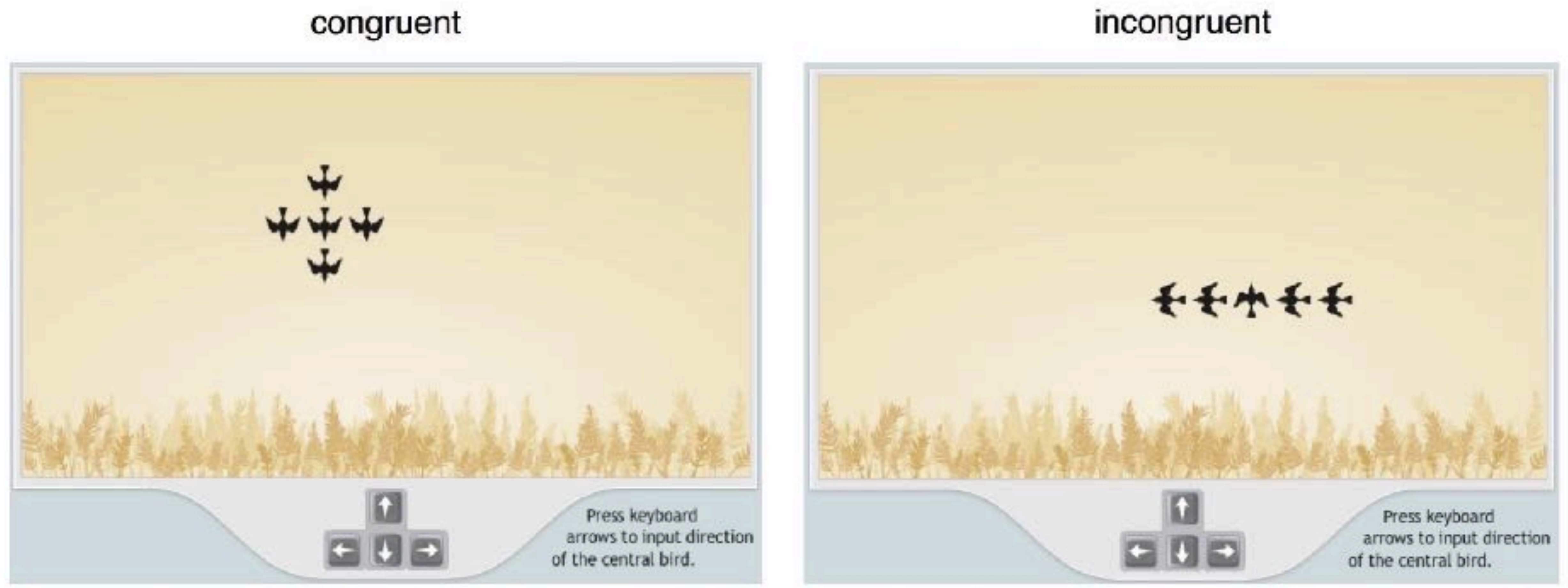
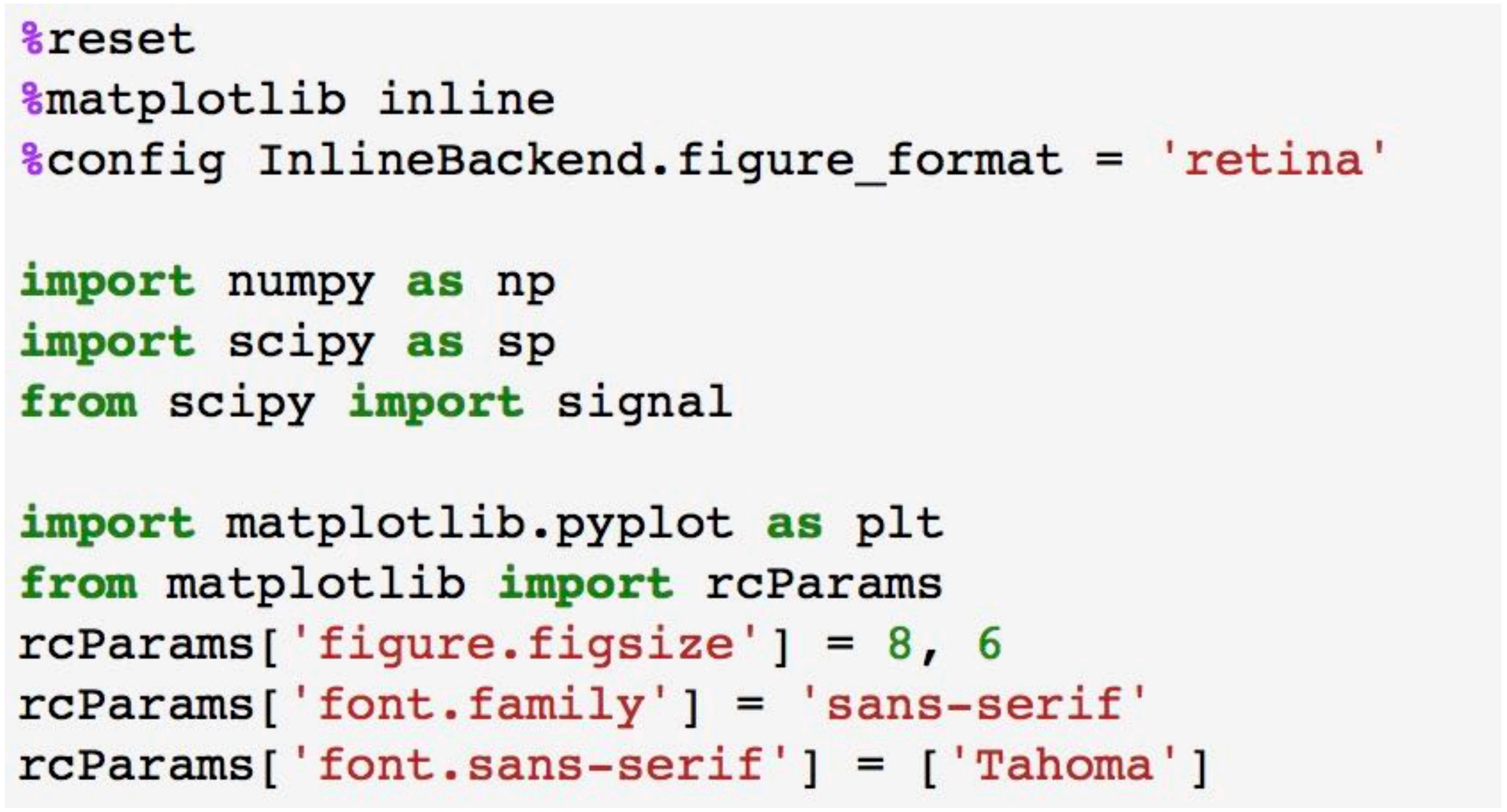

congruent
incongruent
Press keyboard arrows to input direction of the central bird.
Press keyboard arrows to input direction of the central bird.

Fig. 1. Behavioral task. Examples of the two conditions included in the behavioral paradigm that formed the basis of these analyses. In this task—a modified Flanker paradigm— subjects report the direction of the central bird. On the left is an example of a congruent trial wherein the central target bird is facing in the same direction as the flanking stimuli. On the right is an example of an incongruent trial wherein the central target bird is facing in a different direction. The

Jupyter - Beginning an analysis



```
%reset
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import scipy as sp
from scipy import signal

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

Jupyter - Beginning an analysis

magic to clear all variables

```
%reset
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import scipy as sp
from scipy import signal

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

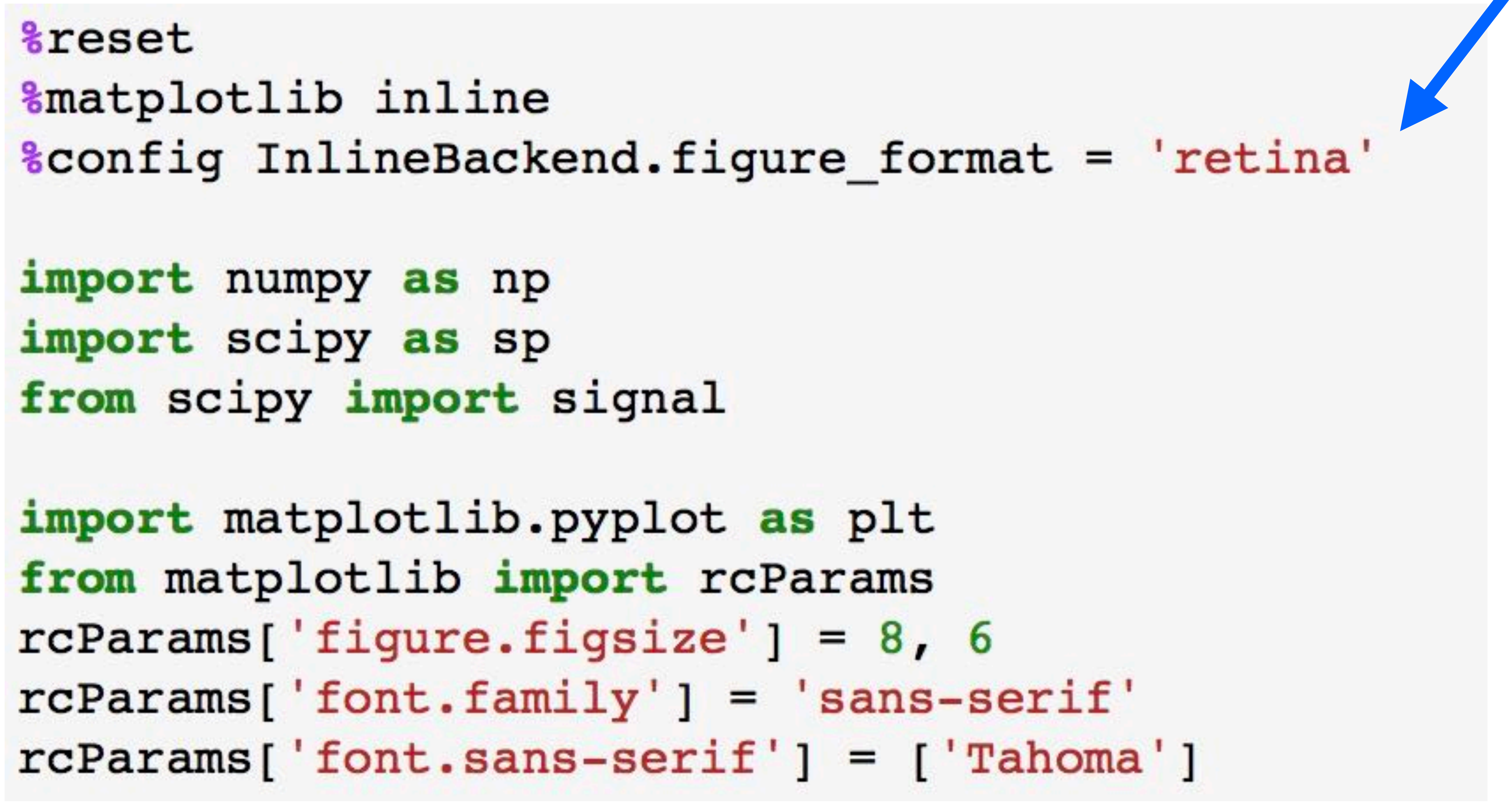
Jupyter - Beginning an analysis

magic to allow inline plotting

```
%reset
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import scipy as sp
from scipy import signal

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

Jupyter - Beginning an analysis

magic for high resolution figures



```
%reset
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import scipy as sp
from scipy import signal

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

UC San Diego

Jupyter - retina resolution

ICK

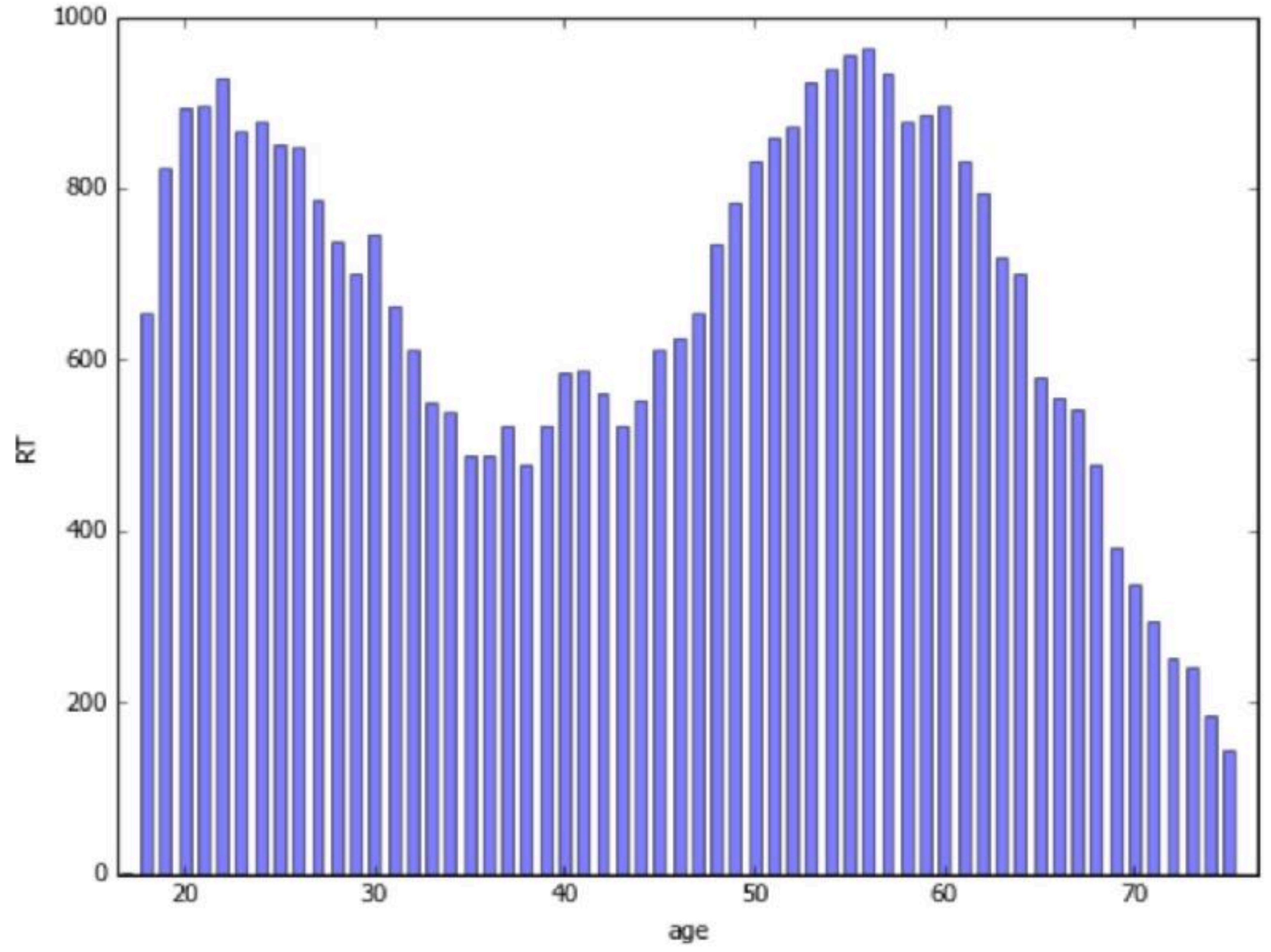


Fig. 2

Jupyter - retina resolution

ICK

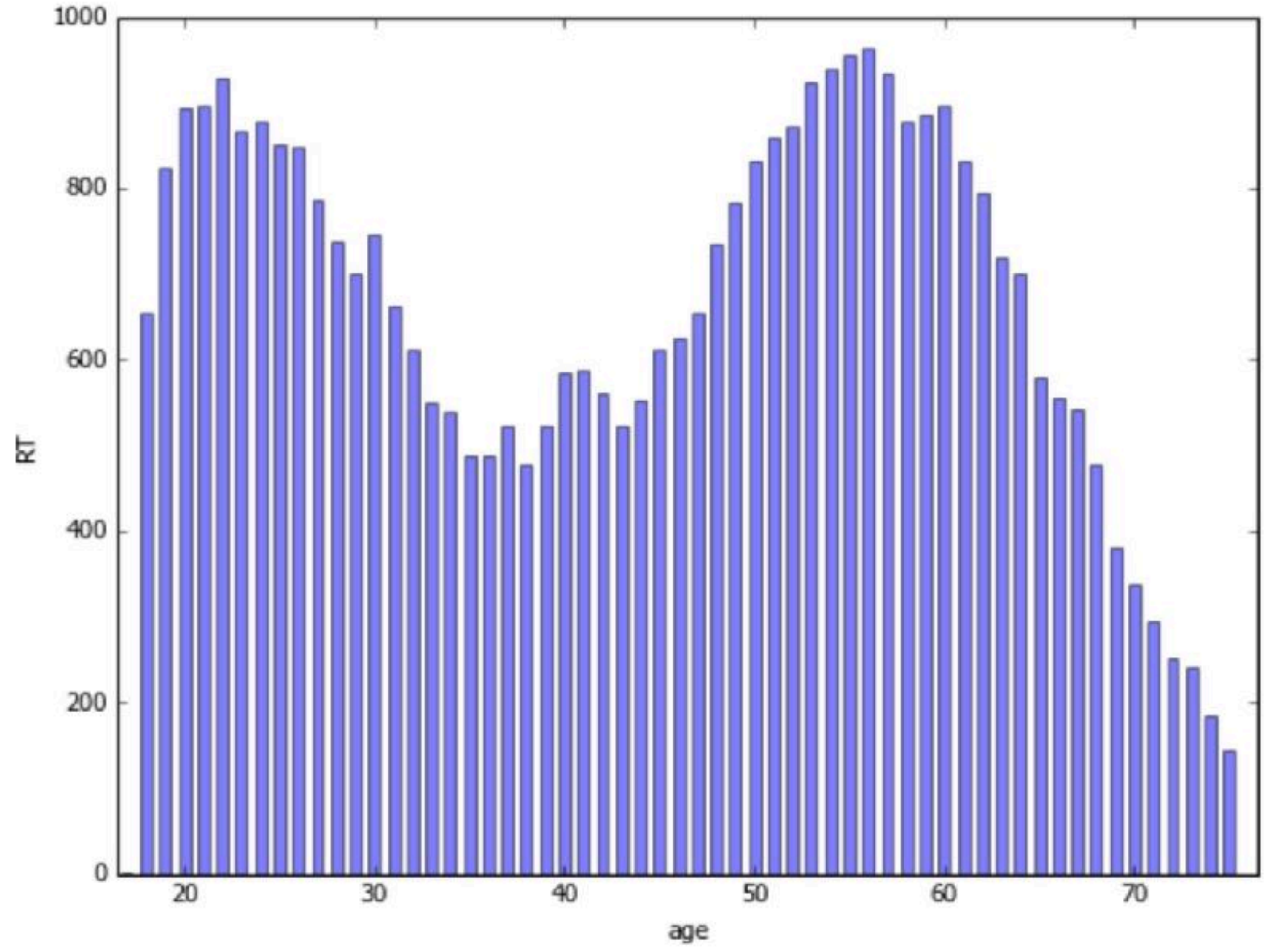


Fig. 2

YAY

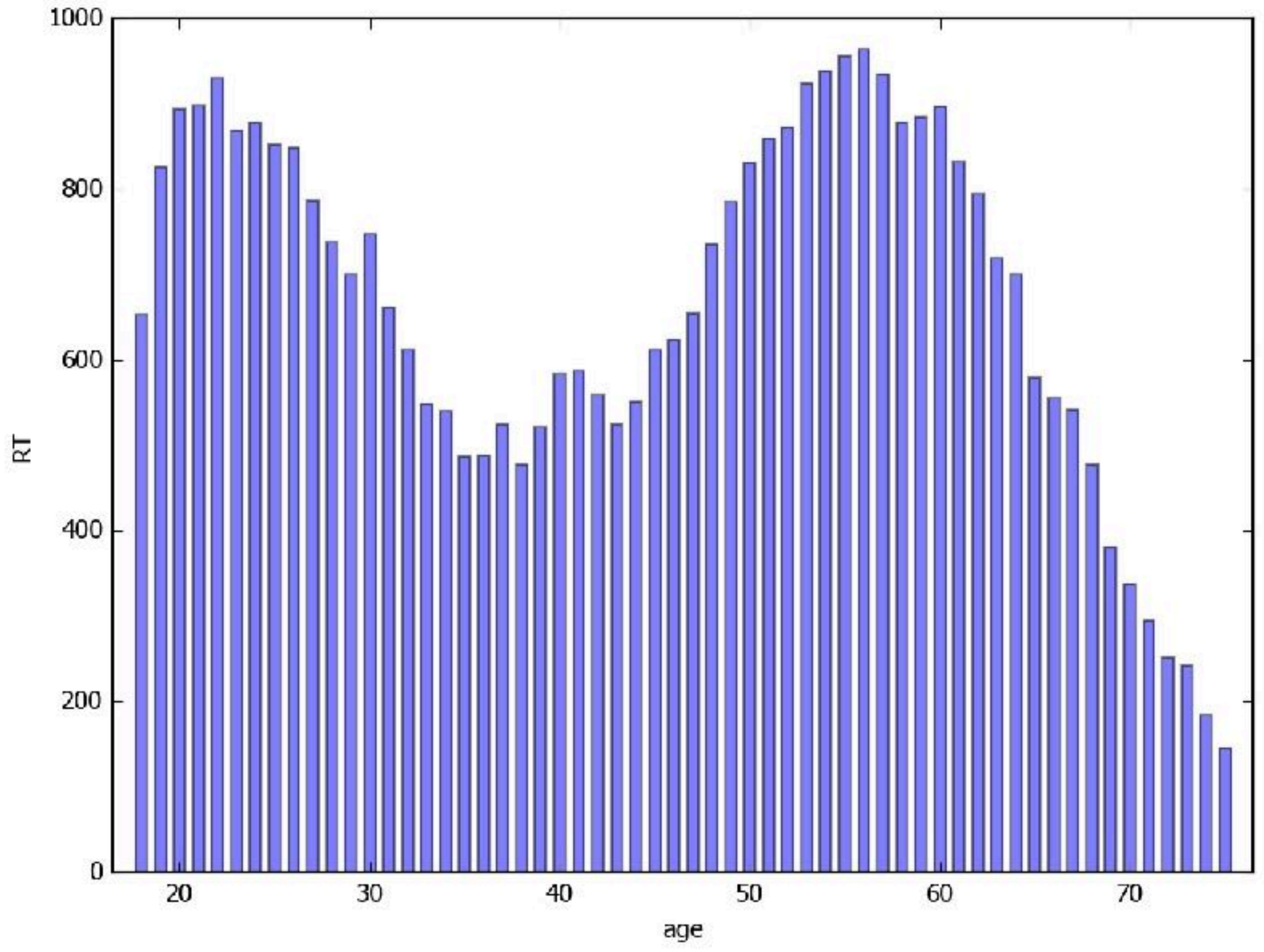
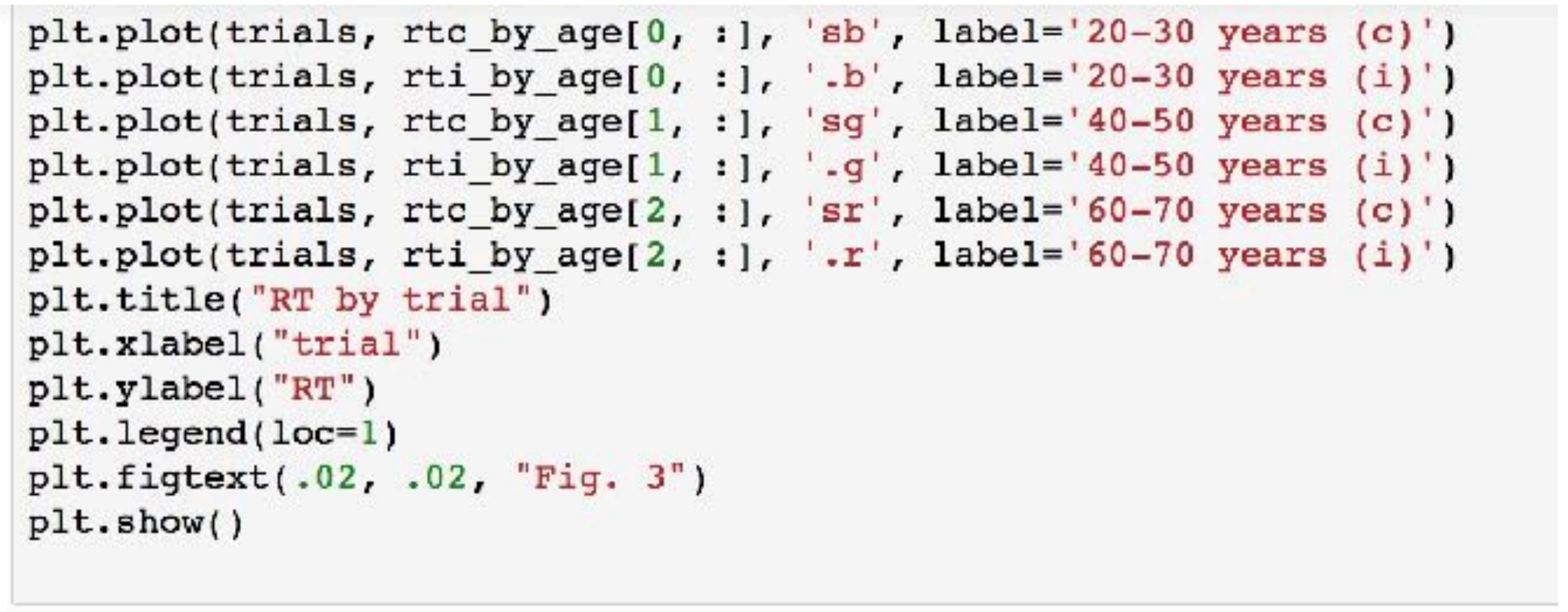


Fig. 2

UC San Diego

Jupyter - Plots in-line!



```
plt.plot(trials, rtc_by_age[0, :], 'sb', label='20-30 years (c)')
plt.plot(trials, rti_by_age[0, :], '.b', label='20-30 years (i)')
plt.plot(trials, rtc_by_age[1, :], 'sg', label='40-50 years (c)')
plt.plot(trials, rti_by_age[1, :], '.g', label='40-50 years (i)')
plt.plot(trials, rtc_by_age[2, :], 'sr', label='60-70 years (c)')
plt.plot(trials, rti_by_age[2, :], '.r', label='60-70 years (i)')
plt.title("RT by trial")
plt.xlabel("trial")
plt.ylabel("RT")
plt.legend(loc=1)
plt.figtext(.02, .02, "Fig. 3")
plt.show()
```

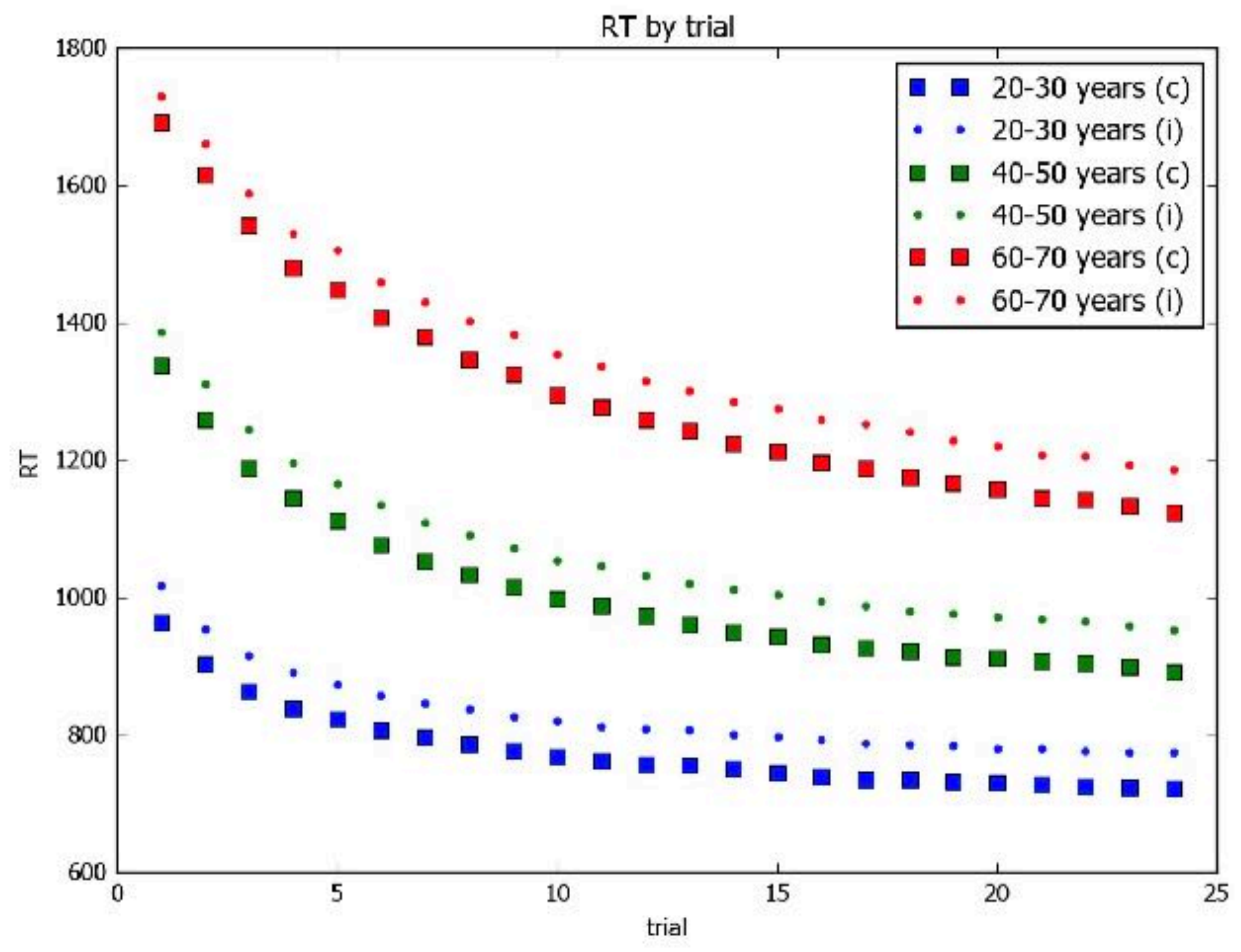
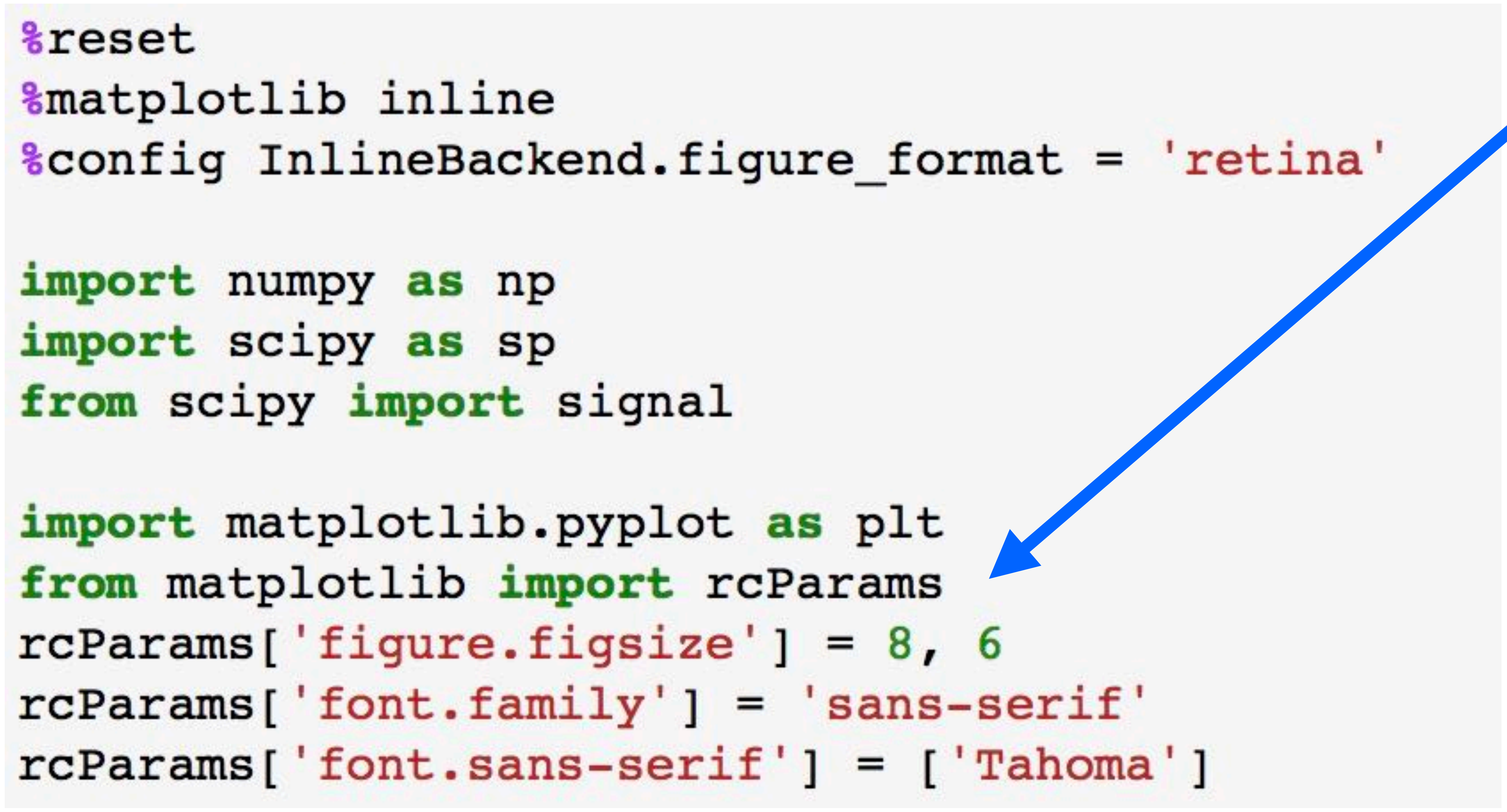


Fig. 3

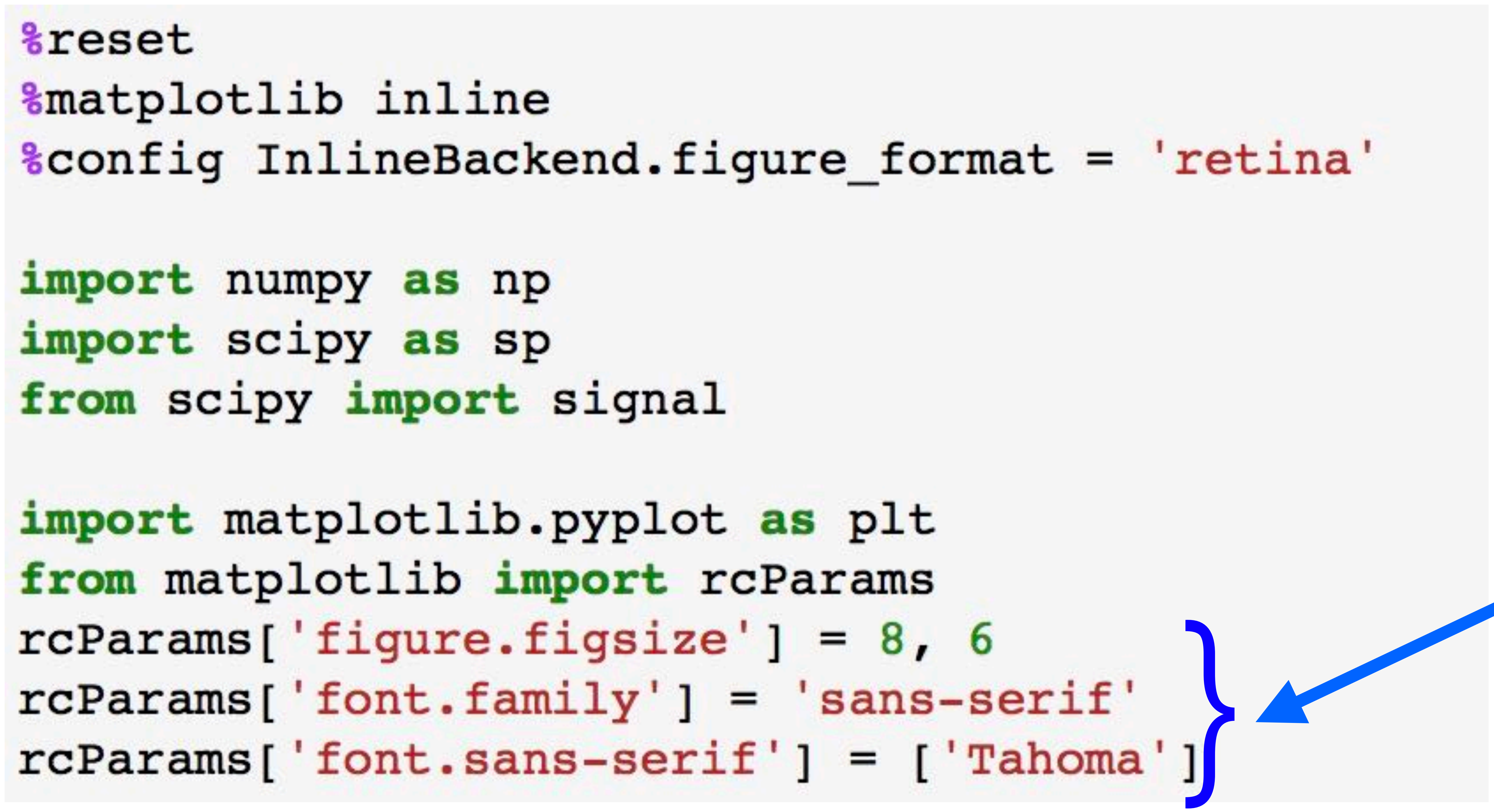
Jupyter - Beginning an analysis



```
%reset
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import scipy as sp
from scipy import signal

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

plotting parameters

Jupyter - Beginning an analysis



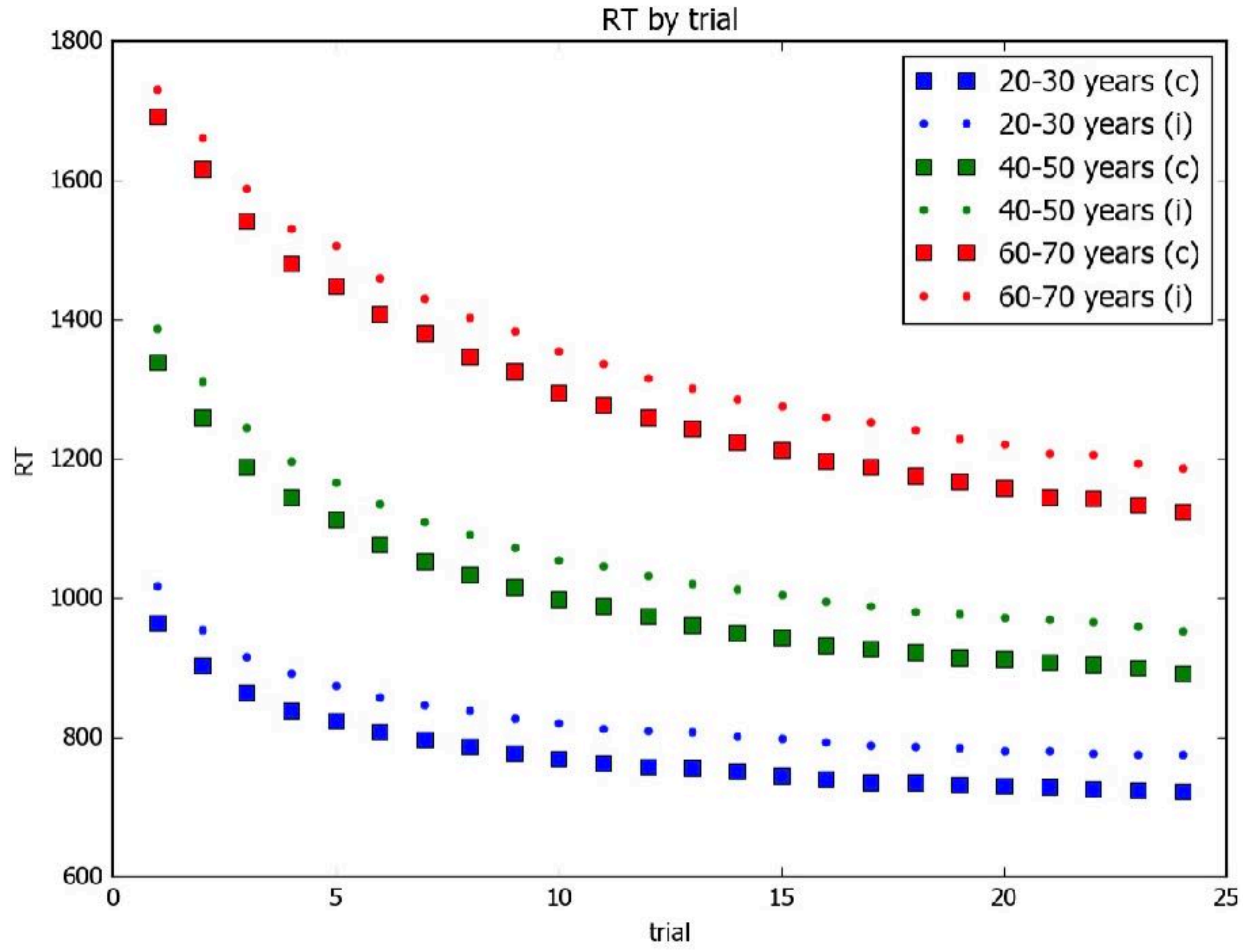
```
%reset
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import scipy as sp
from scipy import signal

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

figure parameters

UC San Diego

Jupyter - Figure parameters

```
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

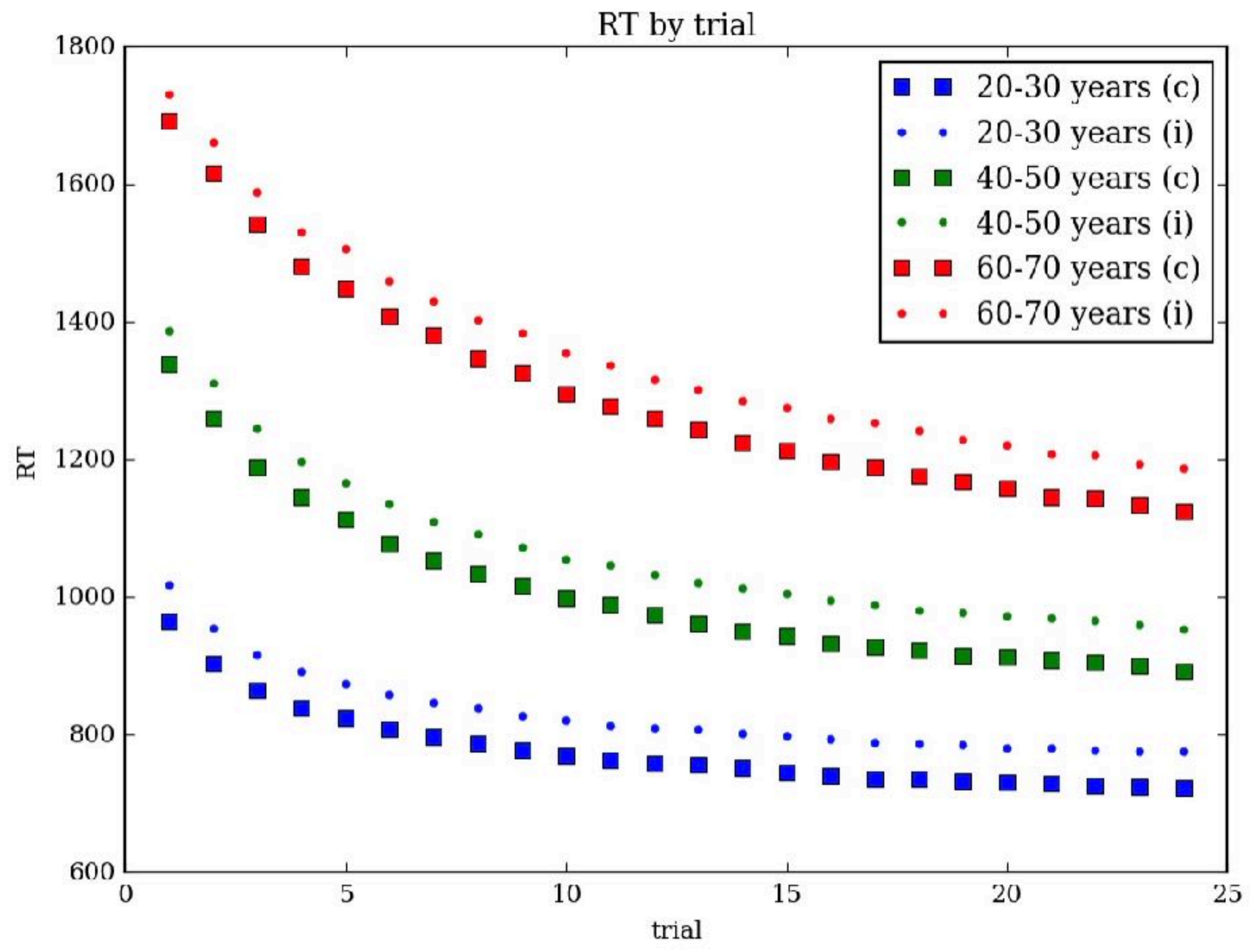


Jupyter - Figure parameters

~~rcParams['figure.figsize'] = 8, 6~~
~~rcParams['font.family'] = 'sans-serif'~~
~~rcParams['font.sans-serif'] = ['Tahoma']~~

```
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

serif font now

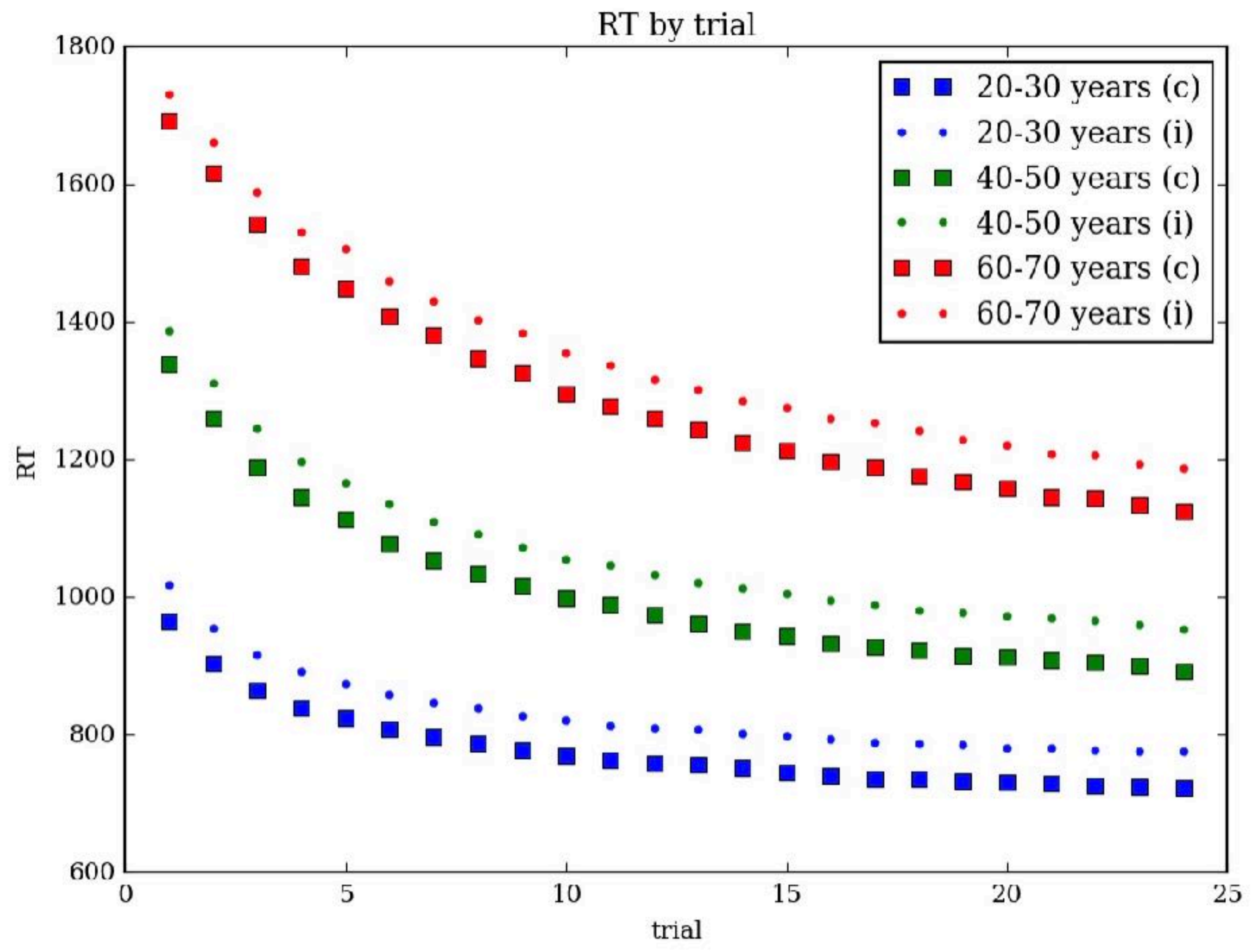


Jupyter - Figure parameters

~~rcParams['figure.figsize'] = 8, 6~~
~~rcParams['font.family'] = 'sans-serif'~~
~~rcParams['font.sans-serif'] = ['Tahoma']~~

```
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

despite this saying sans serif



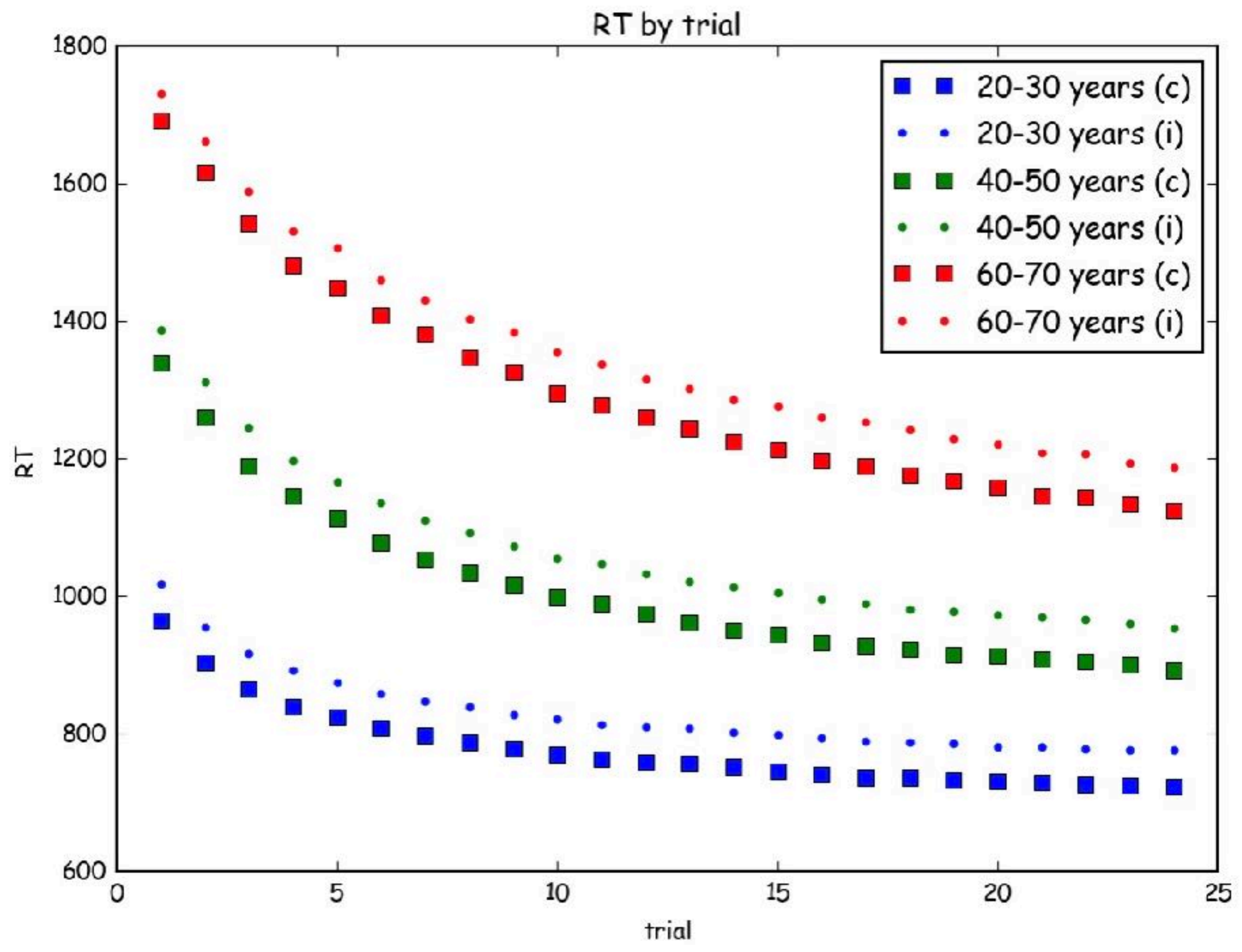
Jupyter - Figure parameters

```
~~rcParams['figure.figsize'] = 8, 6~~
~~rcParams['font.family'] = 'sans-serif'~~
~~rcParams['font.sans-serif'] = ['Tahoma']~~
```

```
~~rcParams['figure.figsize'] = 8, 6~~
~~rcParams['font.family'] = 'serif'~~
~~rcParams['font.sans-serif'] = ['Tahoma']~~
```

```
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = 'Comic Sans MS'
```

comic sans ftw!



Jupyter - Figure parameters

NOTE! I didn't restart the jupyter kernel before plotting again, meaning it's still plotting in comic sans!

~~rcParams['figure.figsize'] = 8, 6~~
~~rcParams['font.family'] = 'sans-serif'~~
~~rcParams['font.sans-serif'] = 'Comic Sans MS'~~

```
rcParams['figure.figsize'] = 4, 3
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

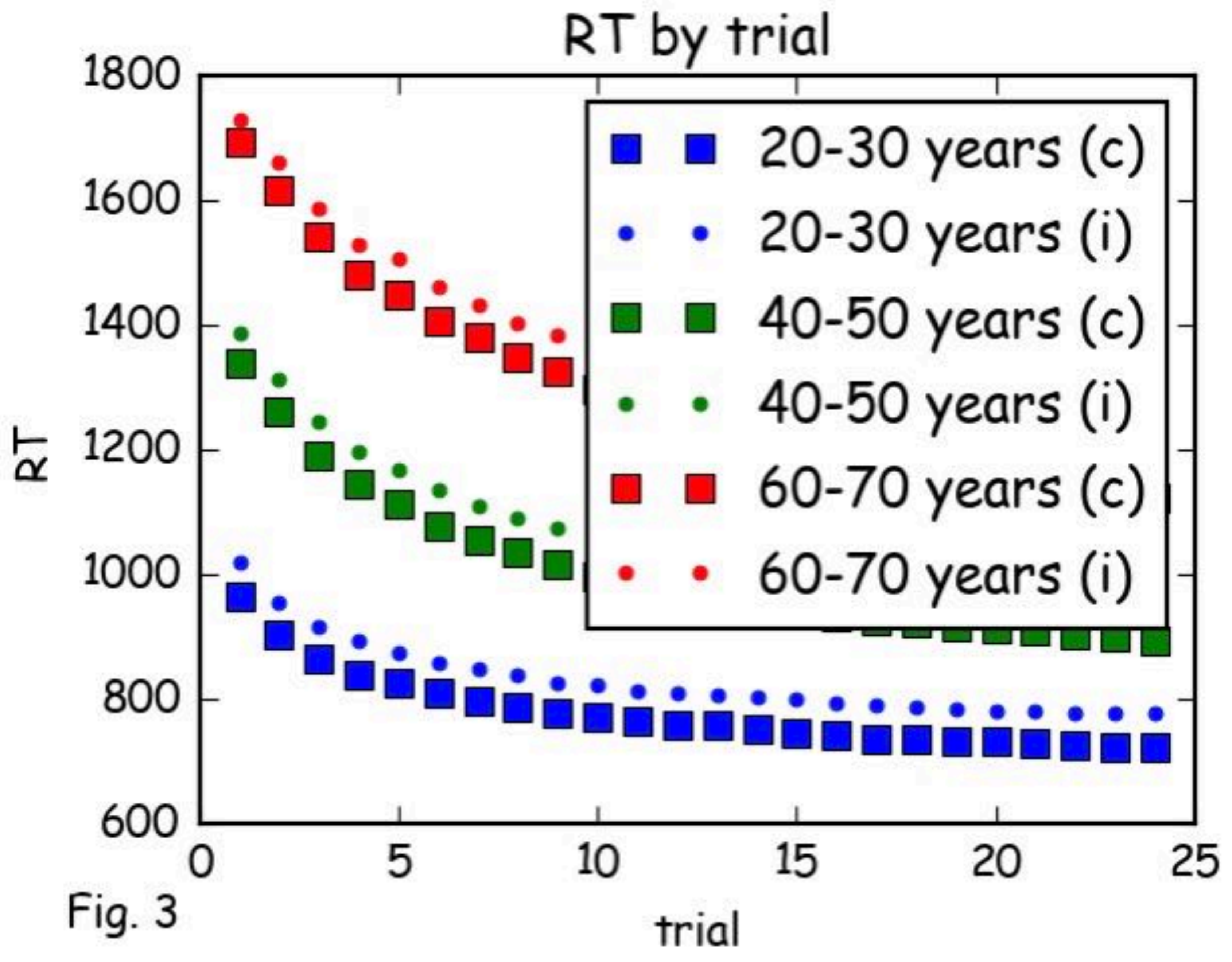
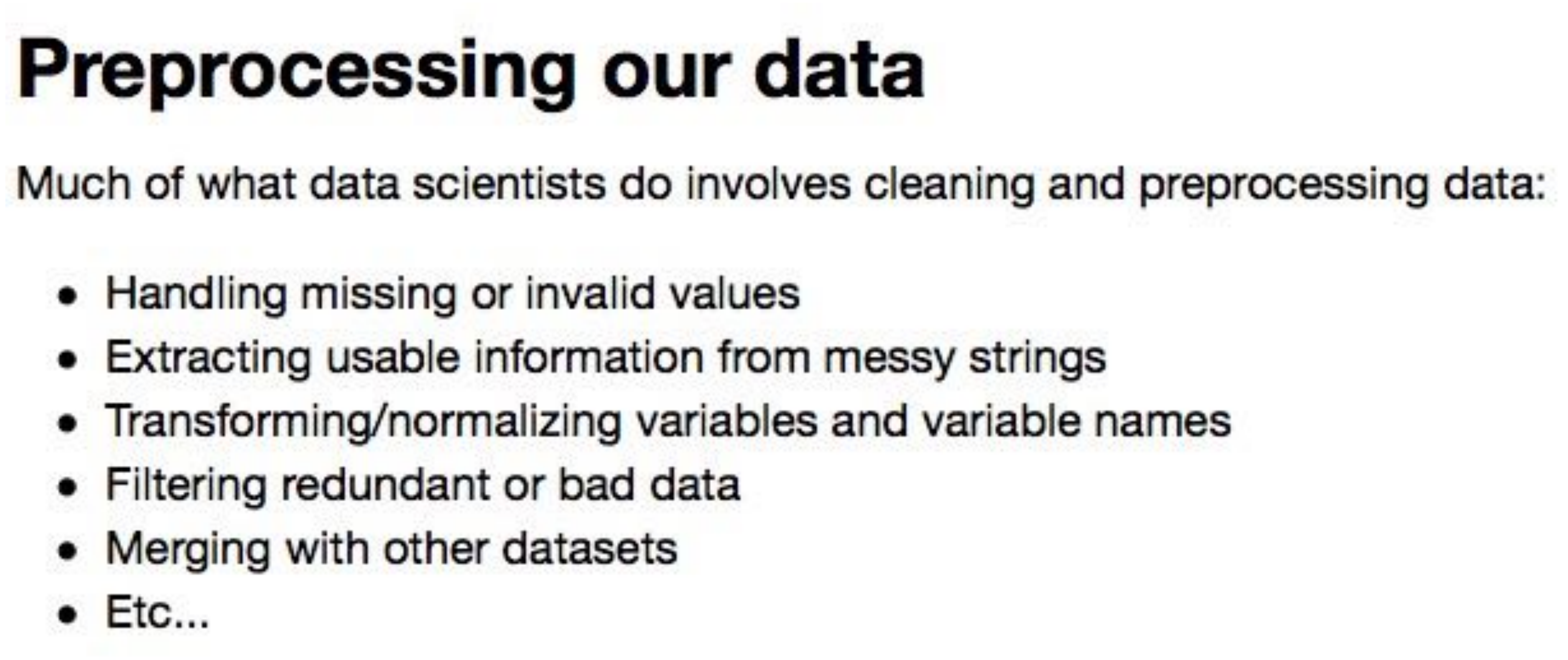


Fig. 3

Pandas



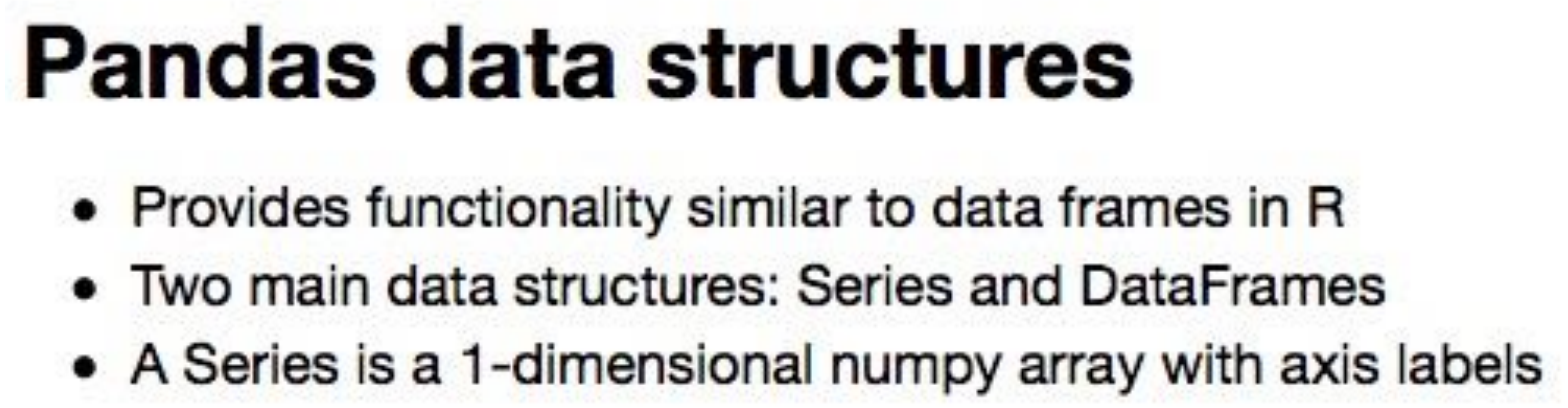
Preprocessing our data

Much of what data scientists do involves cleaning and preprocessing data:

- Handling missing or invalid values
- Extracting usable information from messy strings
- Transforming/normalizing variables and variable names
- Filtering redundant or bad data
- Merging with other datasets
- Etc...

Source: Tal Yarkoni

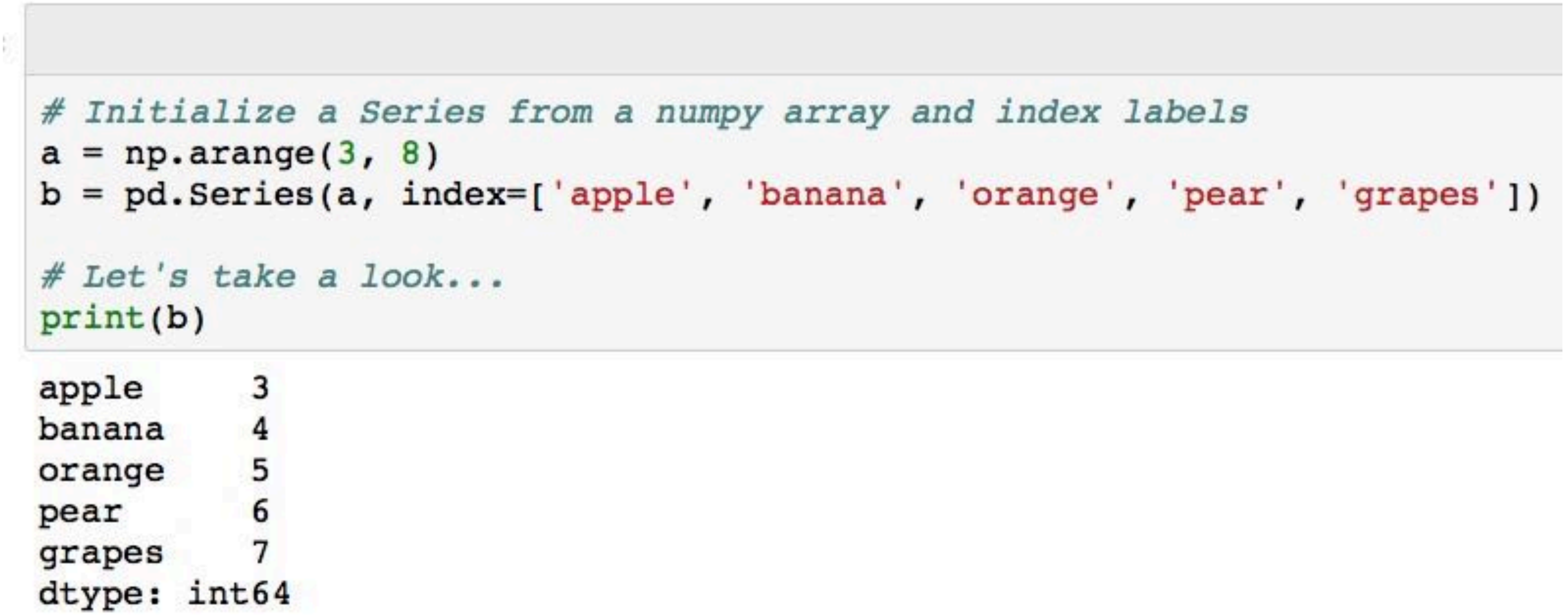
Pandas



Pandas data structures

- Provides functionality similar to data frames in R
- Two main data structures: Series and DataFrames
- A Series is a 1-dimensional numpy array with axis labels

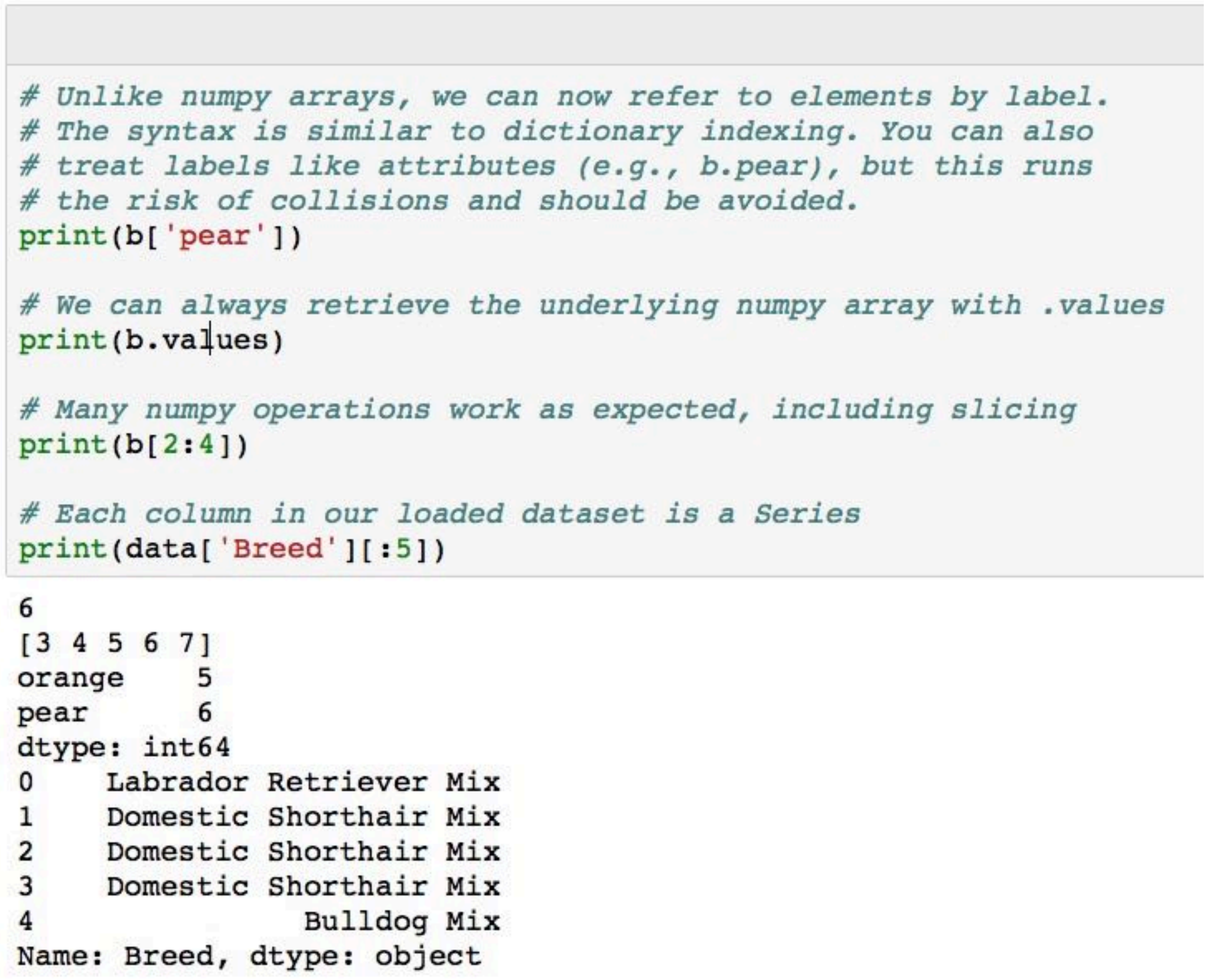
Pandas



```python
# Initialize a Series from a numpy array and index labels
a = np.arange(3, 8)
b = pd.Series(a, index=['apple', 'banana', 'orange', 'pear', 'grapes'])

# Let's take a look...
print(b)
```

```
apple     3
banana    4
orange    5
pear      6
grapes    7
dtype: int64
```

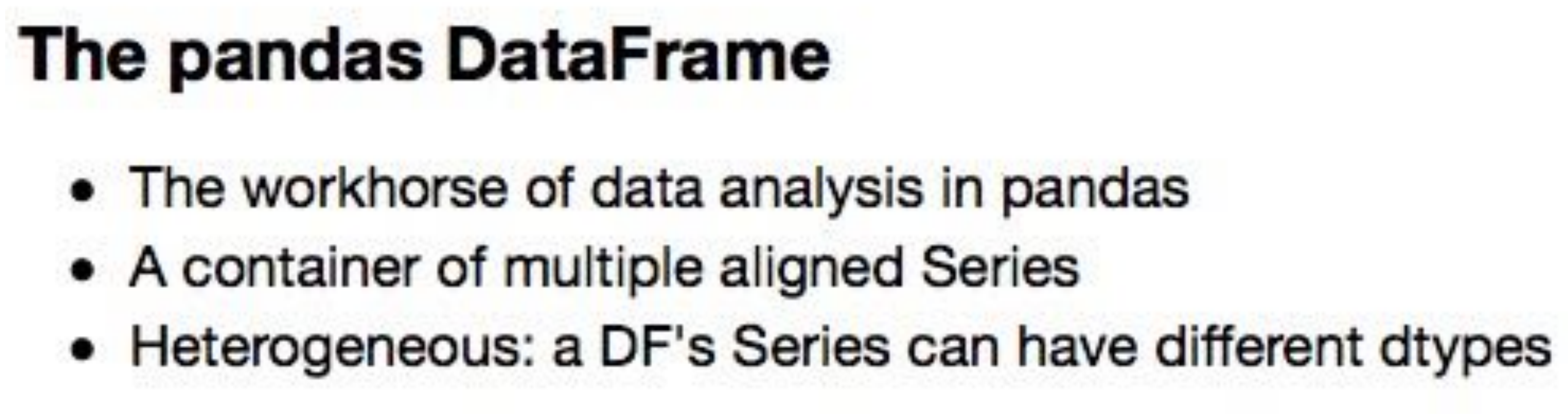
Pandas



```
# Unlike numpy arrays, we can now refer to elements by label.
# The syntax is similar to dictionary indexing. You can also
# treat labels like attributes (e.g., b.pear), but this runs
# the risk of collisions and should be avoided.
print(b['pear'])

# We can always retrieve the underlying numpy array with .values
print(b.values)

# Many numpy operations work as expected, including slicing
print(b[2:4])

# Each column in our loaded dataset is a Series
print(data['Breed'][:5])
```

```
6
[3 4 5 6 7]
orange    5
pear      6
dtype: int64
0     Labrador Retriever Mix
1     Domestic Shorthair Mix
2     Domestic Shorthair Mix
3     Domestic Shorthair Mix
4                Bulldog Mix
Name: Breed, dtype: object
```

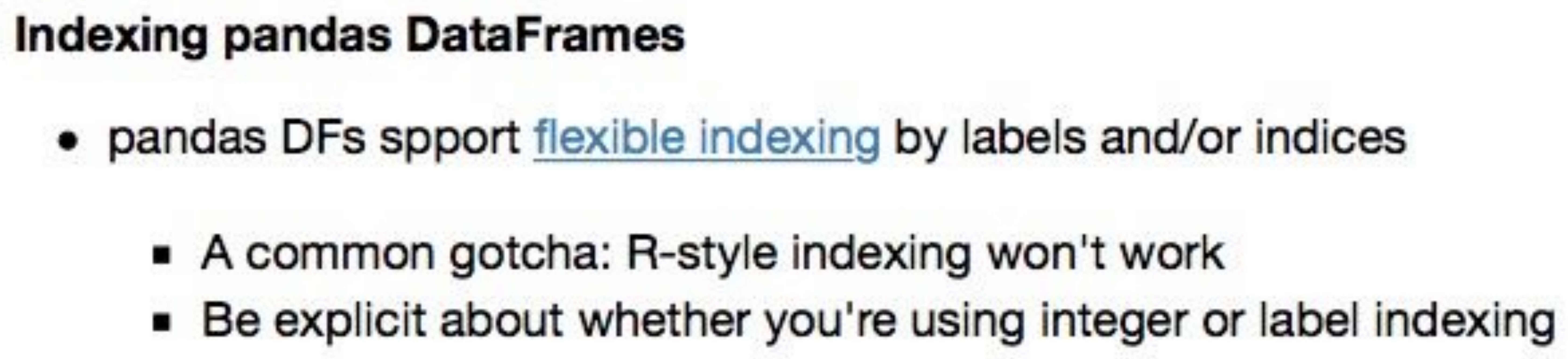
Pandas



The pandas DataFrame

- The workhorse of data analysis in pandas
- A container of multiple aligned Series
- Heterogeneous: a DF's Series can have different dtypes

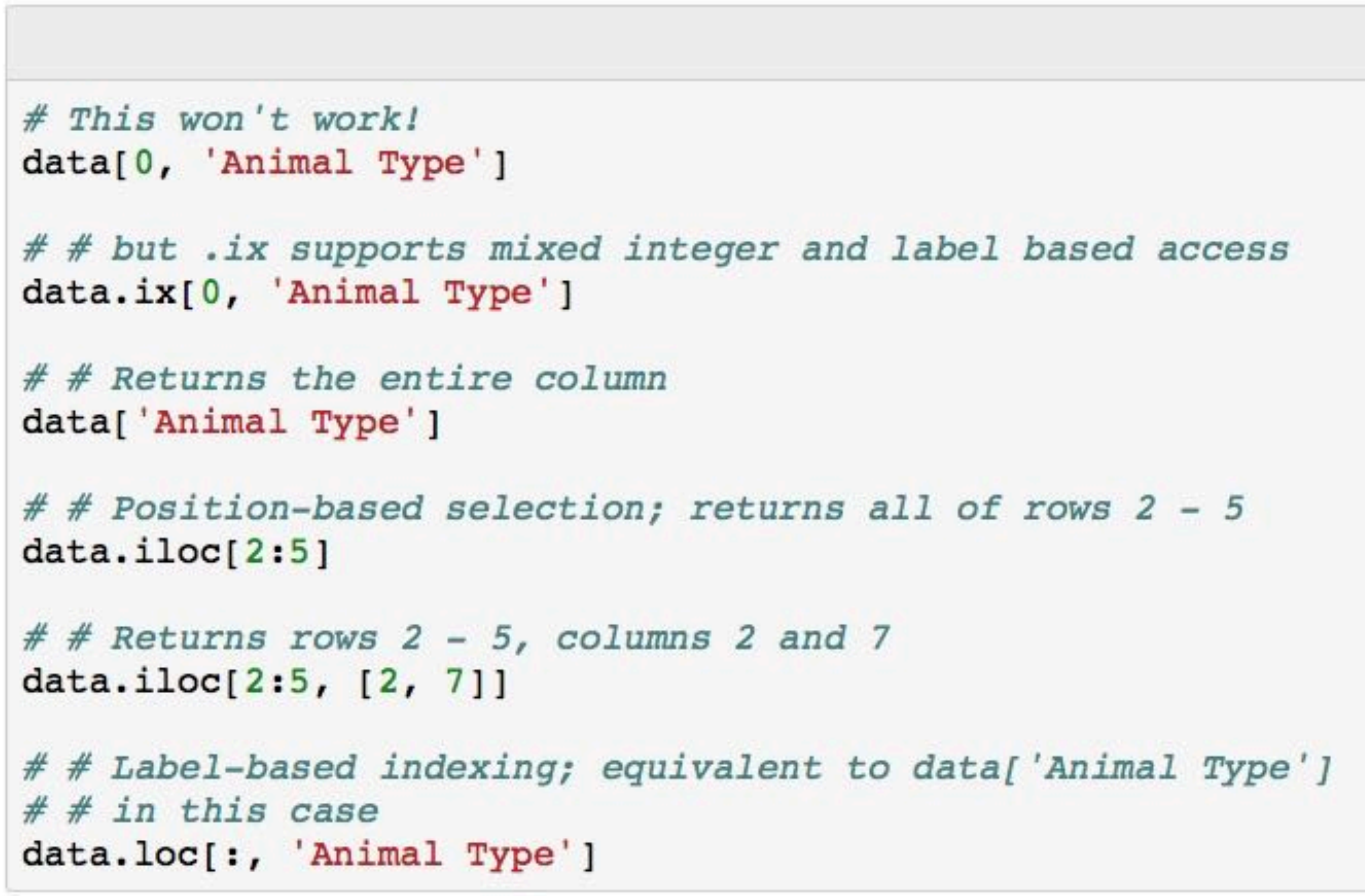
Pandas



Indexing pandas DataFrames

- pandas DFs spport flexible indexing by labels and/or indices
    - A common gotcha: R-style indexing won't work
    - Be explicit about whether you're using integer or label indexing

Pandas

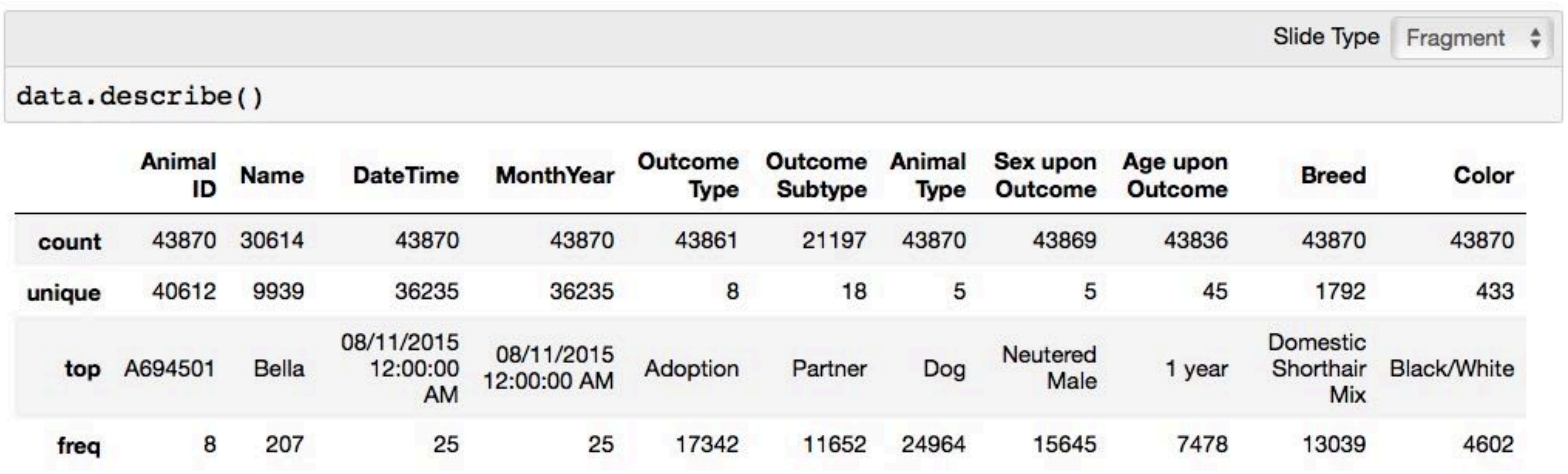


```
# This won't work!
data[0, 'Animal Type']

# # but .ix supports mixed integer and label based access
data.ix[0, 'Animal Type']

# # Returns the entire column
data['Animal Type']

# # Position-based selection; returns all of rows 2 - 5
data.iloc[2:5]

# # Returns rows 2 - 5, columns 2 and 7
data.iloc[2:5, [2, 7]]

# # Label-based indexing; equivalent to data['Animal Type']
# # in this case
data.loc[:, 'Animal Type']
```

UC San Diego

Source: Tal Yarkoni

Pandas



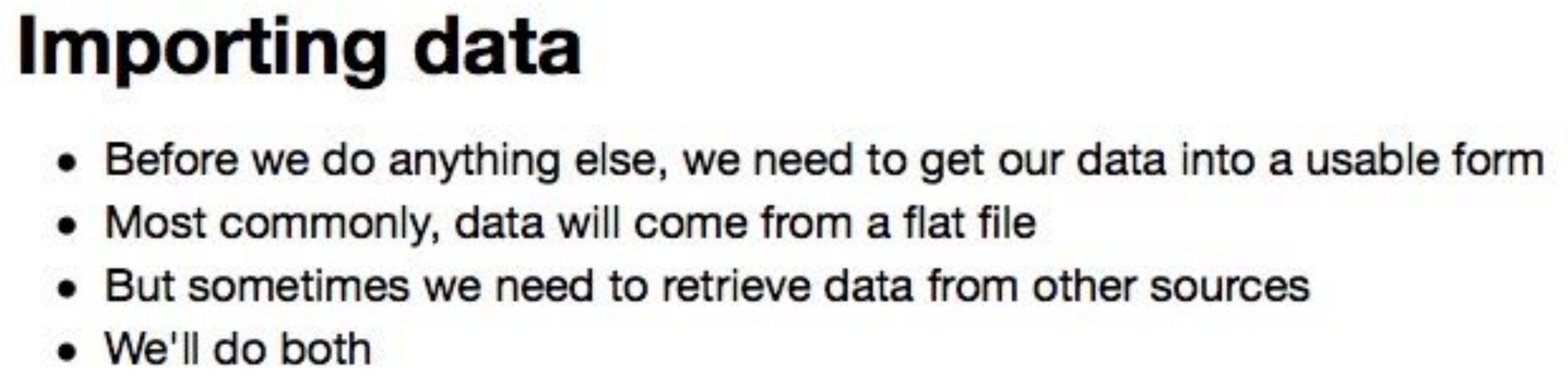
Slide Type Fragment

```
data.describe()
```

	Animal ID	Name	DateTime	MonthYear	Outcome Type	Outcome Subtype	Animal Type	Sex upon Outcome	Age upon Outcome	Breed	Color
count	43870	30614	43870	43870	43861	21197	43870	43869	43836	43870	43870
unique	40612	9939	36235	36235	8	18	5	5	45	1792	433
top	A694501	Bella	08/11/2015 12:00:00 AM	08/11/2015 12:00:00 AM	Adoption	Partner	Dog	Neutered Male	1 year	Domestic Shorthair Mix	Black/White
freq	8	207	25	25	17342	11652	24964	15645	7478	13039	4602

Source: Tal Yarkoni

Pandas



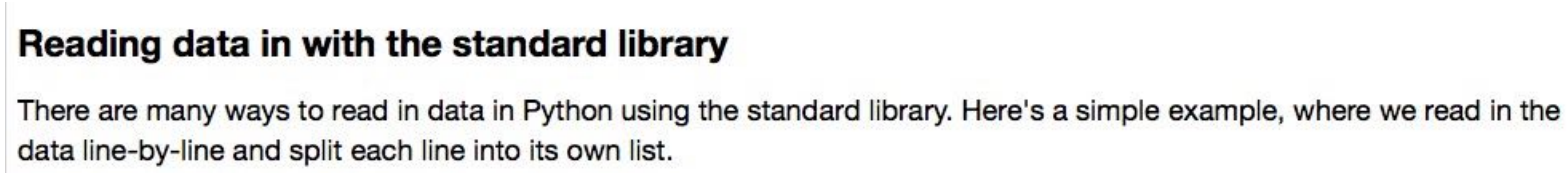
Importing data

- Before we do anything else, we need to get our data into a usable form
- Most commonly, data will come from a flat file
- But sometimes we need to retrieve data from other sources
- We'll do both

UC San Diego

Source: Tal Yarkoni

Not Pandas

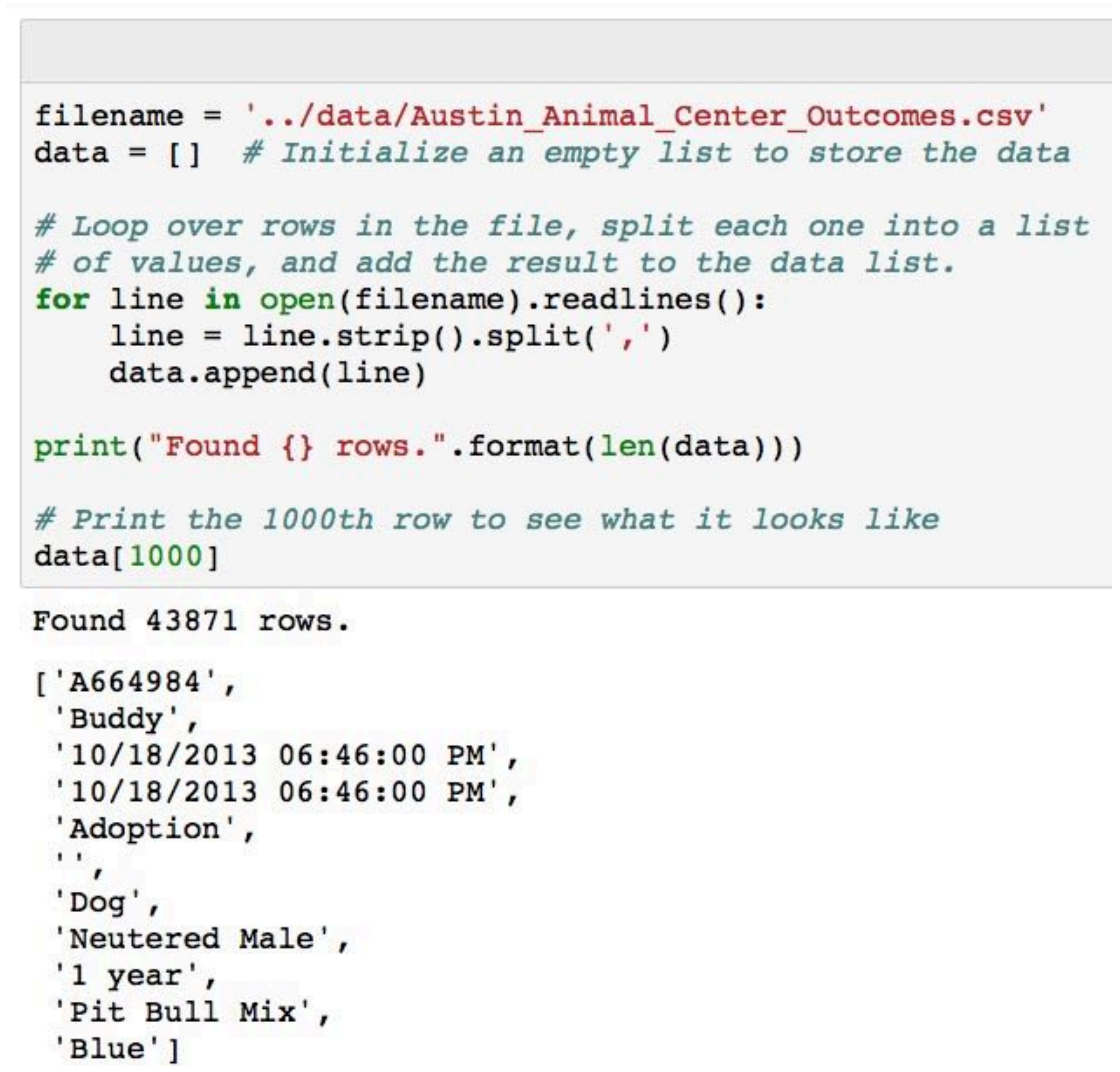


Reading data in with the standard library

There are many ways to read in data in Python using the standard library. Here's a simple example, where we read in the data line-by-line and split each line into its own list.

Source: Tal Yarkoni

Not Pandas

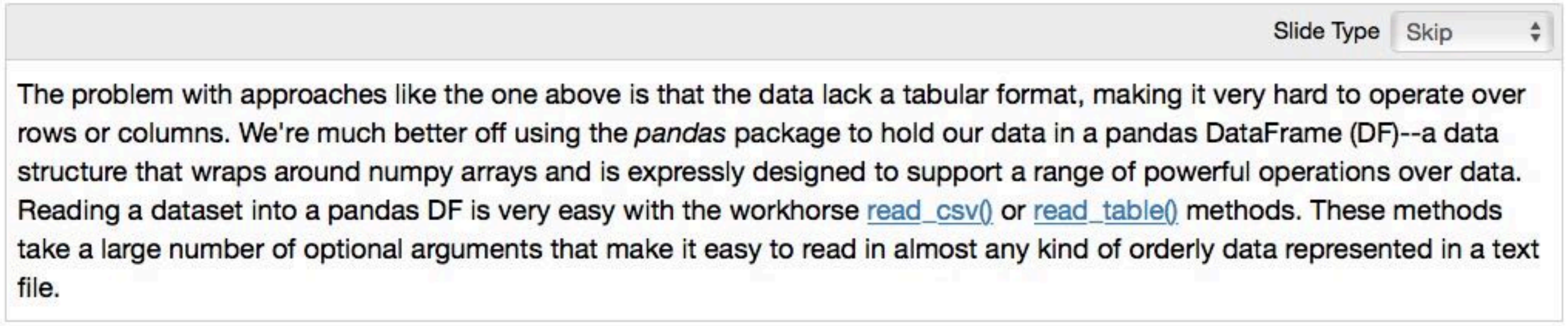


```
filename = '../data/Austin_Animal_Center_Outcomes.csv'
data = []  # Initialize an empty list to store the data

# Loop over rows in the file, split each one into a list
# of values, and add the result to the data list.
for line in open(filename).readlines():
    line = line.strip().split(',')
    data.append(line)

print("Found {} rows.".format(len(data)))

# Print the 1000th row to see what it looks like
data[1000]
```

```
Found 43871 rows.

['A664984',
 'Buddy',
 '10/18/2013 06:46:00 PM',
 '10/18/2013 06:46:00 PM',
 'Adoption',
 '',
 'Dog',
 'Neutered Male',
 '1 year',
 'Pit Bull Mix',
 'Blue']
```

Source: Tal Yarkoni

Pandas



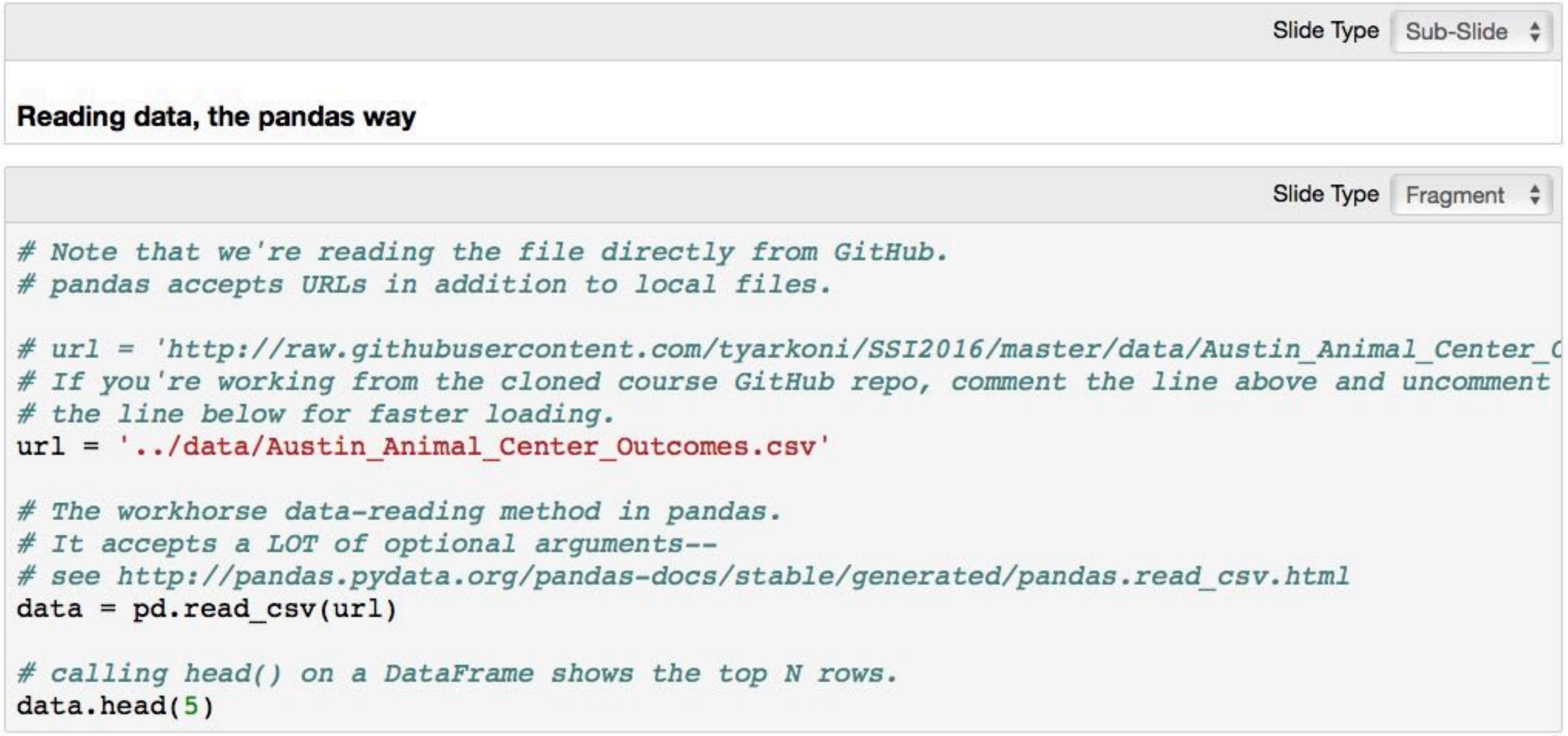
Slide Type Skip

The problem with approaches like the one above is that the data lack a tabular format, making it very hard to operate over rows or columns. We're much better off using the *pandas* package to hold our data in a pandas DataFrame (DF)--a data structure that wraps around numpy arrays and is expressly designed to support a range of powerful operations over data. Reading a dataset into a pandas DF is very easy with the workhorse read_csv() or read_table() methods. These methods take a large number of optional arguments that make it easy to read in almost any kind of orderly data represented in a text file.

Source: Tal Yarkoni

UC San Diego

Pandas



Slide Type Sub-Slide

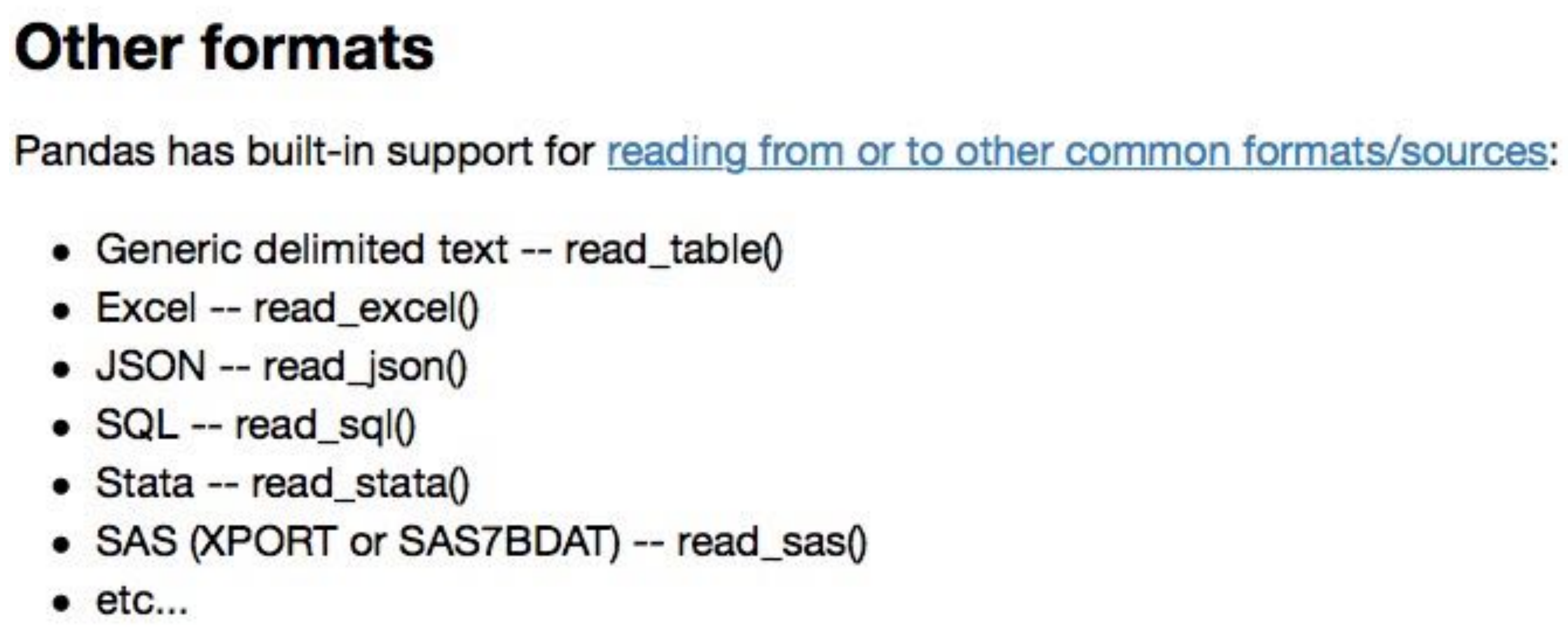
Reading data, the pandas way

Slide Type Fragment

```
# Note that we're reading the file directly from GitHub.
# pandas accepts URLs in addition to local files.

# url = 'http://raw.githubusercontent.com/tyarkoni/SSI2016/master/data/Austin_Animal_Center_
# If you're working from the cloned course GitHub repo, comment the line above and uncomment
# the line below for faster loading.
url = '../data/Austin_Animal_Center_Outcomes.csv'

# The workhorse data-reading method in pandas.
# It accepts a LOT of optional arguments--
# see http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html
data = pd.read_csv(url)

# calling head() on a DataFrame shows the top N rows.
data.head(5)
```

Source: Tal Yarkoni

Pandas



Other formats

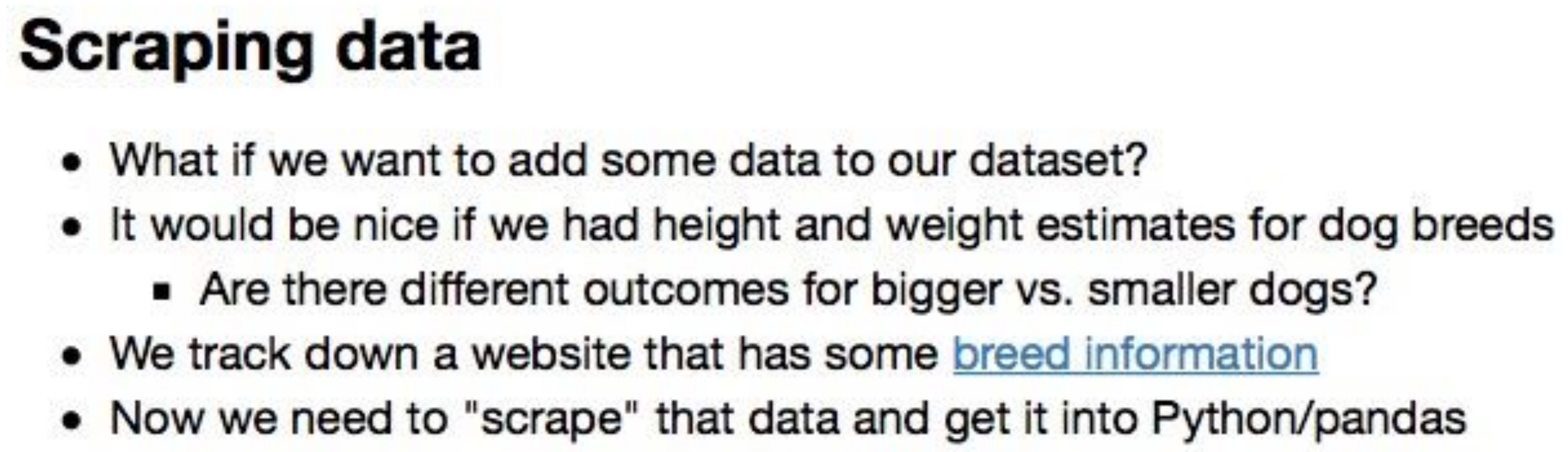
Pandas has built-in support for reading from or to other common formats/sources:

- Generic delimited text -- read_table()
- Excel -- read_excel()
- JSON -- read_json()
- SQL -- read_sql()
- Stata -- read_stata()
- SAS (XPORT or SAS7BDAT) -- read_sas()
- etc...

UC San Diego

Source: Tal Yarkoni

Pandas



Scraping data

- What if we want to add some data to our dataset?
- It would be nice if we had height and weight estimates for dog breeds
  - Are there different outcomes for bigger vs. smaller dogs?
- We track down a website that has some breed information
- Now we need to "scrape" that data and get it into Python/pandas

Source: Tal Yarkoni

COGS 108
Data Science in Practice

Data cleaning

The Joy of (not) Cooking (your data)

- Reality is just someone's else's unorganized datasets.

The Joy of (not) Cooking (your data)

- Reality is just someone's else's unorganized datasets.
- Your job is to *be as faithful to this reality as possible*…

The Joy of (not) Cooking (your data)

- Reality is just someone's else's unorganized datasets.
- Your job is to *be as faithful to this reality as possible*…
- …which means you need to identify the right data for the question at hand.

The Joy of (not) Cooking (your data)

- Reality is just someone's else's unorganized datasets.
- Your job is to *be as faithful to this reality as possible*…
- …which means you need to identify the right data for the question at hand.
- You also need to remove obvious *bad* data.

UC San Diego

The Joy of (not) Cooking (your data)

- Reality is just someone's else's unorganized datasets.
- Your job is to *be as faithful to this reality as possible…*
- …which means you need to identify the right data for the question at hand.
- You also need to remove obvious *bad* data.
- The trick is learning how to tell bad data from good.

UC San Diego

The Map is not the Territory

“Identifying the minimum set of features needed to account for a particular phenomenon and describing these accurately enough to do the job is a key component of model building.”

The Map is not the Territory

"Identifying the minimum set of features needed to account for a particular phenomenon and describing these accurately enough to do the job is a key component of model building.

"Anything more than this minimum set makes the model harder to understand and more difficult to evaluate."

Source: Abbot, "Theoretical Neuroscience Rising" *Neuron* 2008

The Map is not the Territory

“Identifying the minimum set of features needed to account for a particular phenomenon and describing these accurately enough to do the job is a key component of model building.

“Anything more than this minimum set makes the model harder to understand and more difficult to evaluate.

“The term “realistic” model is a sociological rather than a scientific term.”

The Map is not the Territory

“The truly realistic model is as impossible and useless a concept as Borges’ “map of the empire that was of the same scale as the empire and that coincided with it point for point” (Borges, 1975).”

The Map is not the Territory

“The truly realistic model is as impossible and useless a concept as Borges’ “map of the empire that was of the same scale as the empire and that coincided with it point for point” (Borges, 1975).

“In any model, a phenomenon must be accounted for by an approximate description of a subset of the features that exist in the natural system.”

Source: Abbot, “Theoretical Neuroscience Rising” *Neuron* 2008

The Map is not the Territory

“The truly realistic model is as impossible and useless a concept as Borges’ “map of the empire that was of the same scale as the empire and that coincided with it point for point” (Borges, 1975).

“In any model, a phenomenon must be accounted for by an approximate description of a subset of the features that exist in the natural system.

“The art of modeling lies in deciding what this subset should be and how it should be described.”

Source: Abbot, “Theoretical Neuroscience Rising” *Neuron* 2008

Sooo…???

That all sounds frickin' awesome.

So how do we *do* that?

Sooo…???

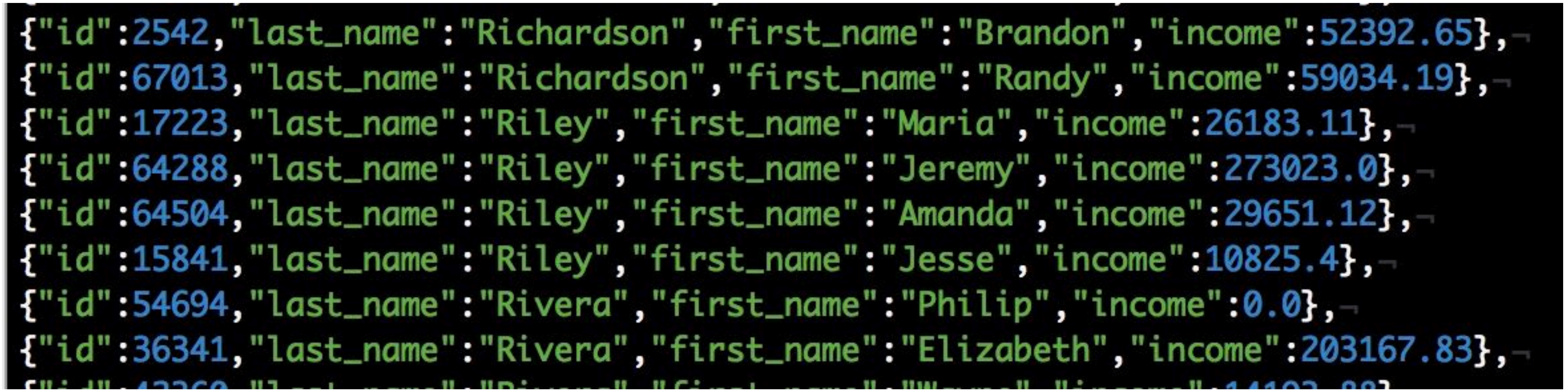
That all sounds frickin' awesome.

So how do we *do* that?

Answer: *It's boring.*

JSON data

- JSON: [id, last_name, first_name, income]

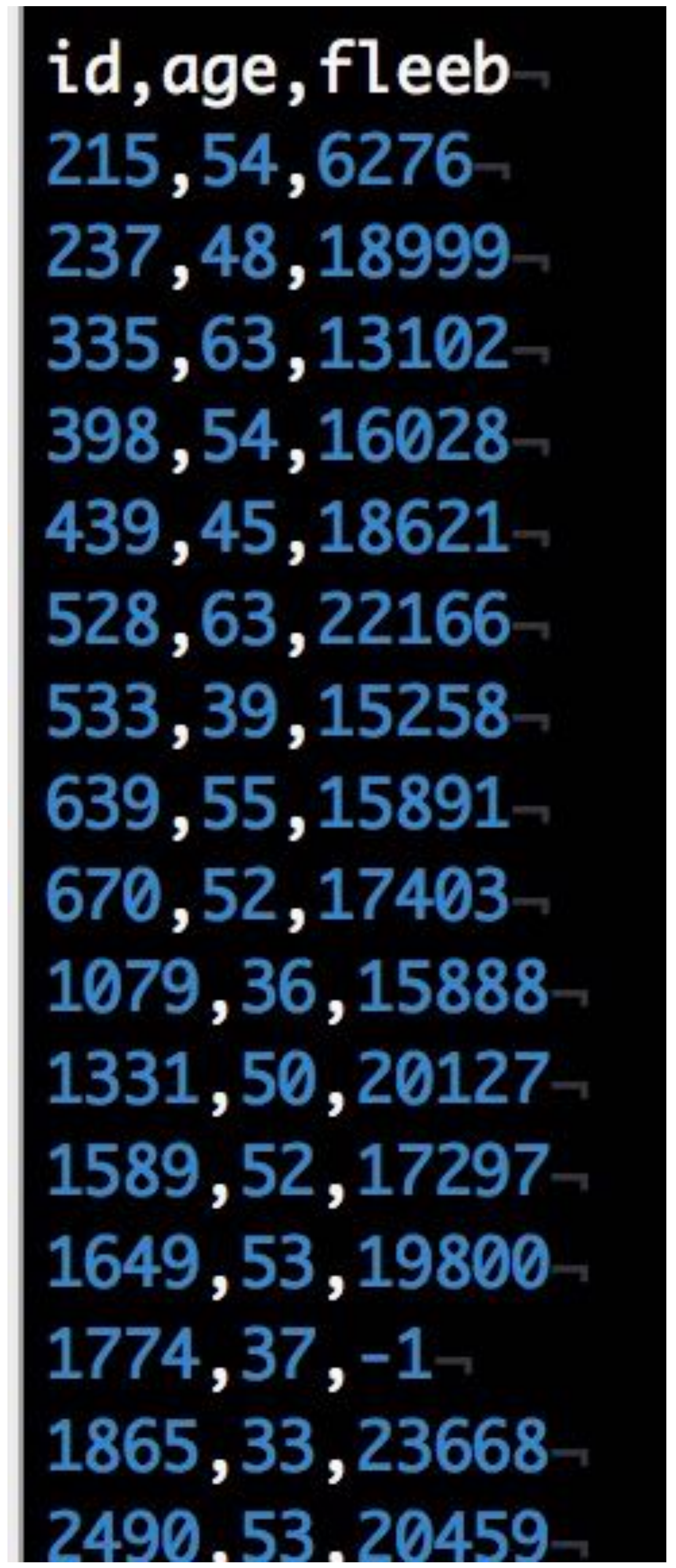


```
{"id":2542,"last_name":"Richardson","first_name":"Brandon","income":52392.65},
{"id":67013,"last_name":"Richardson","first_name":"Randy","income":59034.19},
{"id":17223,"last_name":"Riley","first_name":"Maria","income":26183.11},
{"id":64288,"last_name":"Riley","first_name":"Jeremy","income":273023.0},
{"id":64504,"last_name":"Riley","first_name":"Amanda","income":29651.12},
{"id":15841,"last_name":"Riley","first_name":"Jesse","income":10825.4},
{"id":54694,"last_name":"Rivera","first_name":"Philip","income":0.0},
{"id":36341,"last_name":"Rivera","first_name":"Elizabeth","income":203167.83},
```

UC San Diego

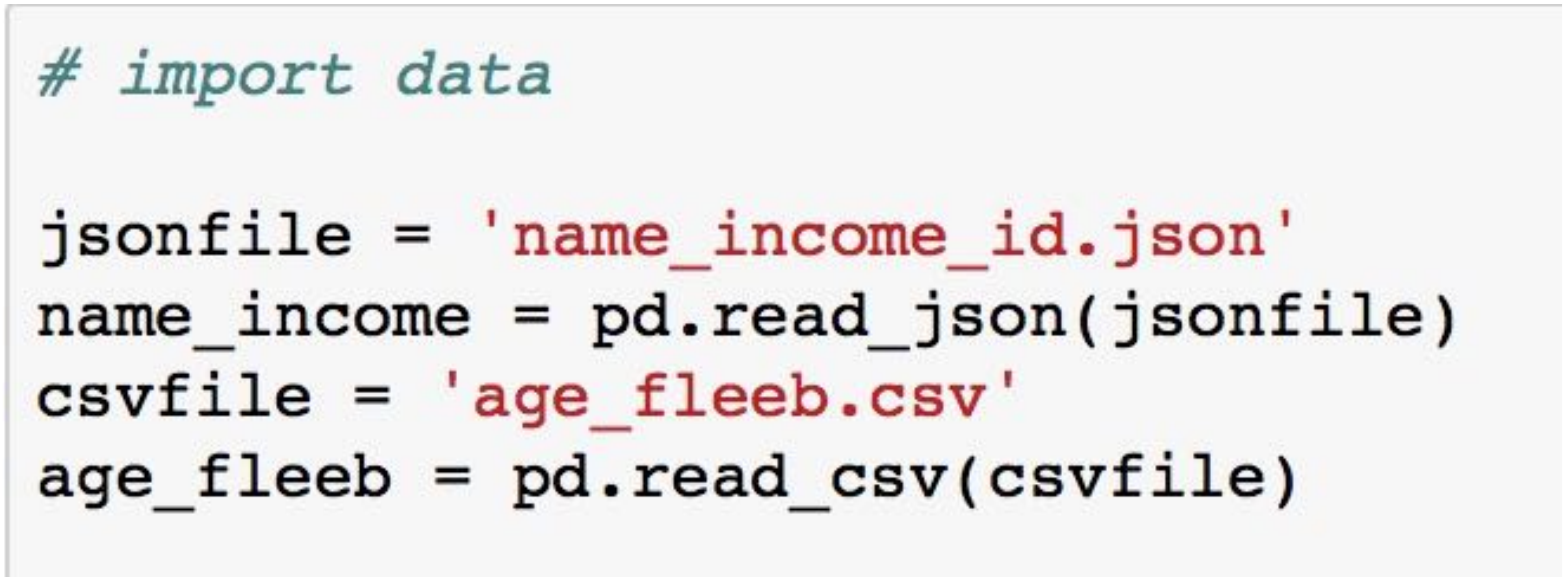
CSV data

- CSV: [id, age, fleeb]



```
id,age,fleeb
215,54,6276
237,48,18999
335,63,13102
398,54,16028
439,45,18621
528,63,22166
533,39,15258
639,55,15891
670,52,17403
1079,36,15888
1331,50,20127
1589,52,17297
1649,53,19800
1774,37,-1
1865,33,23668
2490,53,20459
```

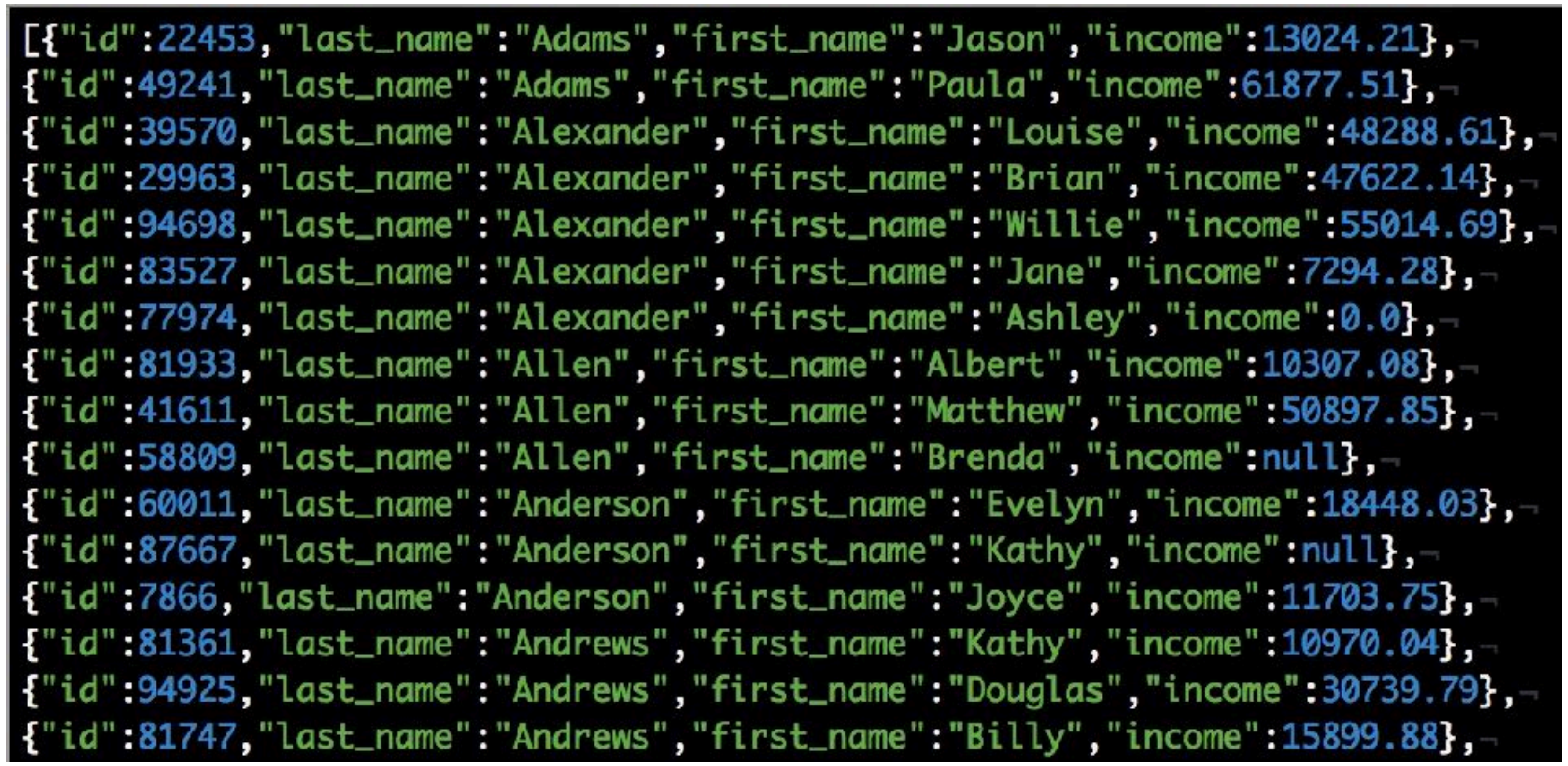
Import JSON and CSV



```
# import data

jsonfile = 'name_income_id.json'
name_income = pd.read_json(jsonfile)
csvfile = 'age_fleeb.csv'
age_fleeb = pd.read_csv(csvfile)
```

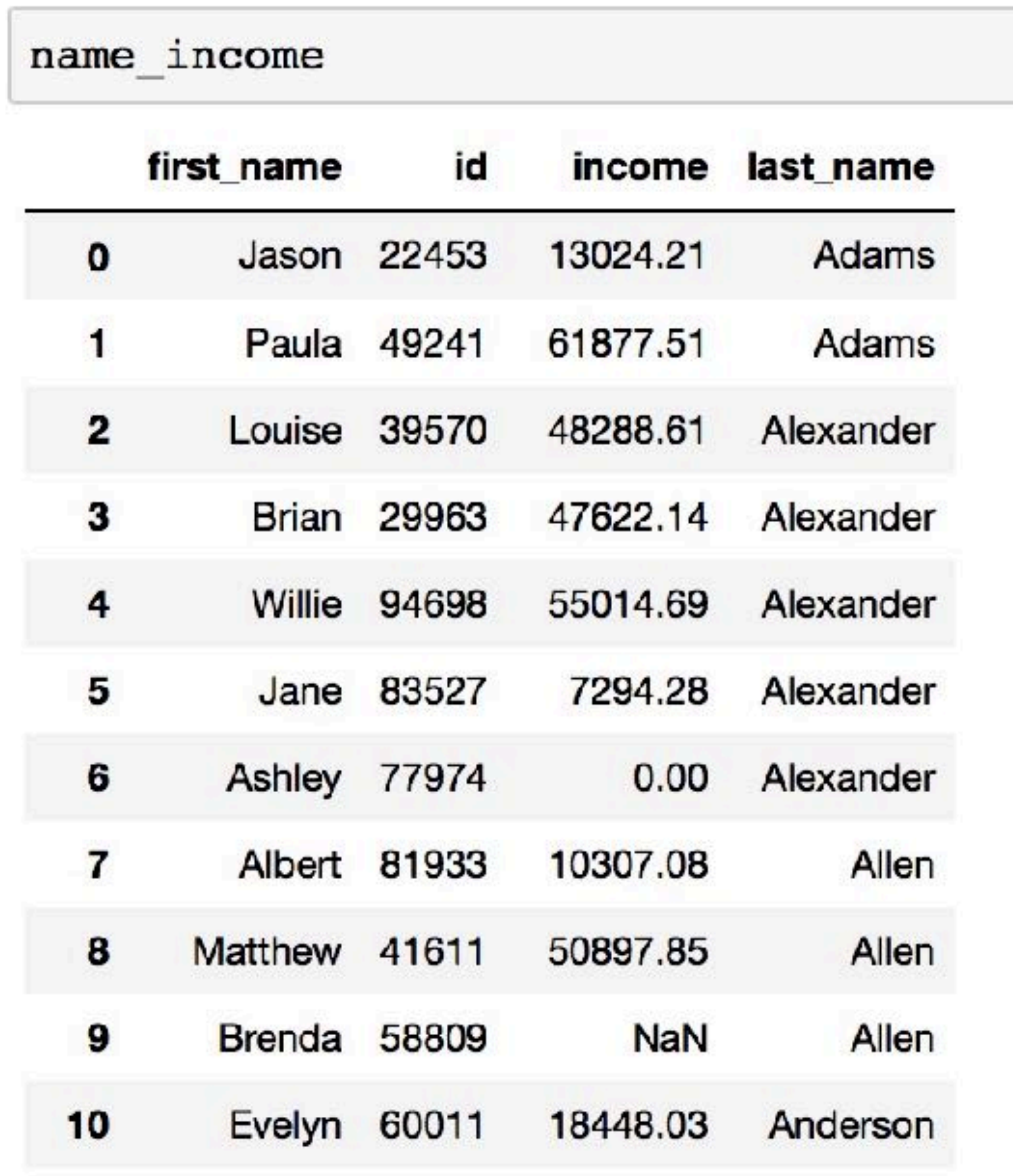
Raw JSON and in Pandas

raw JSON



```
[{"id":22453,"last_name":"Adams","first_name":"Jason","income":13024.21},
{"id":49241,"last_name":"Adams","first_name":"Paula","income":61877.51},
{"id":39570,"last_name":"Alexander","first_name":"Louise","income":48288.61},
{"id":29963,"last_name":"Alexander","first_name":"Brian","income":47622.14},
{"id":94698,"last_name":"Alexander","first_name":"Willie","income":55014.69},
{"id":83527,"last_name":"Alexander","first_name":"Jane","income":7294.28},
{"id":77974,"last_name":"Alexander","first_name":"Ashley","income":0.0},
{"id":81933,"last_name":"Allen","first_name":"Albert","income":10307.08},
{"id":41611,"last_name":"Allen","first_name":"Matthew","income":50897.85},
{"id":58809,"last_name":"Allen","first_name":"Brenda","income":null},
{"id":60011,"last_name":"Anderson","first_name":"Evelyn","income":18448.03},
{"id":87667,"last_name":"Anderson","first_name":"Kathy","income":null},
{"id":7866,"last_name":"Anderson","first_name":"Joyce","income":11703.75},
{"id":81361,"last_name":"Andrews","first_name":"Kathy","income":10970.04},
{"id":94925,"last_name":"Andrews","first_name":"Douglas","income":30739.79},
{"id":81747,"last_name":"Andrews","first_name":"Billy","income":15899.88},
```

dataframe

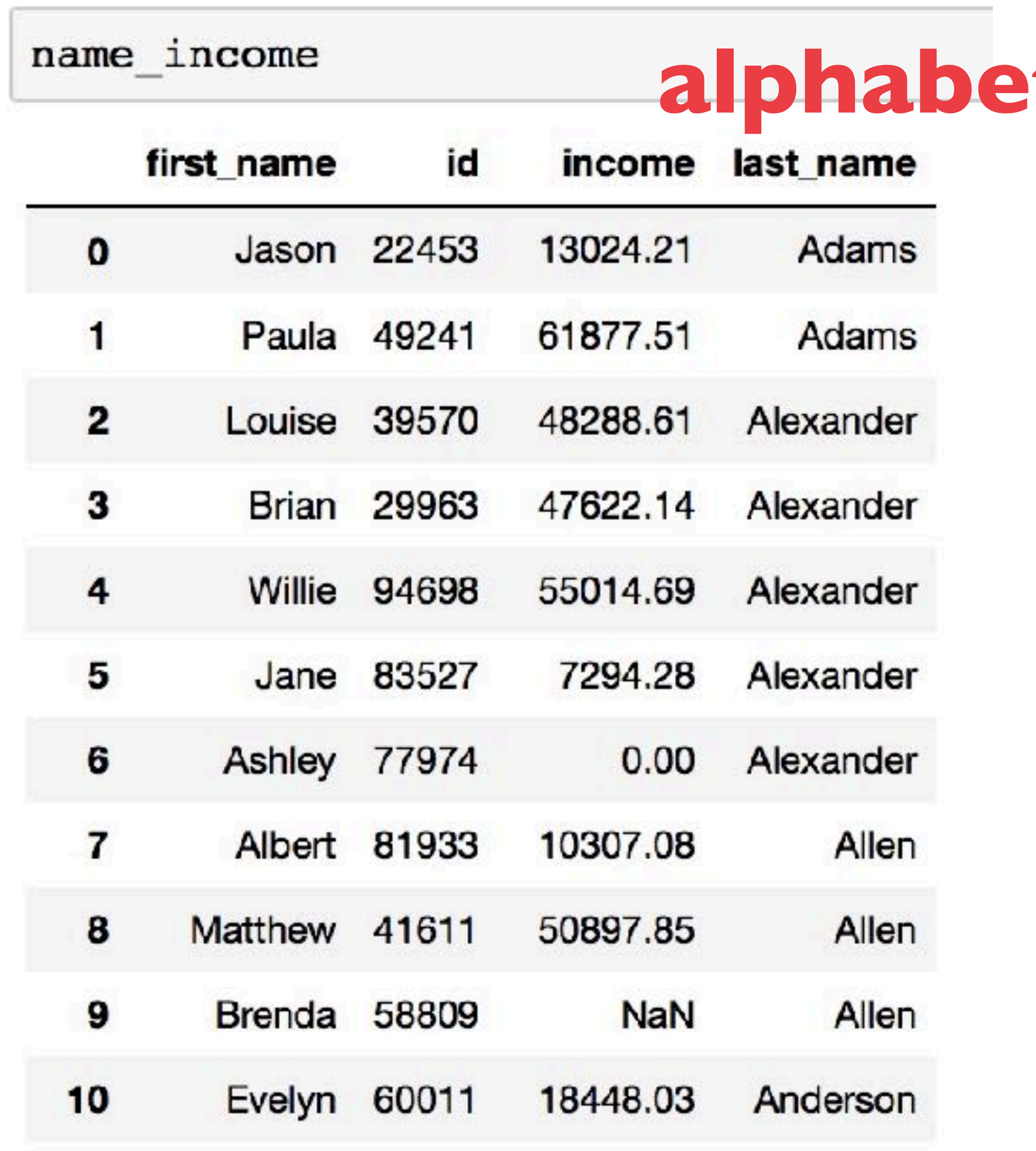


```
name_income
```

	first_name	id	income	last_name
0	Jason	22453	13024.21	Adams
1	Paula	49241	61877.51	Adams
2	Louise	39570	48288.61	Alexander
3	Brian	29963	47622.14	Alexander
4	Willie	94698	55014.69	Alexander
5	Jane	83527	7294.28	Alexander
6	Ashley	77974	0.00	Alexander
7	Albert	81933	10307.08	Allen
8	Matthew	41611	50897.85	Allen
9	Brenda	58809	NaN	Allen
10	Evelyn	60011	18448.03	Anderson

Reorder dataframe columns

dataframe

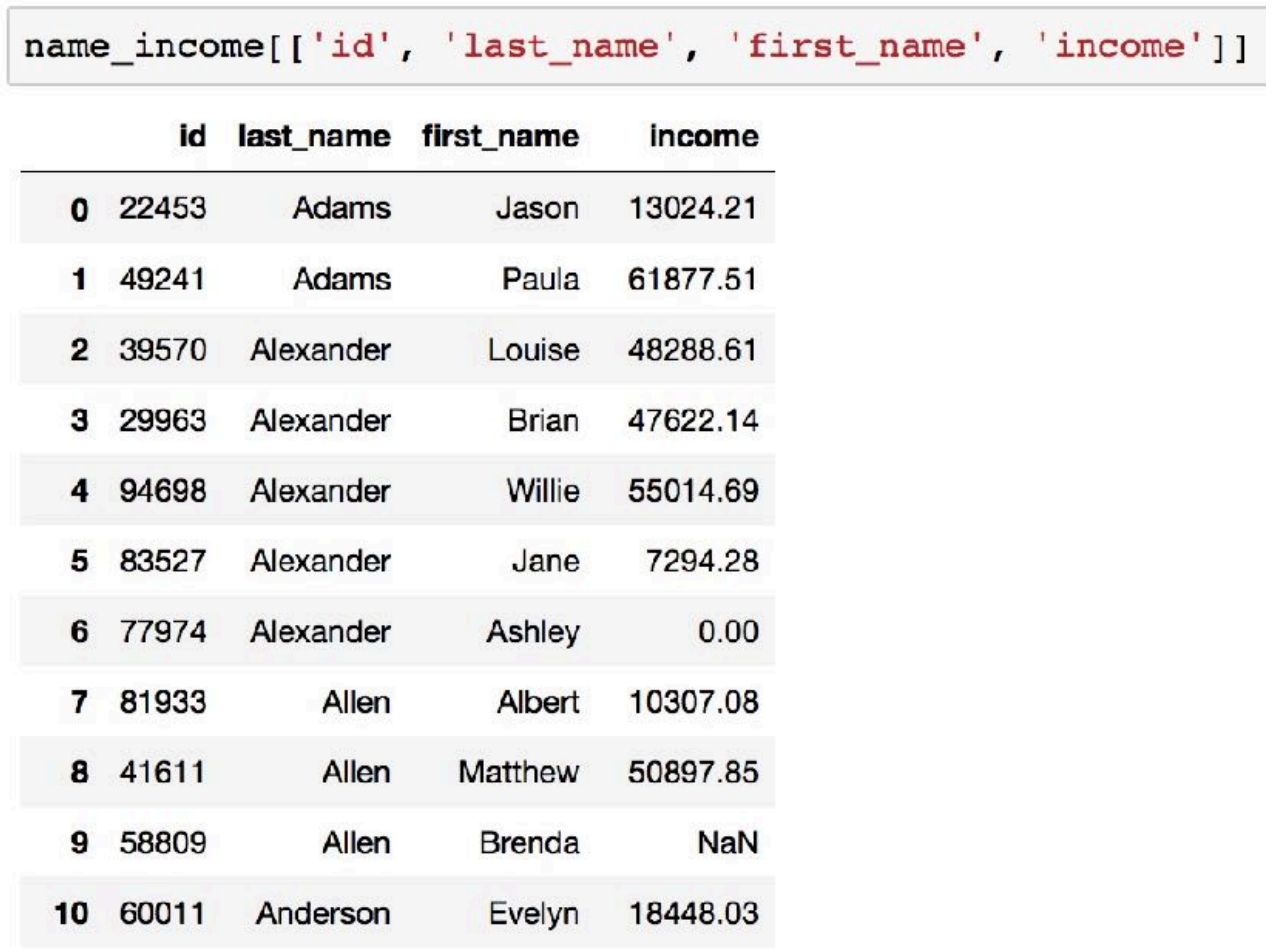


```
name_income
```

alphabetical!

	first_name	id	income	last_name
0	Jason	22453	13024.21	Adams
1	Paula	49241	61877.51	Adams
2	Louise	39570	48288.61	Alexander
3	Brian	29963	47622.14	Alexander
4	Willie	94698	55014.69	Alexander
5	Jane	83527	7294.28	Alexander
6	Ashley	77974	0.00	Alexander
7	Albert	81933	10307.08	Allen
8	Matthew	41611	50897.85	Allen
9	Brenda	58809	NaN	Allen
10	Evelyn	60011	18448.03	Anderson

dataframe

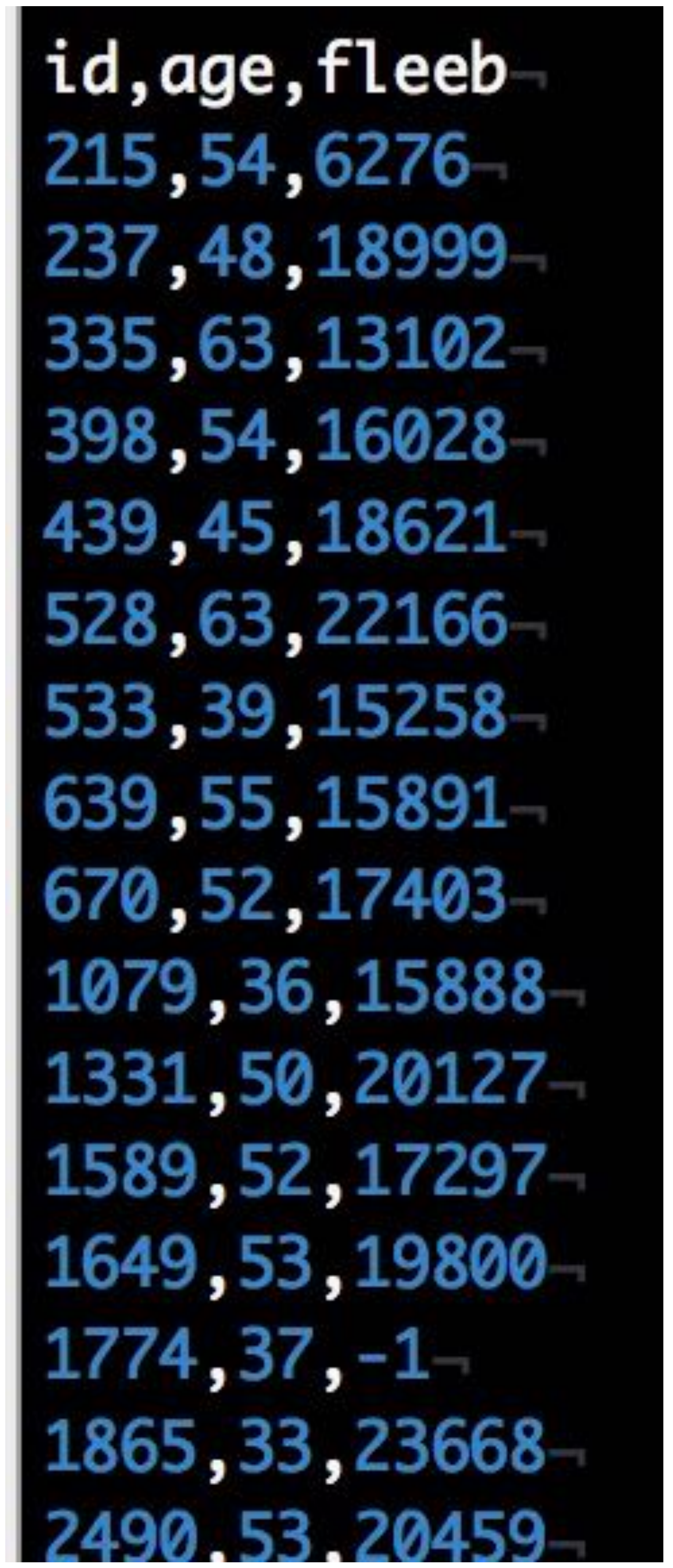


```
name_income[['id', 'last_name', 'first_name', 'income']]
```

	id	last_name	first_name	income
0	22453	Adams	Jason	13024.21
1	49241	Adams	Paula	61877.51
2	39570	Alexander	Louise	48288.61
3	29963	Alexander	Brian	47622.14
4	94698	Alexander	Willie	55014.69
5	83527	Alexander	Jane	7294.28
6	77974	Alexander	Ashley	0.00
7	81933	Allen	Albert	10307.08
8	41611	Allen	Matthew	50897.85
9	58809	Allen	Brenda	NaN
10	60011	Anderson	Evelyn	18448.03

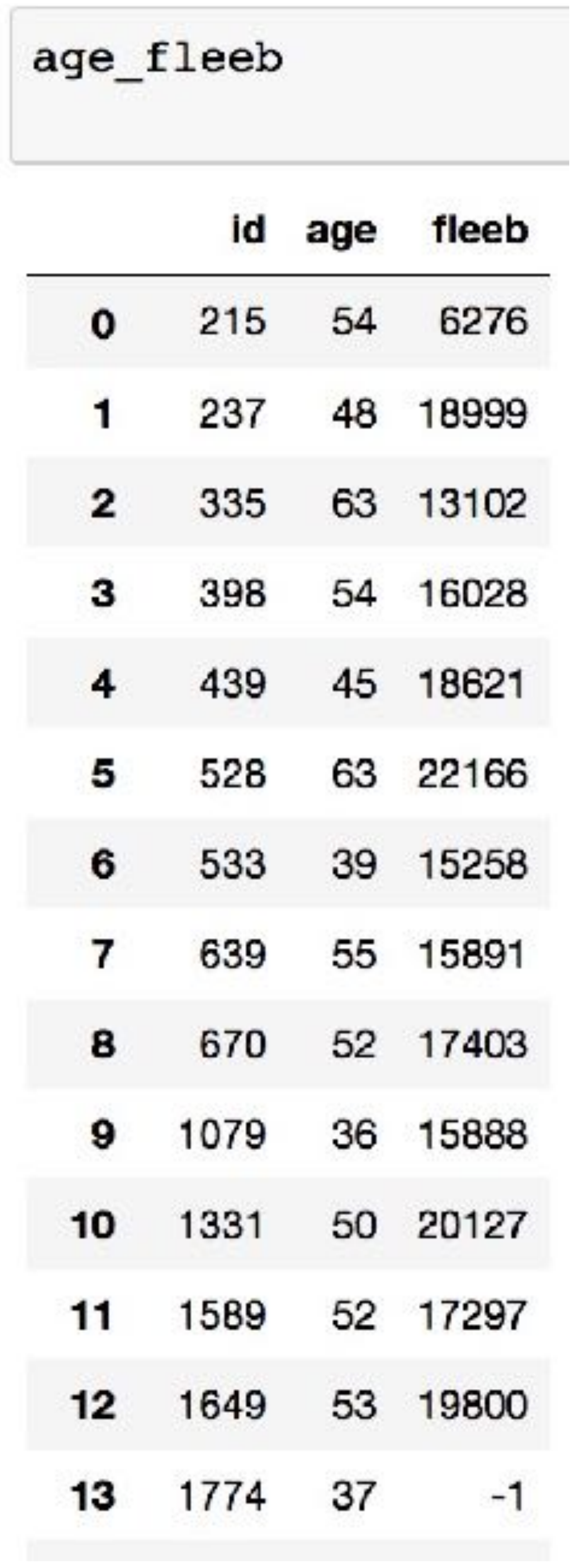
Raw CSV and in Pandas

raw CSV



```
id,age,fleeb
215,54,6276
237,48,18999
335,63,13102
398,54,16028
439,45,18621
528,63,22166
533,39,15258
639,55,15891
670,52,17403
1079,36,15888
1331,50,20127
1589,52,17297
1649,53,19800
1774,37,-1
1865,33,23668
2490,53,20459
```

dataframe

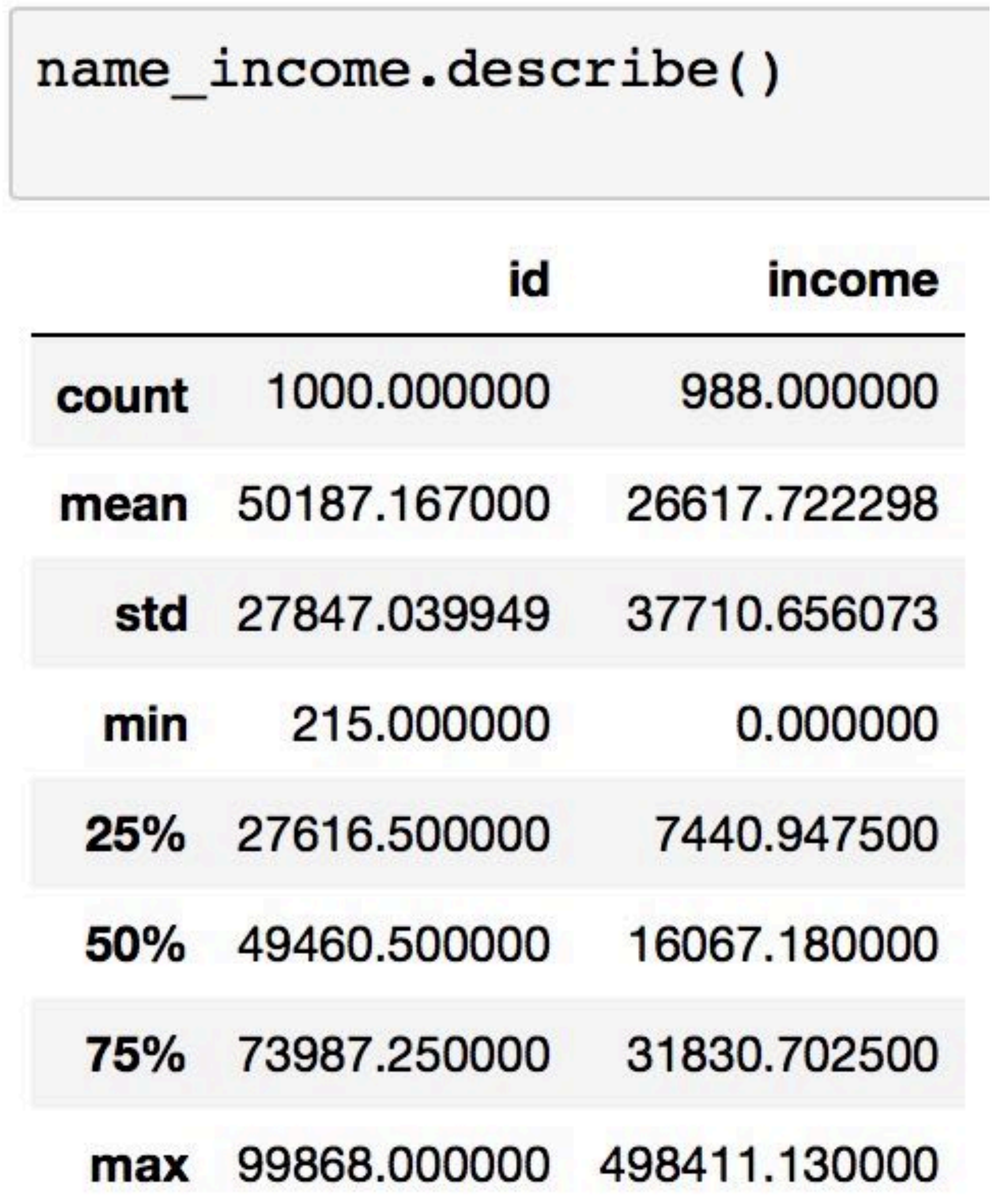


```
age_fleeb
```

	id	age	fleeb
0	215	54	6276
1	237	48	18999
2	335	63	13102
3	398	54	16028
4	439	45	18621
5	528	63	22166
6	533	39	15258
7	639	55	15891
8	670	52	17403
9	1079	36	15888
10	1331	50	20127
11	1589	52	17297
12	1649	53	19800
13	1774	37	-1

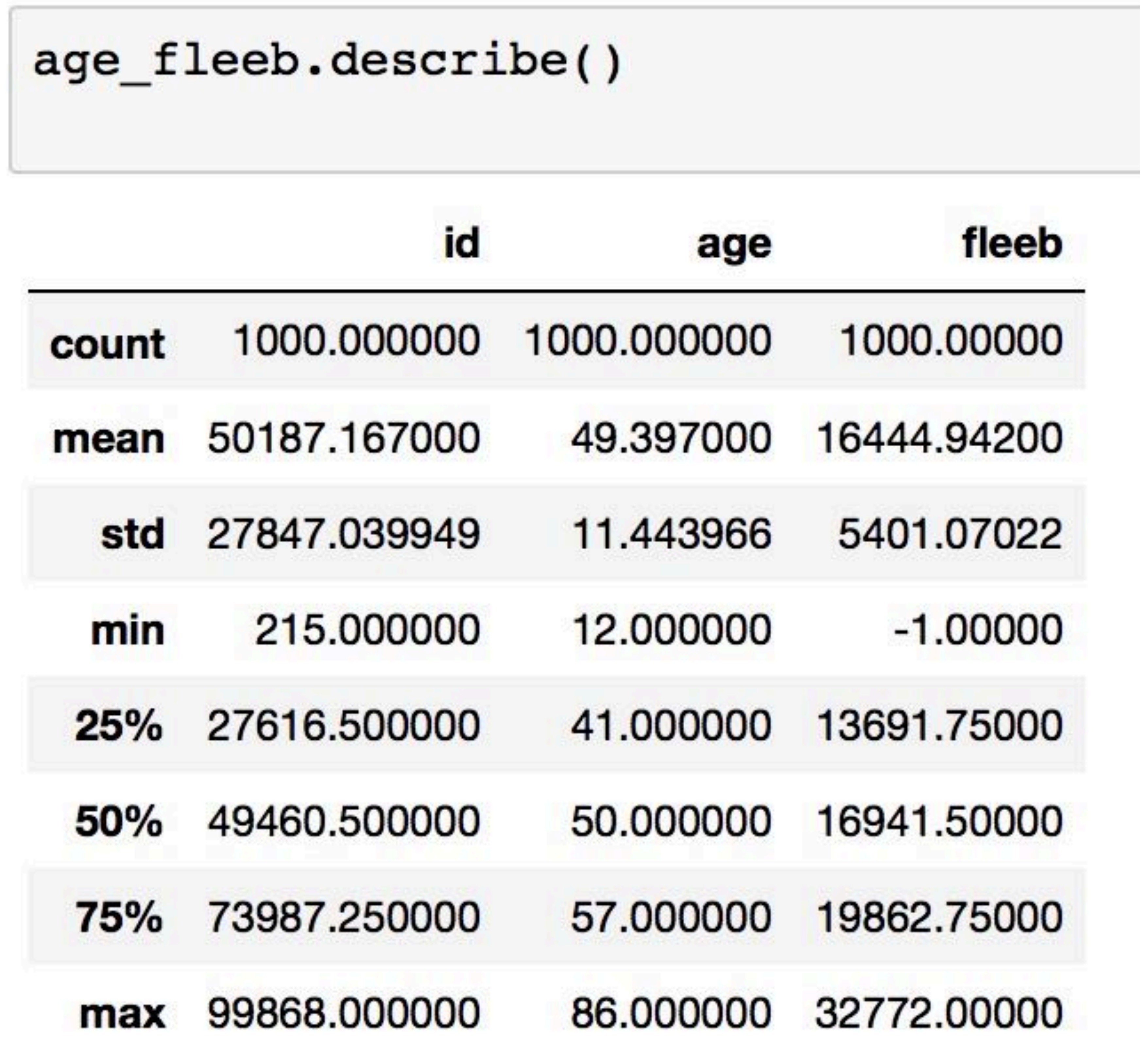
column order preserved

Describing data



```
name_income.describe()
```

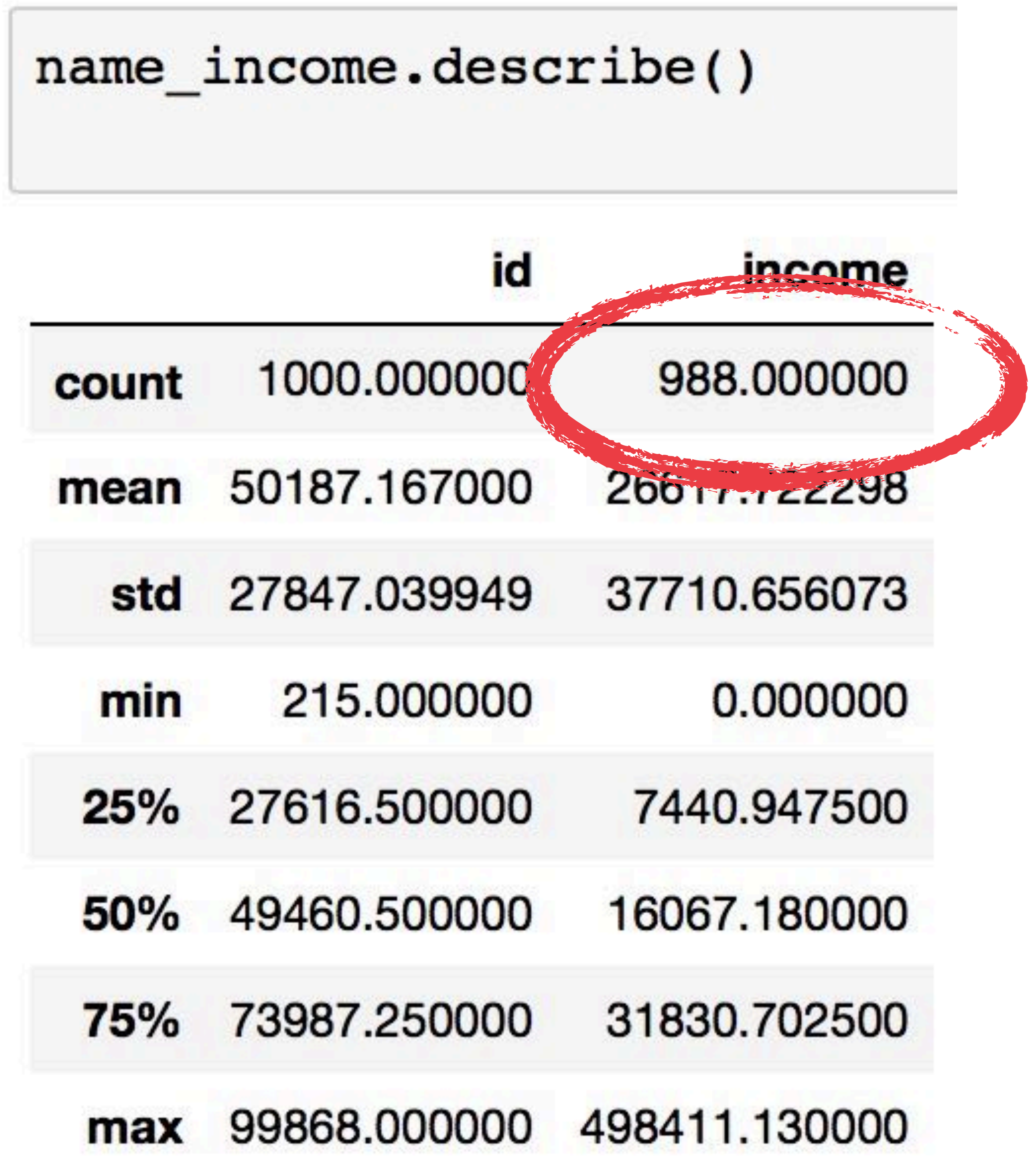
	id	income
count	1000.000000	988.000000
mean	50187.167000	26617.722298
std	27847.039949	37710.656073
min	215.000000	0.000000
25%	27616.500000	7440.947500
50%	49460.500000	16067.180000
75%	73987.250000	31830.702500
max	99868.000000	498411.130000



```
age_fleeb.describe()
```

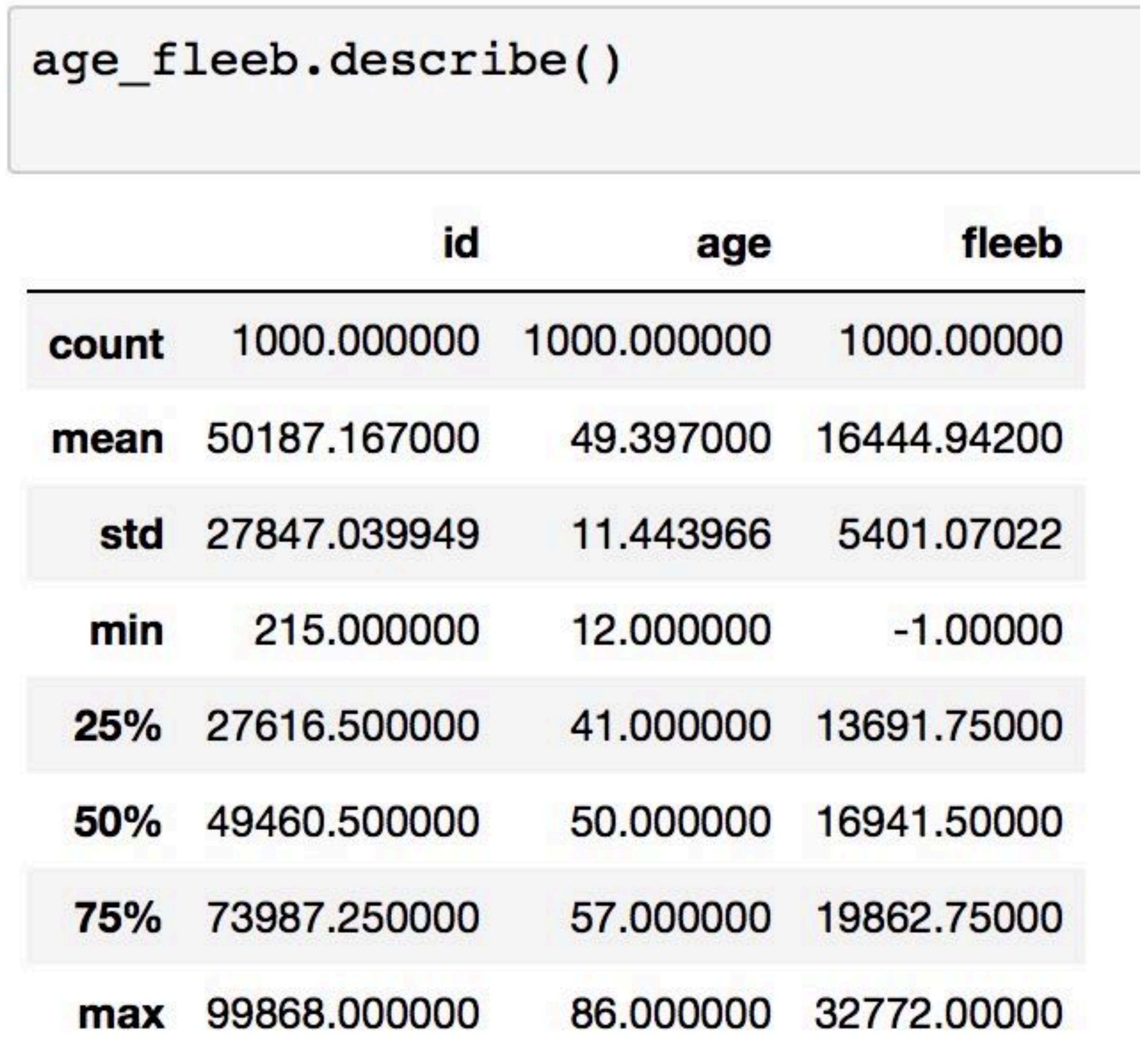
	id	age	fleeb
count	1000.000000	1000.000000	1000.00000
mean	50187.167000	49.397000	16444.94200
std	27847.039949	11.443966	5401.07022
min	215.000000	12.000000	-1.00000
25%	27616.500000	41.000000	13691.75000
50%	49460.500000	50.000000	16941.50000
75%	73987.250000	57.000000	19862.75000
max	99868.000000	86.000000	32772.00000

Describing data



```python
name_income.describe()
```

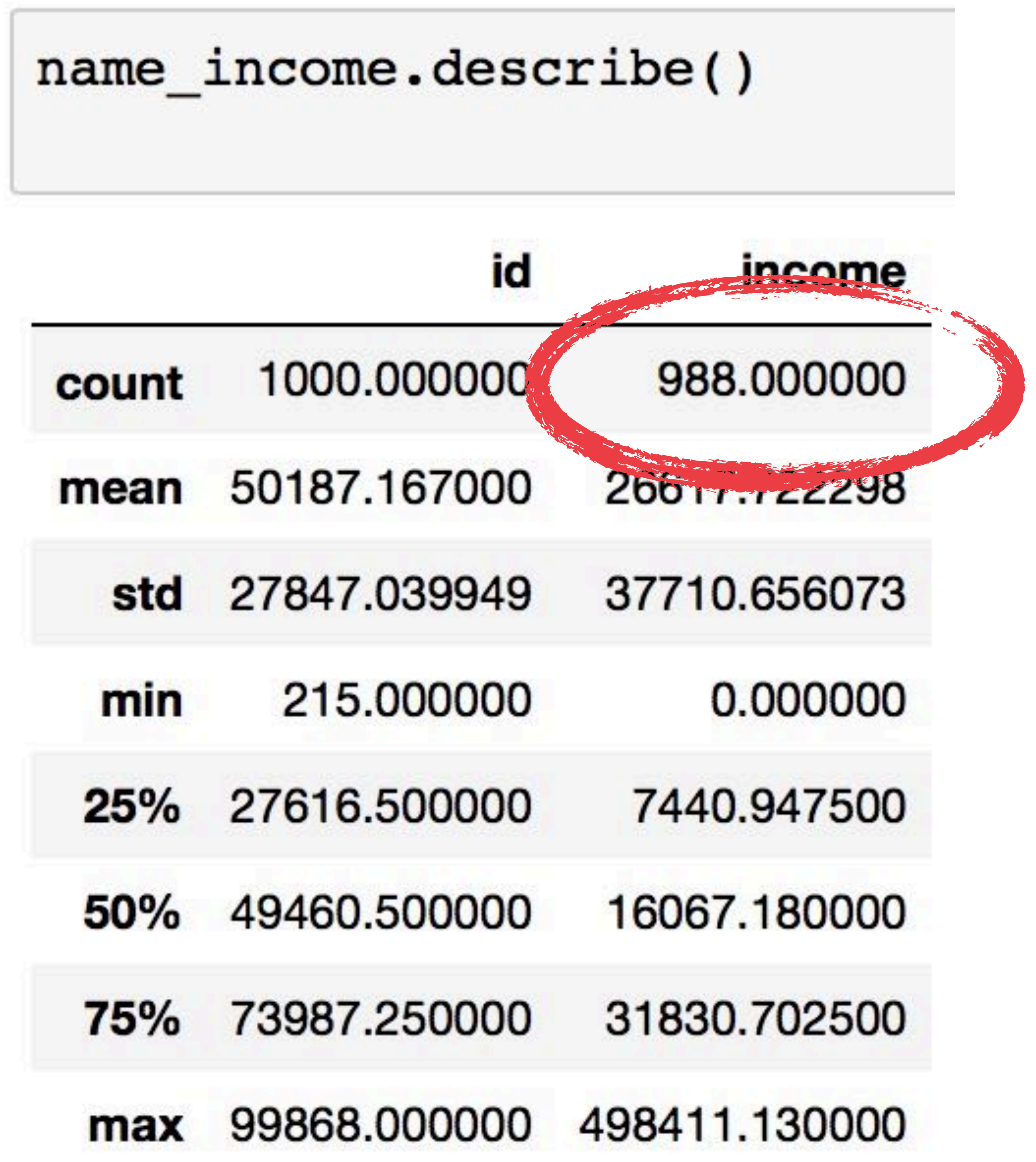
	id	income
count	1000.000000	988.000000
mean	50187.167000	26617.722298
std	27847.039949	37710.656073
min	215.000000	0.000000
25%	27616.500000	7440.947500
50%	49460.500000	16067.180000
75%	73987.250000	31830.702500
max	99868.000000	498411.130000



```python
age_fleeb.describe()
```

	id	age	fleeb
count	1000.000000	1000.000000	1000.00000
mean	50187.167000	49.397000	16444.94200
std	27847.039949	11.443966	5401.07022
min	215.000000	12.000000	-1.00000
25%	27616.500000	41.000000	13691.75000
50%	49460.500000	50.000000	16941.50000
75%	73987.250000	57.000000	19862.75000
max	99868.000000	86.000000	32772.00000

Describing data

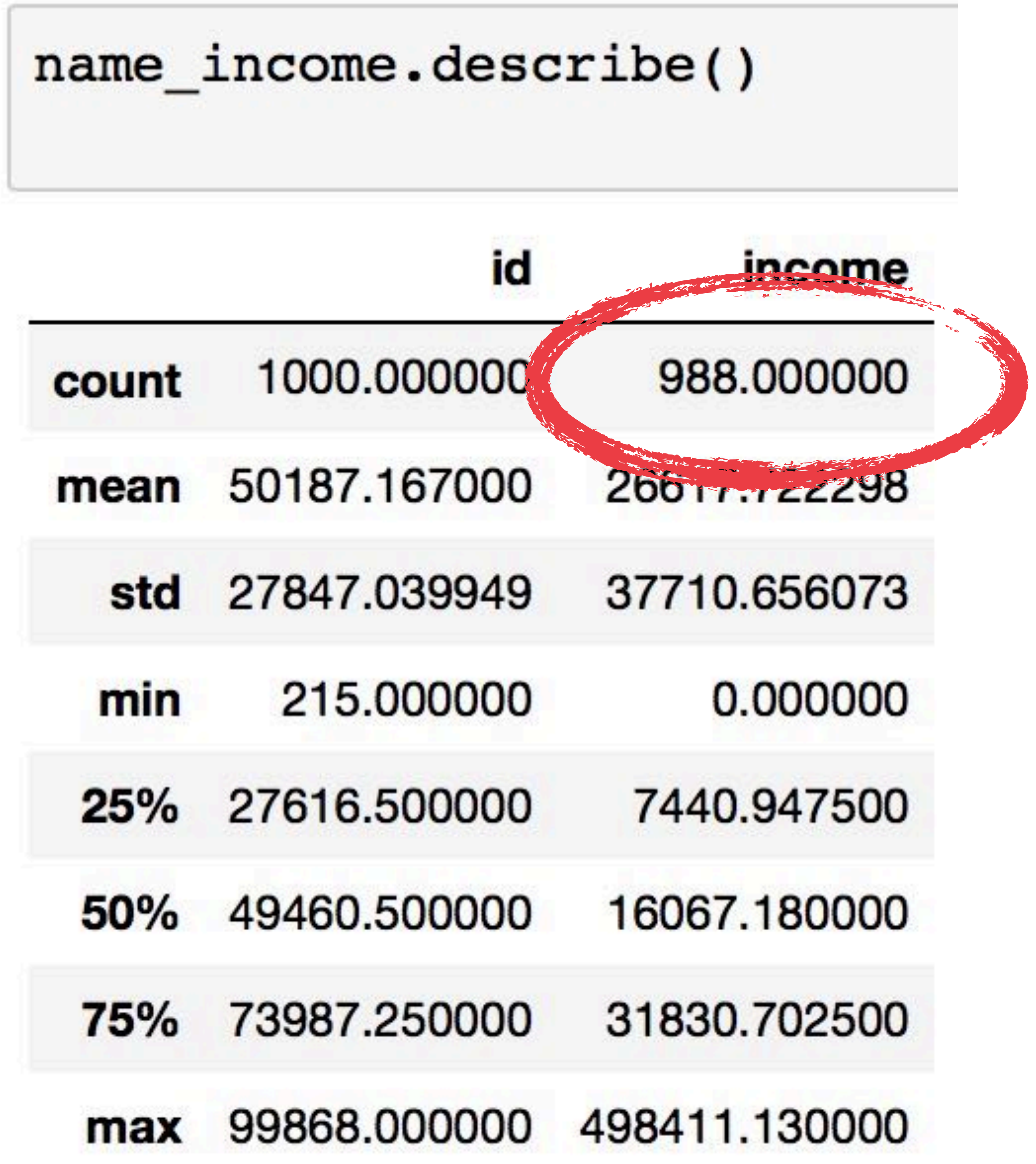


```
name_income.describe()
```

	id	income
count	1000.000000	988.000000
mean	50187.167000	26617.722298
std	27847.039949	37710.656073
min	215.000000	0.000000
25%	27616.500000	7440.947500
50%	49460.500000	16067.180000
75%	73987.250000	31830.702500
max	99868.000000	498411.130000

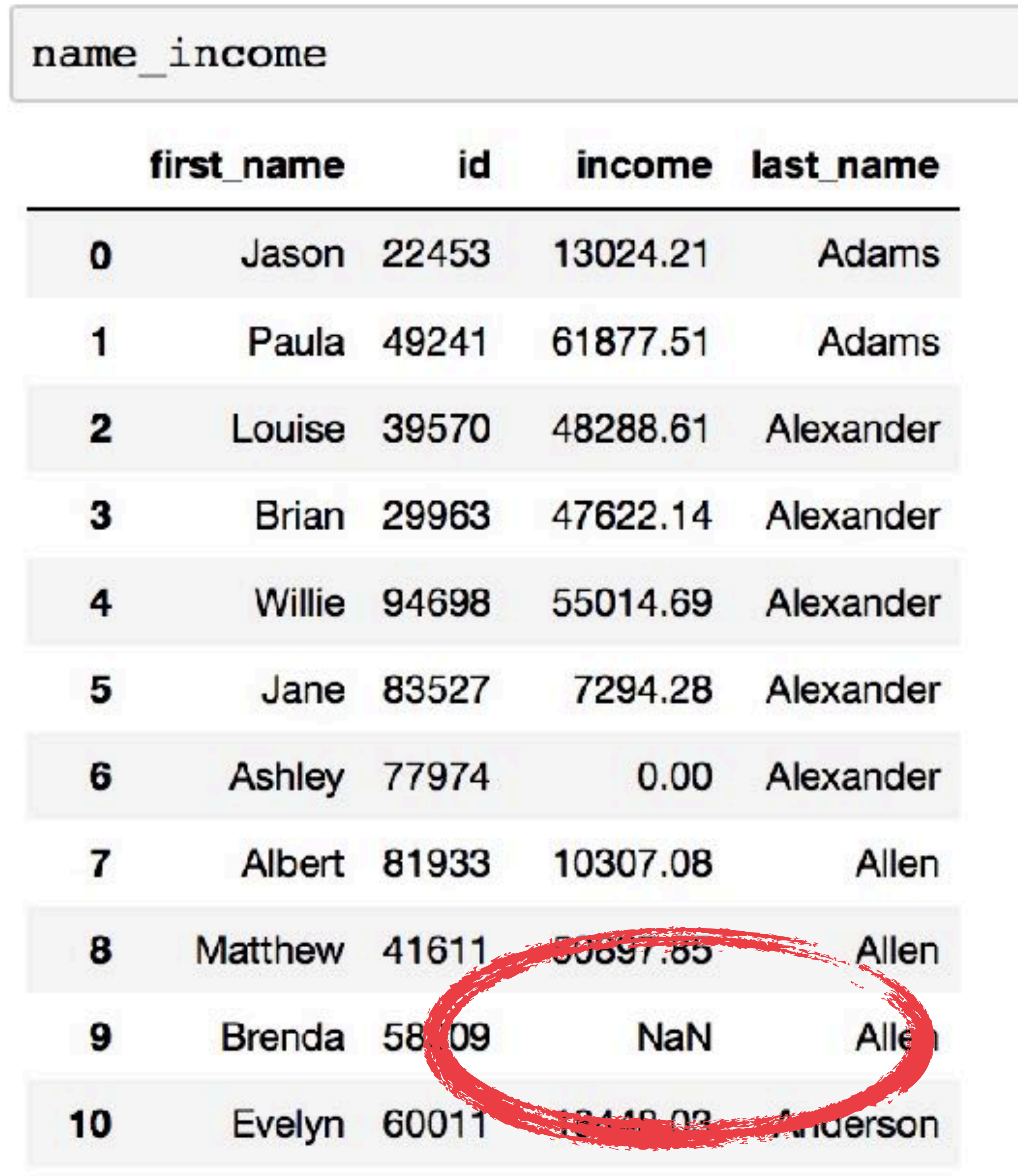
UC San Diego

Describing data



```
name_income.describe()
```

	id	income
count	1000.000000	988.000000
mean	50187.167000	26617.722298
std	27847.039949	37710.656073
min	215.000000	0.000000
25%	27616.500000	7440.947500
50%	49460.500000	16067.180000
75%	73987.250000	31830.702500
max	99868.000000	498411.130000

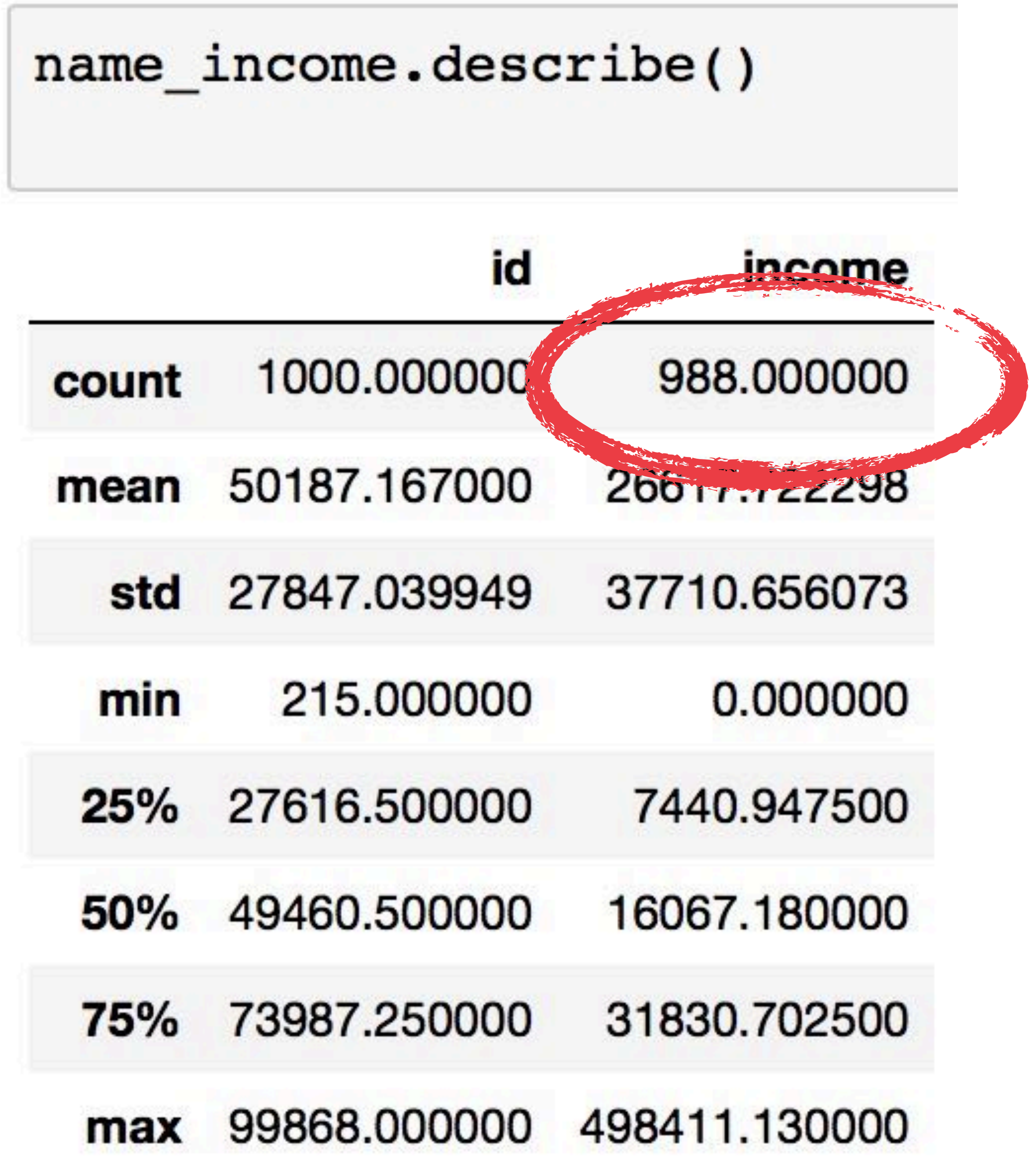


```
name_income
```

	first_name	id	income	last_name
0	Jason	22453	13024.21	Adams
1	Paula	49241	61877.51	Adams
2	Louise	39570	48288.61	Alexander
3	Brian	29963	47622.14	Alexander
4	Willie	94698	55014.69	Alexander
5	Jane	83527	7294.28	Alexander
6	Ashley	77974	0.00	Alexander
7	Albert	81933	10307.08	Allen
8	Matthew	41611	50697.85	Allen
9	Brenda	58[illegible]09	NaN	Allen
10	Evelyn	60011	13448.03	Anderson

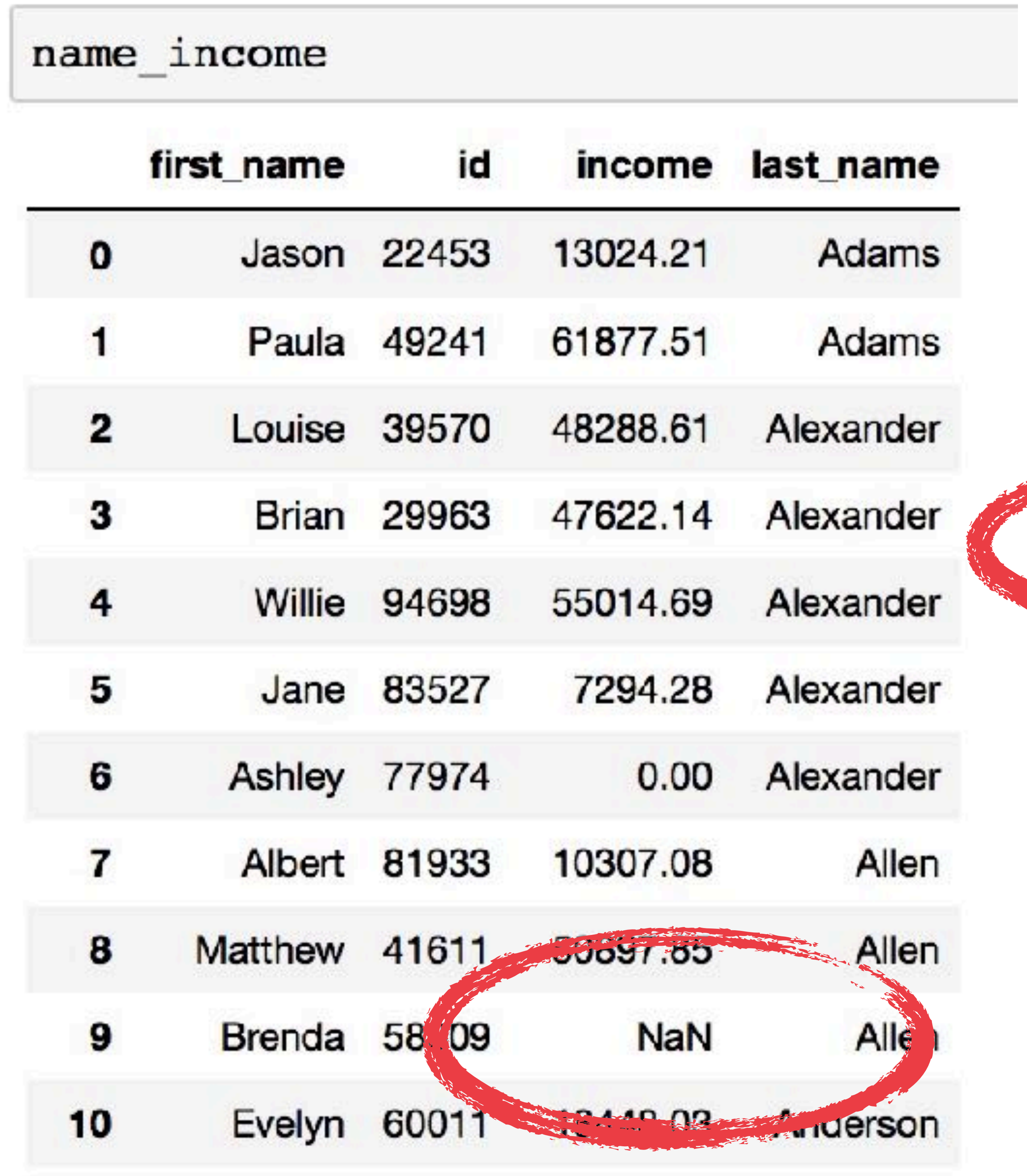
UC San Diego

Describing data



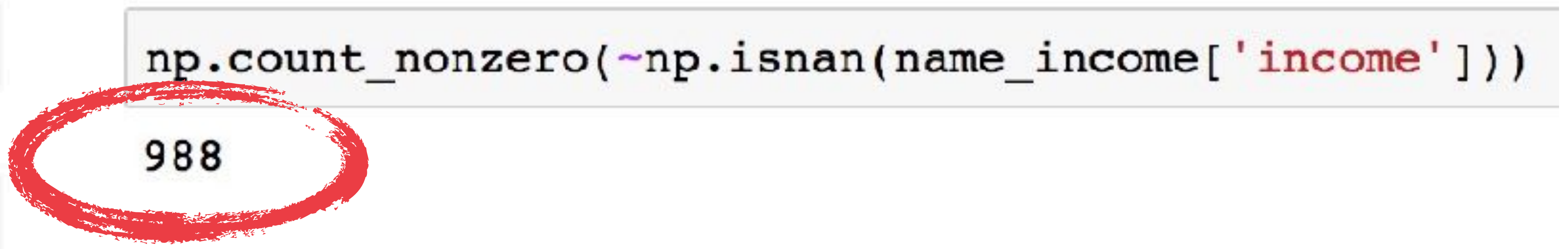
```
name_income.describe()
```

	id	income
count	1000.000000	988.000000
mean	50187.167000	26617.722298
std	27847.039949	37710.656073
min	215.000000	0.000000
25%	27616.500000	7440.947500
50%	49460.500000	16067.180000
75%	73987.250000	31830.702500
max	99868.000000	498411.130000



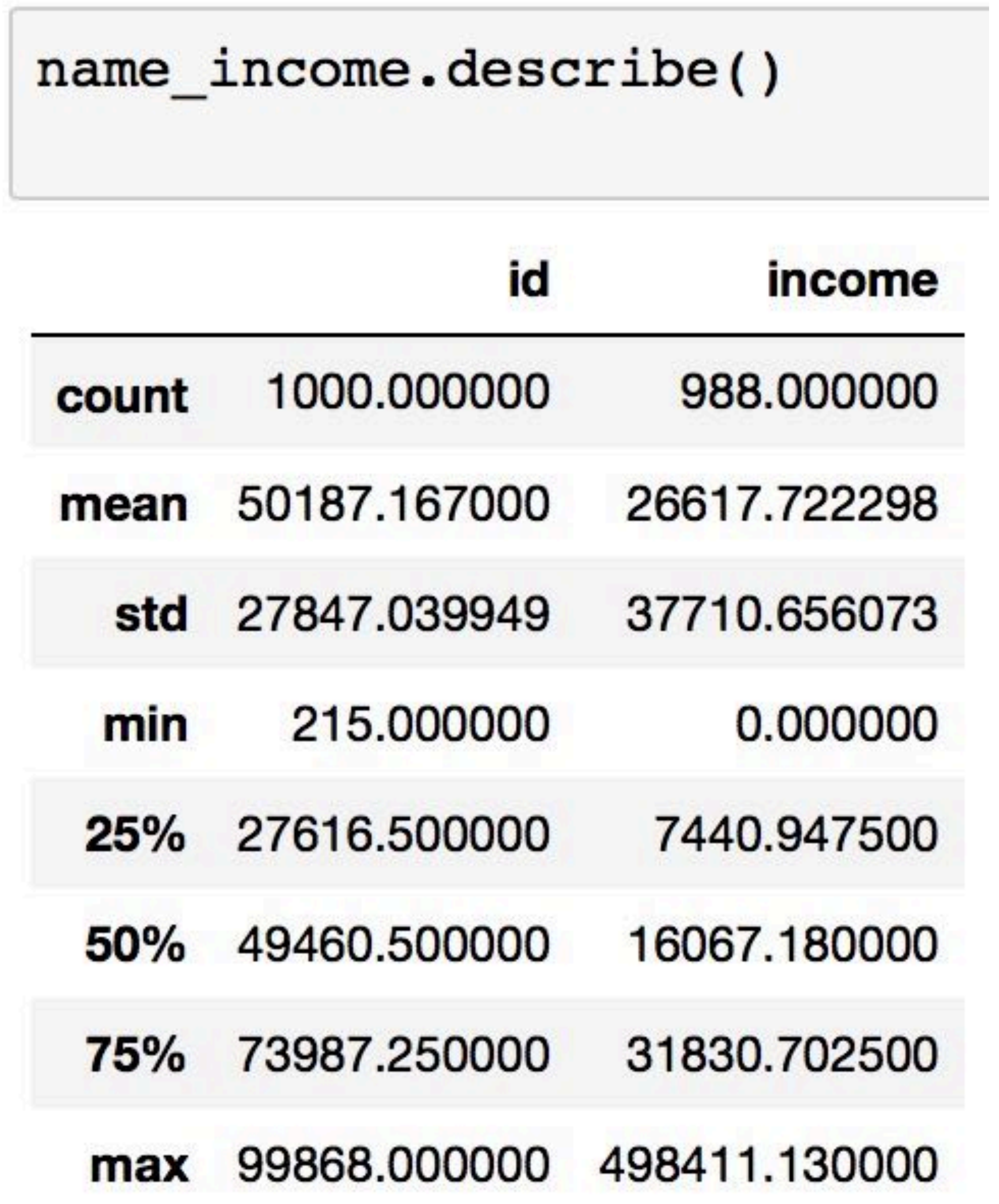
```
name_income
```

	first_name	id	income	last_name
0	Jason	22453	13024.21	Adams
1	Paula	49241	61877.51	Adams
2	Louise	39570	48288.61	Alexander
3	Brian	29963	47622.14	Alexander
4	Willie	94698	55014.69	Alexander
5	Jane	83527	7294.28	Alexander
6	Ashley	77974	0.00	Alexander
7	Albert	81933	10307.08	Allen
8	Matthew	41611	50897.85	Allen
9	Brenda	58[illegible]09	NaN	Allen
10	Evelyn	60011	[illegible]	Anderson



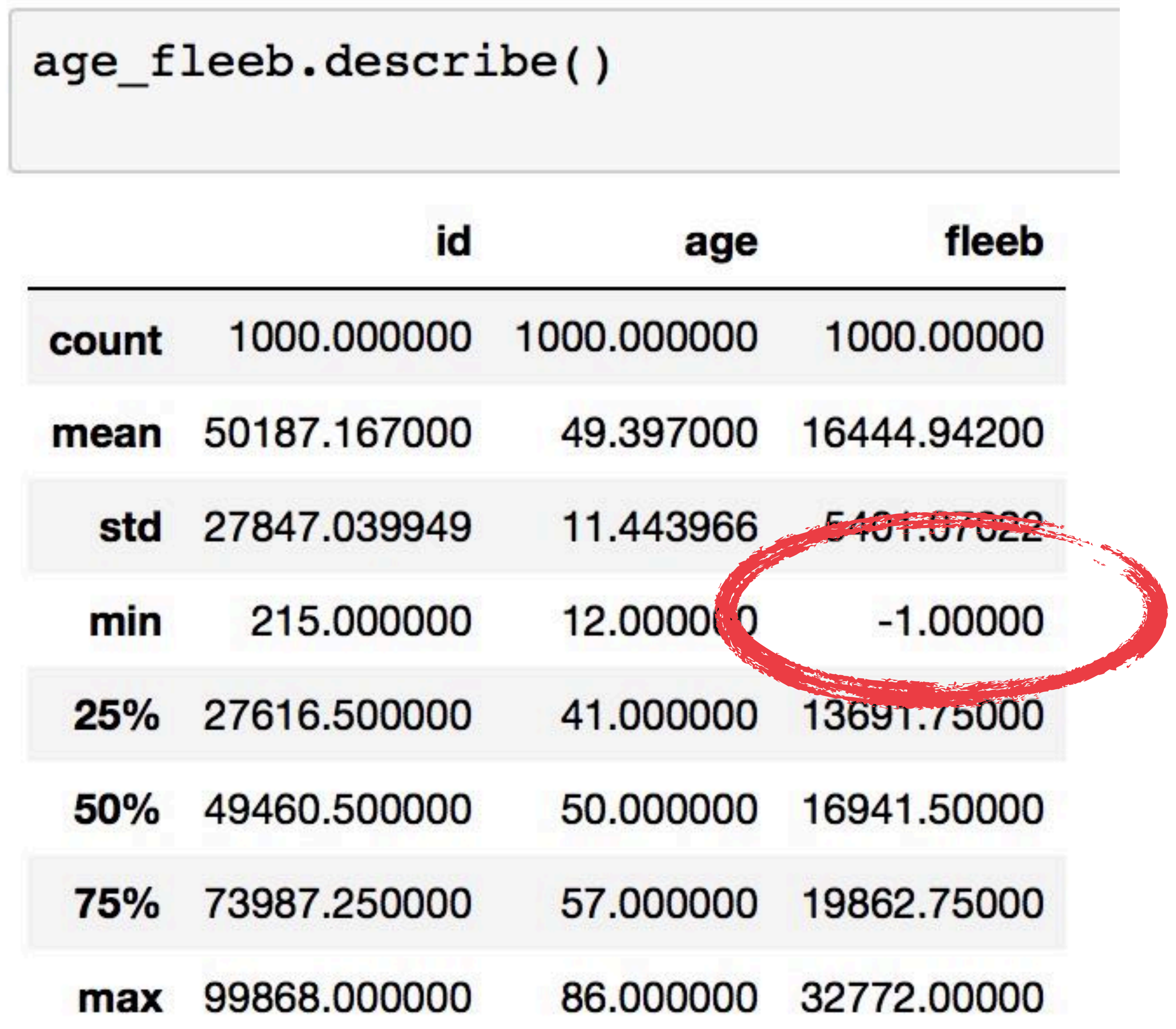
```
np.count_nonzero(~np.isnan(name_income['income']))
988
```

Describing data



```
name_income.describe()
```

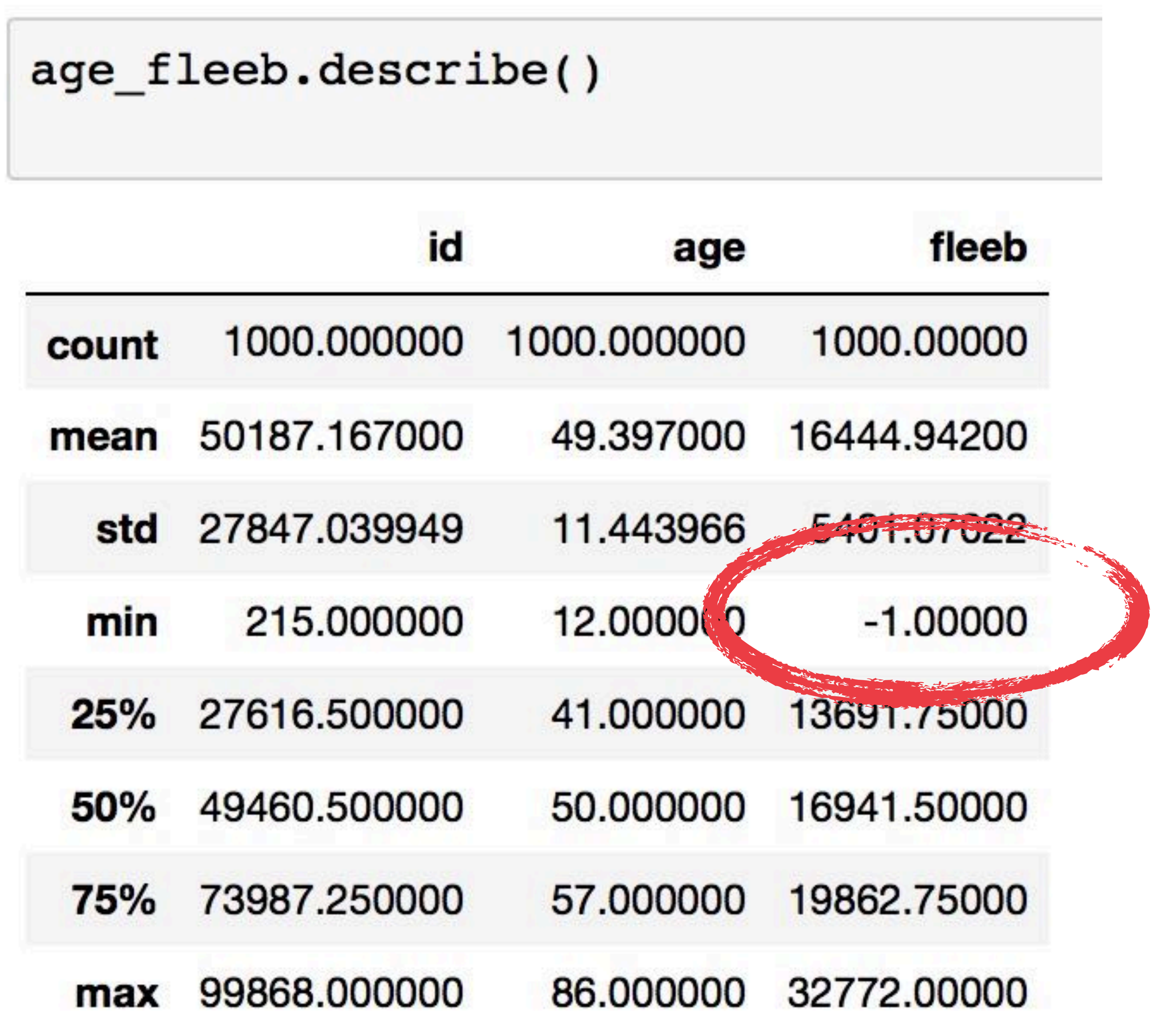
	id	income
count	1000.000000	988.000000
mean	50187.167000	26617.722298
std	27847.039949	37710.656073
min	215.000000	0.000000
25%	27616.500000	7440.947500
50%	49460.500000	16067.180000
75%	73987.250000	31830.702500
max	99868.000000	498411.130000



```
age_fleeb.describe()
```

	id	age	fleeb
count	1000.000000	1000.000000	1000.00000
mean	50187.167000	49.397000	16444.94200
std	27847.039949	11.443966	5461.07022
min	215.000000	12.000000	-1.00000
25%	27616.500000	41.000000	13691.75000
50%	49460.500000	50.000000	16941.50000
75%	73987.250000	57.000000	19862.75000
max	99868.000000	86.000000	32772.00000

Describing data

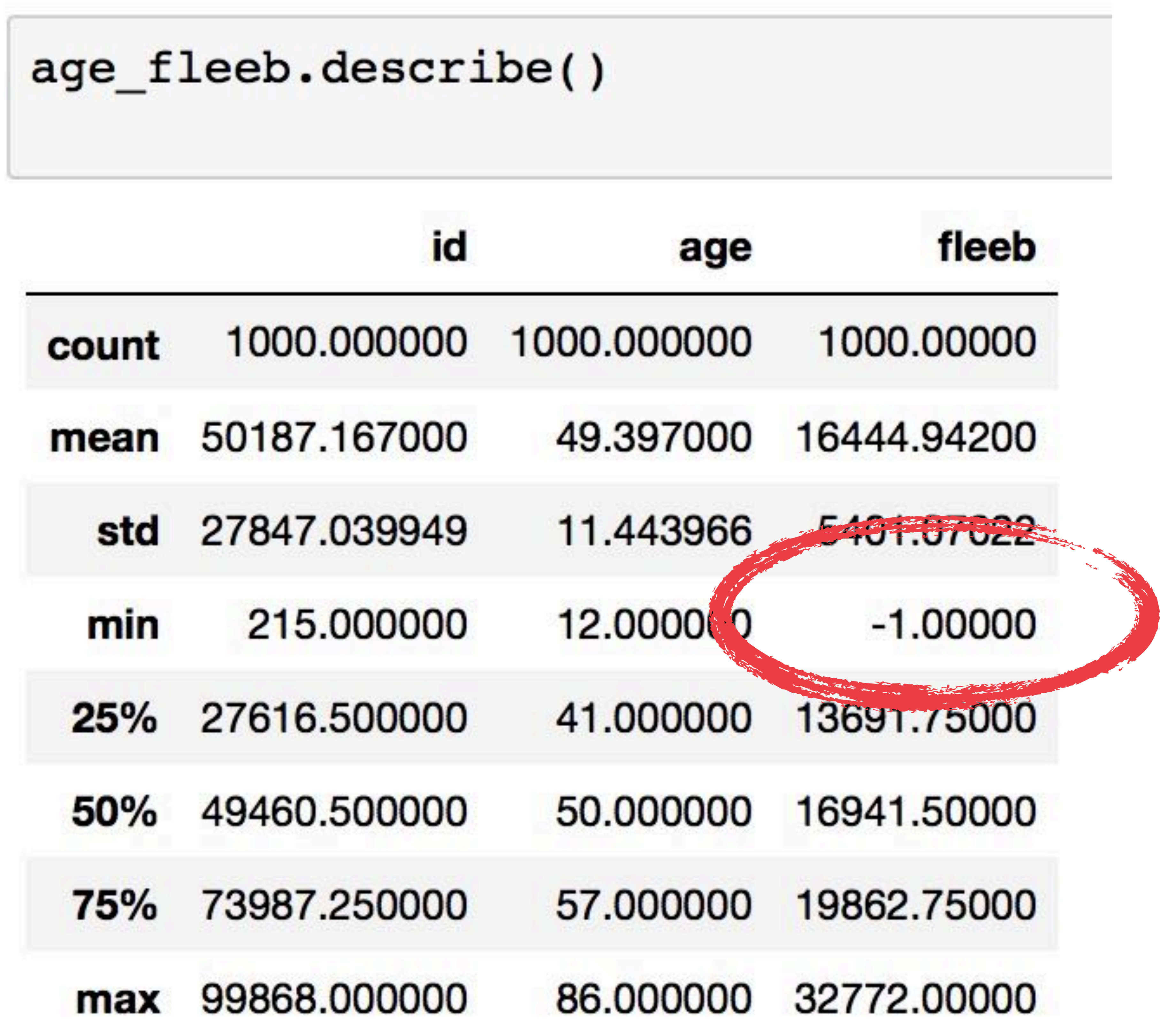


```
age_fleeb.describe()
```

	id	age	fleeb
count	1000.000000	1000.000000	1000.00000
mean	50187.167000	49.397000	16444.94200
std	27847.039949	11.443966	5401.07622
min	215.000000	12.000000	-1.00000
25%	27616.500000	41.000000	13691.75000
50%	49460.500000	50.000000	16941.50000
75%	73987.250000	57.000000	19862.75000
max	99868.000000	86.000000	32772.00000

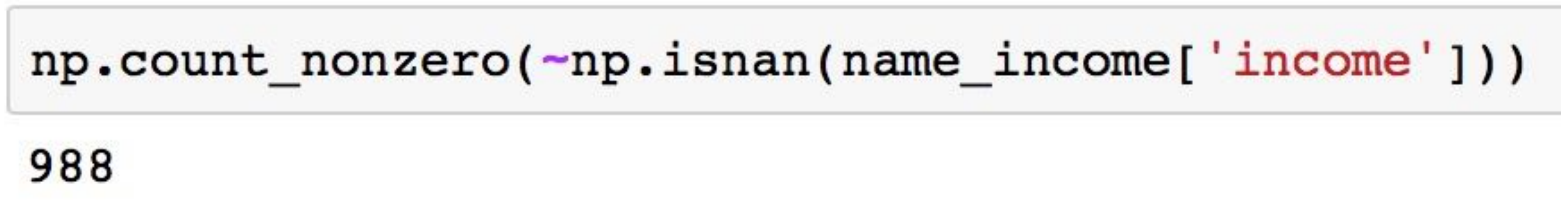
UC San Diego

Describing data



```
age_fleeb.describe()
```

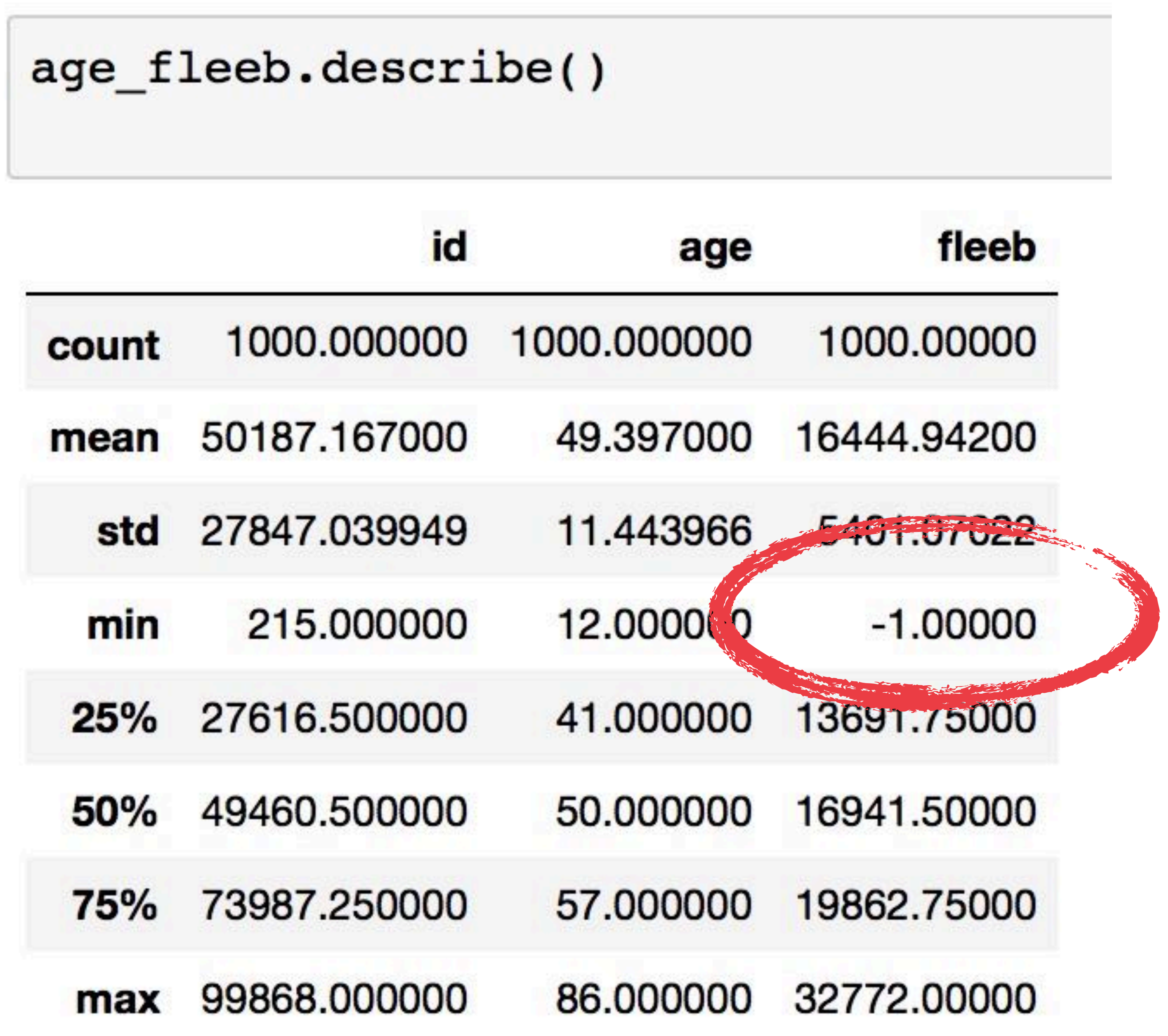
	id	age	fleeb
count	1000.000000	1000.000000	1000.00000
mean	50187.167000	49.397000	16444.94200
std	27847.039949	11.443966	5401.07022
min	215.000000	12.000000	-1.00000
25%	27616.500000	41.000000	13691.75000
50%	49460.500000	50.000000	16941.50000
75%	73987.250000	57.000000	19862.75000
max	99868.000000	86.000000	32772.00000



```
np.count_nonzero(~np.isnan(name_income['income']))
```

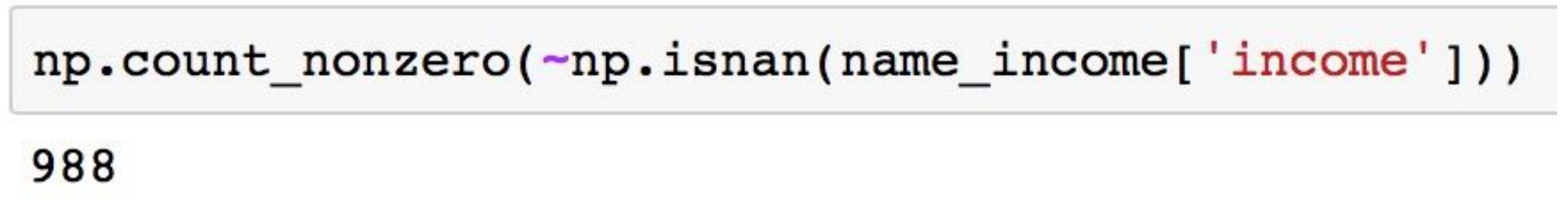
```
988
```

Describing data



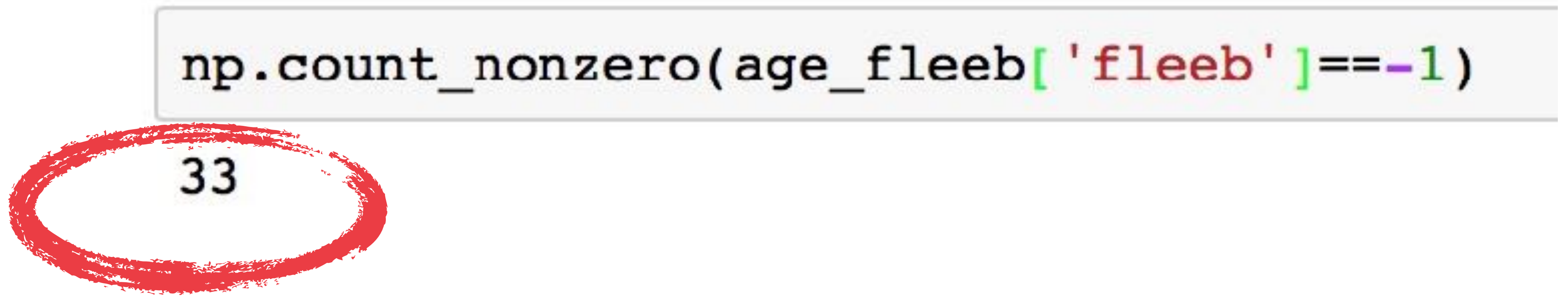
```
age_fleeb.describe()
```

	id	age	fleeb
count	1000.000000	1000.000000	1000.00000
mean	50187.167000	49.397000	16444.94200
std	27847.039949	11.443966	5401.07022
min	215.000000	12.000000	-1.00000
25%	27616.500000	41.000000	13691.75000
50%	49460.500000	50.000000	16941.50000
75%	73987.250000	57.000000	19862.75000
max	99868.000000	86.000000	32772.00000



```
np.count_nonzero(~np.isnan(name_income['income']))
```

```
988
```

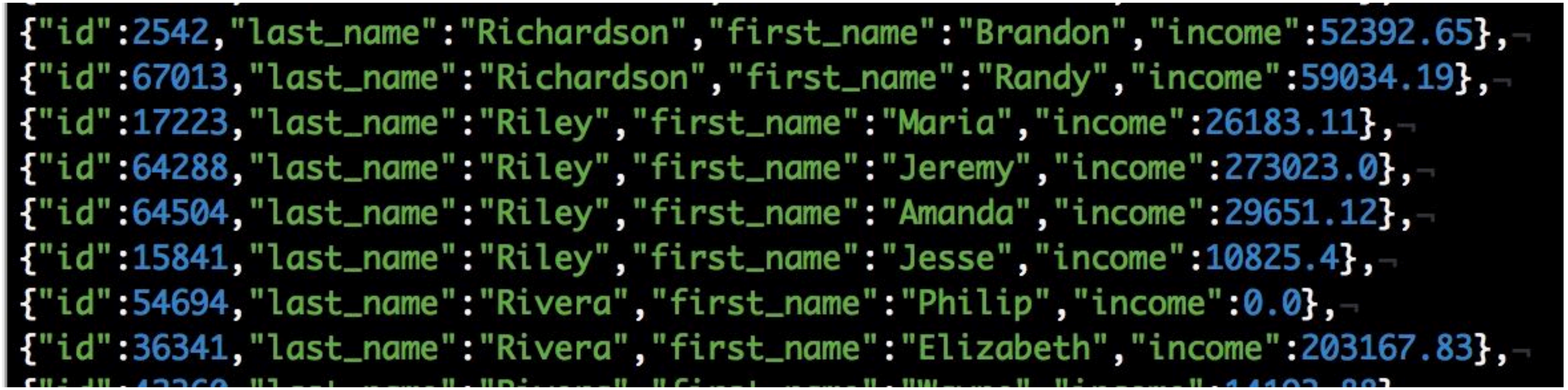


```
np.count_nonzero(age_fleeb['fleeb']==-1)
```

```
33
```

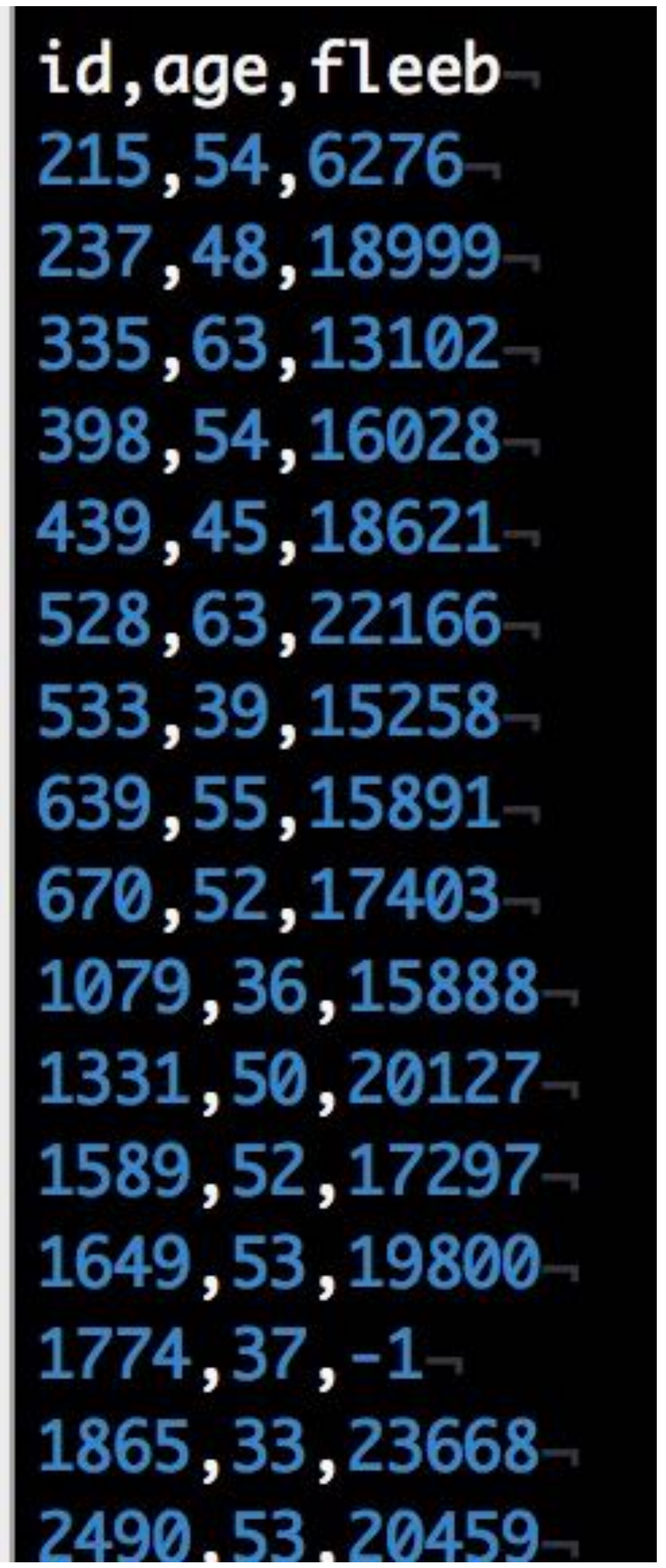
Two datasets

JSON



```
{"id":2542,"last_name":"Richardson","first_name":"Brandon","income":52392.65},
{"id":67013,"last_name":"Richardson","first_name":"Randy","income":59034.19},
{"id":17223,"last_name":"Riley","first_name":"Maria","income":26183.11},
{"id":64288,"last_name":"Riley","first_name":"Jeremy","income":273023.0},
{"id":64504,"last_name":"Riley","first_name":"Amanda","income":29651.12},
{"id":15841,"last_name":"Riley","first_name":"Jesse","income":10825.4},
{"id":54694,"last_name":"Rivera","first_name":"Philip","income":0.0},
{"id":36341,"last_name":"Rivera","first_name":"Elizabeth","income":203167.83},
```

CSV



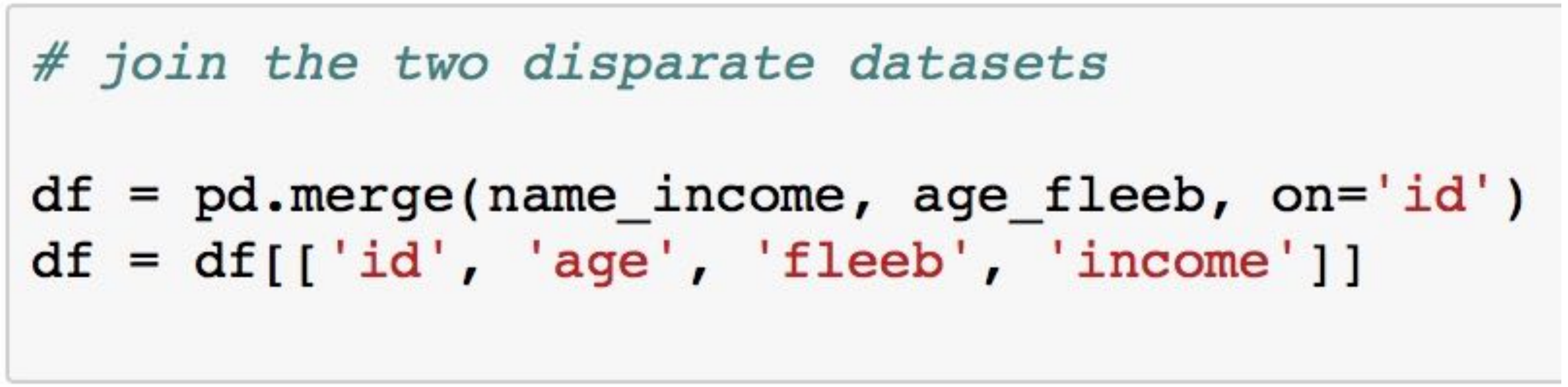
```
id,age,fleeb
215,54,6276
237,48,18999
335,63,13102
398,54,16028
439,45,18621
528,63,22166
533,39,15258
639,55,15891
670,52,17403
1079,36,15888
1331,50,20127
1589,52,17297
1649,53,19800
1774,37,-1
1865,33,23668
2490,53,20459
```

How do you join the two datasets?

- Whiteboard example comparing, e.g., Matlab and manual array creation *vs.* SQL-style joining in pandas

How do you join the two datasets?

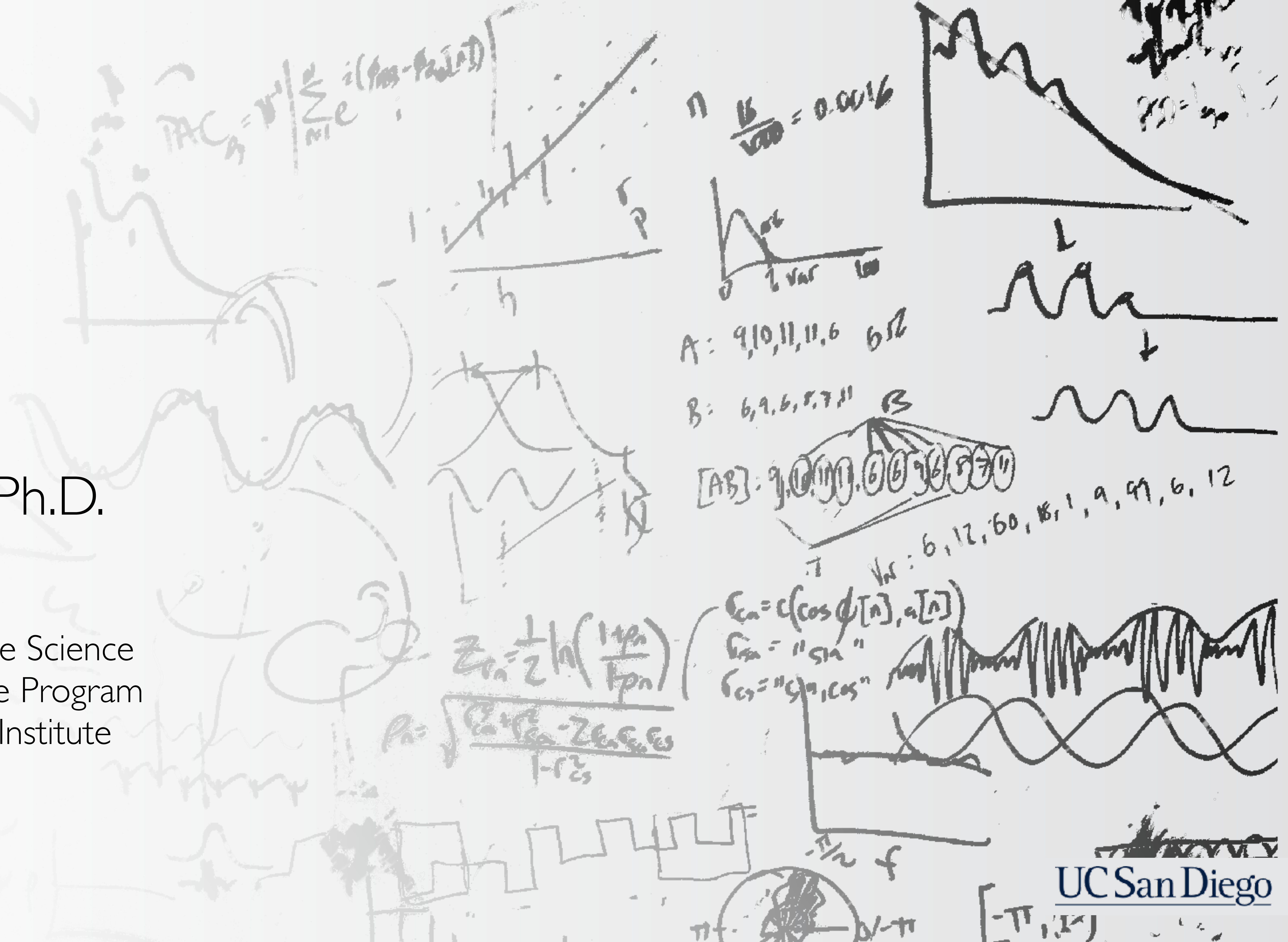
- Whiteboard example comparing, e.g., Matlab and manual array creation *vs.* SQL-style joining in pandas



```
# join the two disparate datasets

df = pd.merge(name_income, age_fleeb, on='id')
df = df[['id', 'age', 'fleeb', 'income']]
```

Administrative stuff

- **REQUIRED LECTURE ANNOUNCEMENT!**
    - Kevin Novak: Chief Data Officer, Tala (Formerly: Head of Data & Engineering, Uber)
    - 2018 January 25 (Thursday)



Bradley Voytek, Ph.D.

UC San Diego

Department of Cognitive Science
Neurosciences Graduate Program
Halicioglu Data Science Institute

bvoytek@ucsd.edu
@bradleyvoytek

UC San Diego