Learn ai

Introduction to Pandas

Lesson 3: Data Manipulation & Processing

An AI Commons initiative

AI

What is Pandas?

- Officially stands for ‘Python Data Analysis Library’
- Powerful Python library for data manipulation and processing
- Your data’s home

Pandas install + import

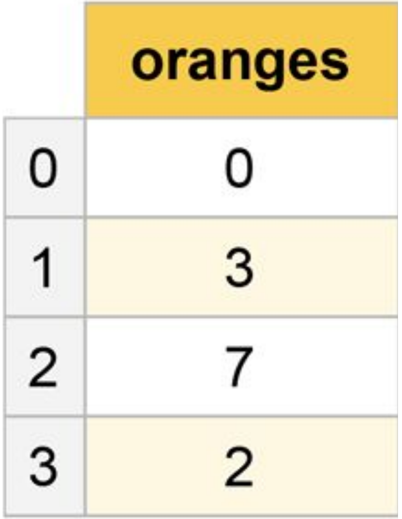
```
conda install pandas or  pip install pandas ← from terminal / command line
```

```
!pip install pandas ← in jupyter notebook / Colab notebook
```

```
import pandas as pd    ← or can just import pandas in the Colab
```

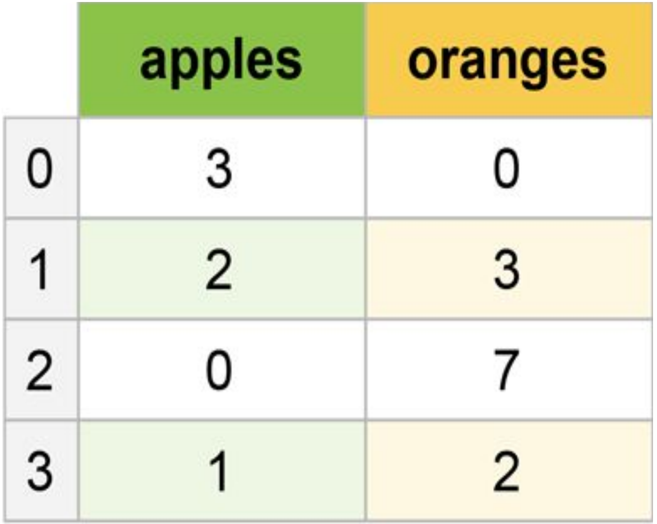
Core components of Pandas

Series



	oranges
0	0
1	3
2	7
3	2

Dataframe



	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Source: https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/

Creating series from scratch

INPUT:
```
pd.Series([1, 3, 5, np.nan, 6, 8])
```

OUTPUT:

0	1.0
1	3.0
2	5.0
3	NaN
4	6.0
5	8.0

Source: https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html

Creating dataframes from scratch

INPUT:

```
data = { 'Amy': [80, 60, 90, 80],  'Ben': [78, 50, 100, 80]}
```

```
marks = pd.DataFrame(data)
```

```
marks
```

OUT:

	Amy	Ben
0	80	78
1	60	50
2	90	100
3	80	80

Read and export to different formats

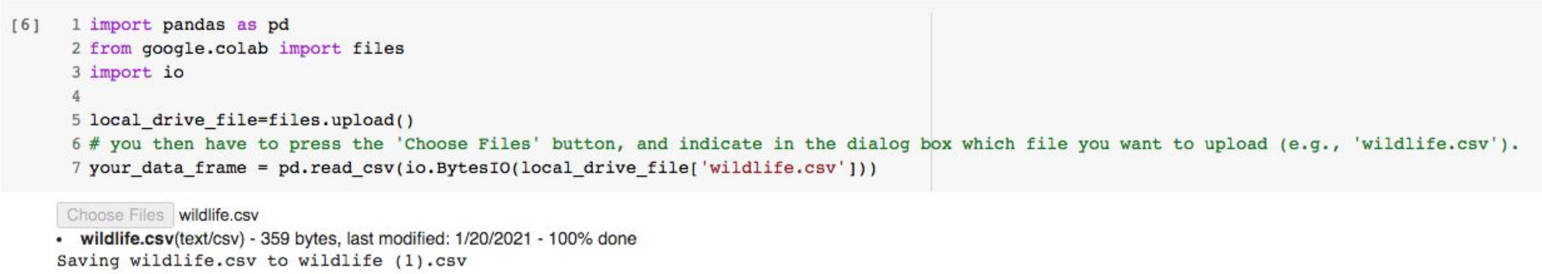
Data format	Read	Save
.csv	`pd.read_csv()`	`pd.to_csv()`
json	`pd.read_json()`	`pd.to_json()`
Excel	`pd.read_excel()`	`pd.to_excel()`
SQL	`pd.read_sql()`	`pd.to_sql()`

Importing .csv files in Colab

There are different ways of importing .csv files to a colab environment.

For example:

- Your . csv file is on a local drive:

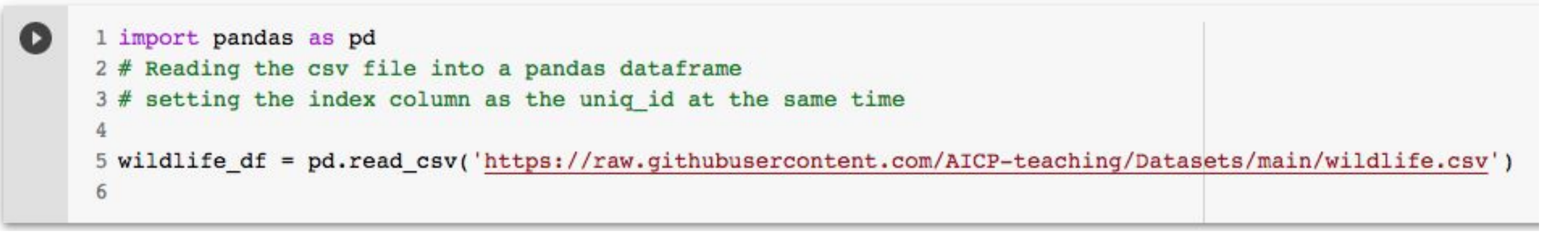


```
[6]
import pandas as pd
from google.colab import files
import io

local_drive_file=files.upload()
# you then have to press the 'Choose Files' button, and indicate in the dialog box which file you want to upload (e.g., 'wildlife.csv').
your_data_frame = pd.read_csv(io.BytesIO(local_drive_file['wildlife.csv']))
```

```
wildlife.csv
• wildlife.csv(text/csv) - 359 bytes, last modified: 1/20/2021 - 100% done
Saving wildlife.csv to wildlife (1).csv
```

- Your .csv file is on GitHub:



```
import pandas as pd
# Reading the csv file into a pandas dataframe
# setting the index column as the uniq_id at the same time

wildlife_df = pd.read_csv('https://raw.githubusercontent.com/AICP-teaching/Datasets/main/wildlife.csv')

```

Main questions data scientists face

- How do we view or get info from data?
- What if there are duplicate rows?
- Why and how to change the column names?
- What do I do with missing values?
- What do I do if I just want part of the data frame?

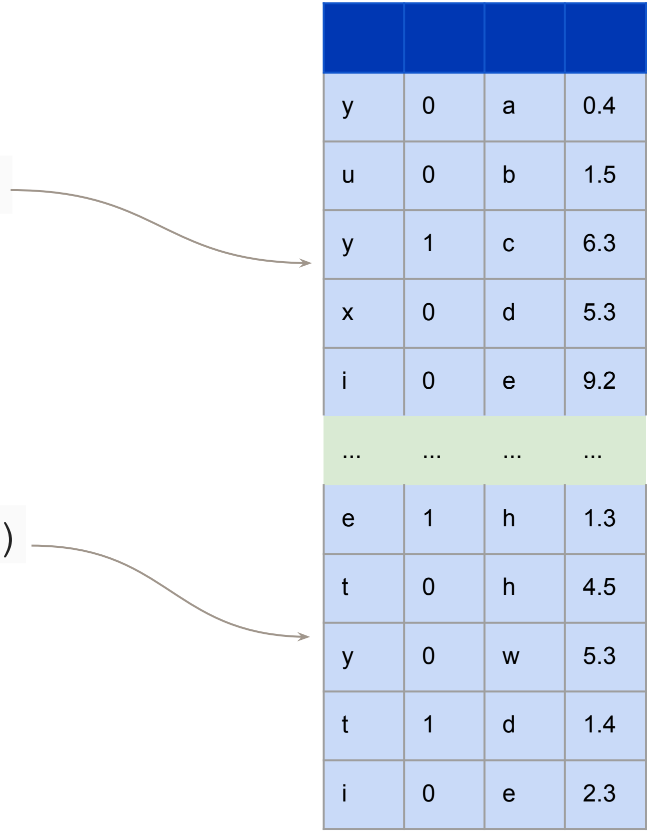
How to view data

head() shows the beginning of a file

```
movies_df.head()
```

tail() shows the end of a file

```
movies_df.tail()
```



y	0	a	0.4
u	0	b	1.5
y	1	c	6.3
x	0	d	5.3
i	0	e	9.2
...	...	...	...
e	1	h	1.3
t	0	h	4.5
y	0	w	5.3
t	1	d	1.4
i	0	e	2.3

Another viewing option in Pandas

```
>>> pandas.set_option('display.max_rows', 10)
```

```
print(movies_df)
```

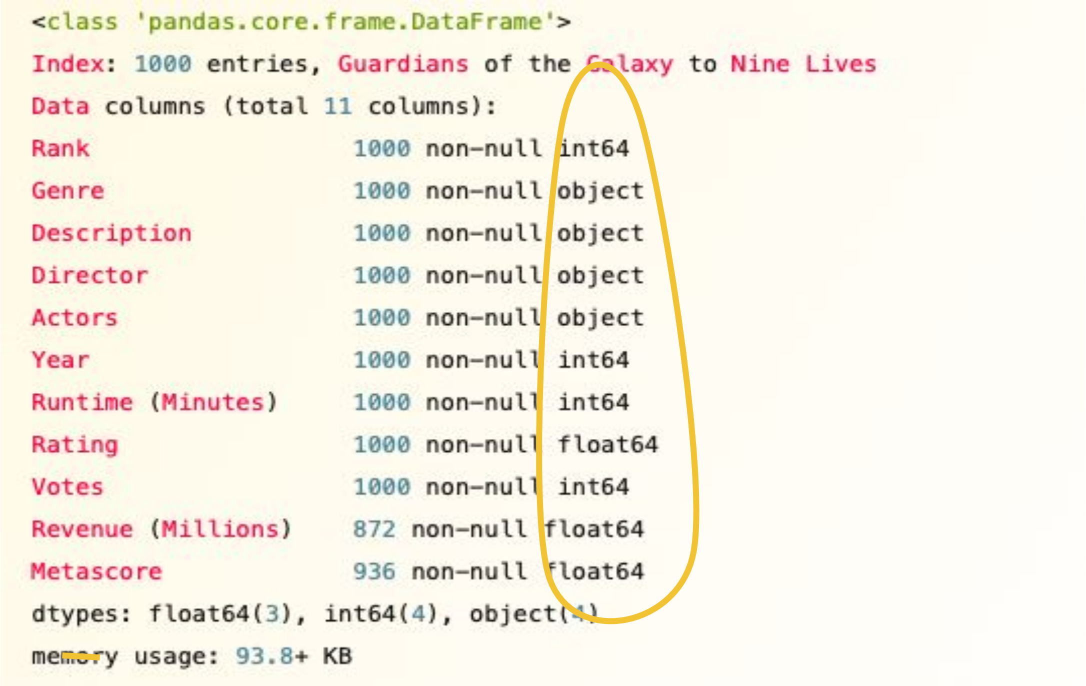
y	0	a	0.4
u	0	b	1.5
y	1	c	6.3
x	0	d	5.3
i	0	e	9.2
...	...	...	...
e	1	h	1.3
t	0	h	4.5
y	0	w	5.3
t	1	d	1.4
i	0	e	2.3

https://dev.to/chanduthedev/how-to-display-all-rows-from-data-frame-using-pandas-dha

Getting info on the data

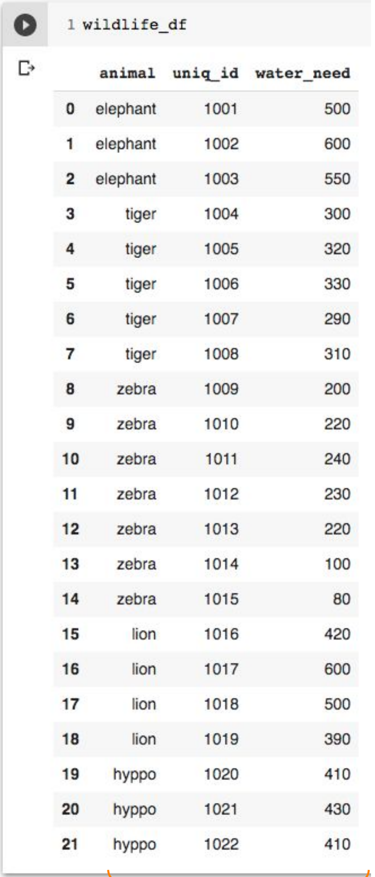
```
movie_df.info()
```

OUTPUT:



```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
Rank                  1000 non-null int64
Genre                 1000 non-null object
Description           1000 non-null object
Director              1000 non-null object
Actors                1000 non-null object
Year                  1000 non-null int64
Runtime (Minutes)     1000 non-null int64
Rating                1000 non-null float64
Votes                 1000 non-null int64
Revenue (Millions)    872 non-null float64
Metascore             936 non-null float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

How to know the shape of the data



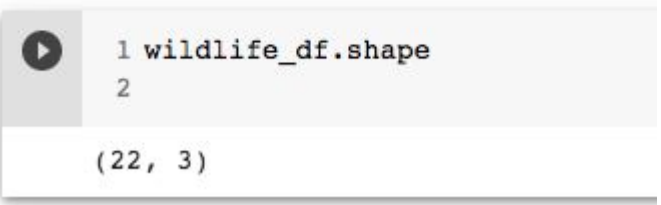
```
wildlife_df
```

	animal	uniq_id	water_need
0	elephant	1001	500
1	elephant	1002	600
2	elephant	1003	550
3	tiger	1004	300
4	tiger	1005	320
5	tiger	1006	330
6	tiger	1007	290
7	tiger	1008	310
8	zebra	1009	200
9	zebra	1010	220
10	zebra	1011	240
11	zebra	1012	230
12	zebra	1013	220
13	zebra	1014	100
14	zebra	1015	80
15	lion	1016	420
16	lion	1017	600
17	lion	1018	500
18	lion	1019	390
19	hyppo	1020	410
20	hyppo	1021	430
21	hyppo	1022	410

22

3

```
(#row, #col)
```

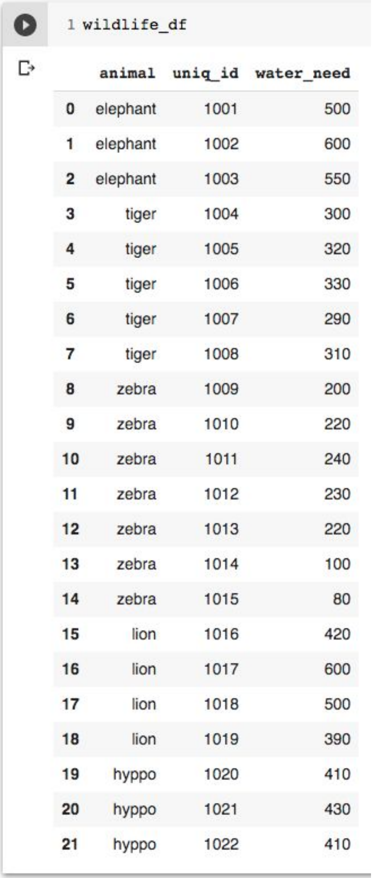


```
wildlife_df.shape

```

```
(22, 3)
```

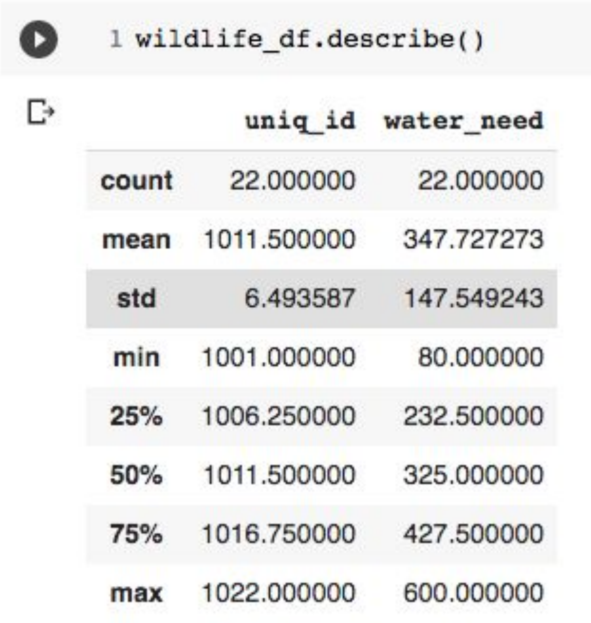
Data’s statistical overview



```python
wildlife_df
```

	animal	uniq_id	water_need
0	elephant	1001	500
1	elephant	1002	600
2	elephant	1003	550
3	tiger	1004	300
4	tiger	1005	320
5	tiger	1006	330
6	tiger	1007	290
7	tiger	1008	310
8	zebra	1009	200
9	zebra	1010	220
10	zebra	1011	240
11	zebra	1012	230
12	zebra	1013	220
13	zebra	1014	100
14	zebra	1015	80
15	lion	1016	420
16	lion	1017	600
17	lion	1018	500
18	lion	1019	390
19	hyppo	1020	410
20	hyppo	1021	430
21	hyppo	1022	410

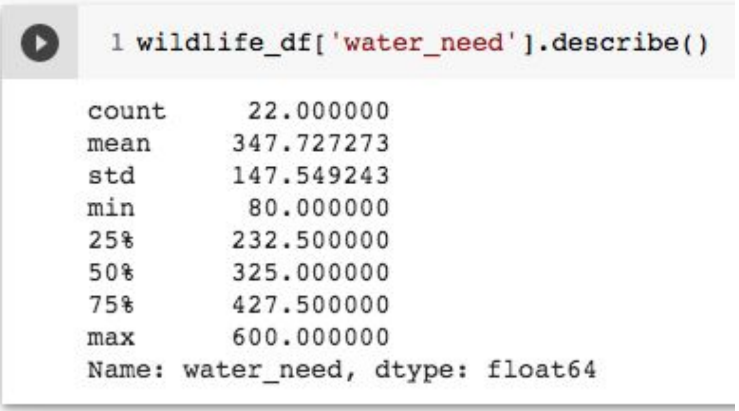
for all data fields



```python
wildlife_df.describe()
```

	uniq_id	water_need
count	22.000000	22.000000
mean	1011.500000	347.727273
std	6.493587	147.549243
min	1001.000000	80.000000
25%	1006.250000	232.500000
50%	1011.500000	325.000000
75%	1016.750000	427.500000
max	1022.000000	600.000000

for specific data fields

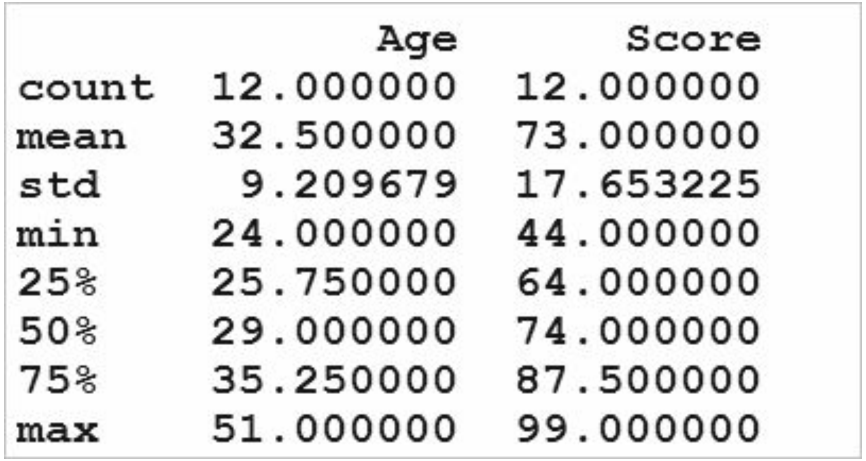


```python
wildlife_df['water_need'].describe()
```

```
count       22.000000
mean       347.727273
std        147.549243
min         80.000000
25%        232.500000
50%        325.000000
75%        427.500000
max        600.000000
Name: water_need, dtype: float64
```

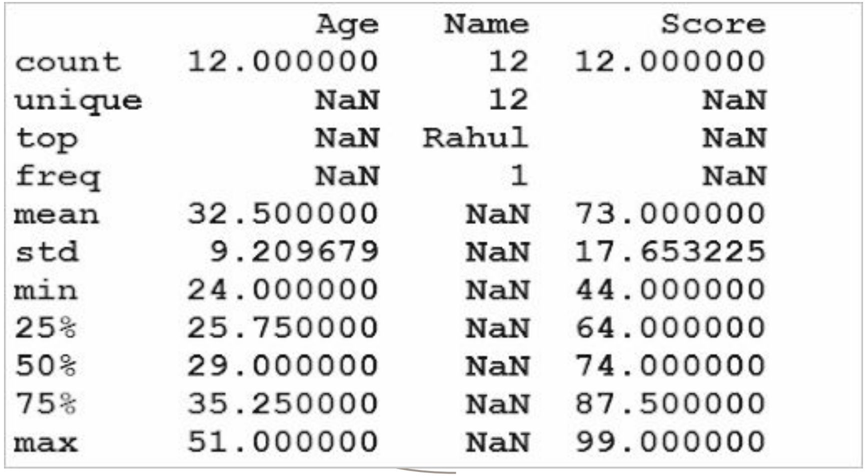
describe() more in depth

df.describe()



```
            Age      Score
count  12.000000  12.000000
mean   32.500000  73.000000
std     9.209679  17.653225
min    24.000000  44.000000
25%    25.750000  64.000000
50%    29.000000  74.000000
75%    35.250000  87.500000
max    51.000000  99.000000
```

df.describe(include = “all”)



```
              Age   Name      Score
count   12.000000     12  12.000000
unique        NaN     12        NaN
top           NaN  Rahul        NaN
freq          NaN      1        NaN
mean    32.500000    NaN  73.000000
std      9.209679    NaN  17.653225
min     24.000000    NaN  44.000000
25%     25.750000    NaN  64.000000
50%     29.000000    NaN  74.000000
75%     35.250000    NaN  87.500000
max     51.000000    NaN  99.000000
```

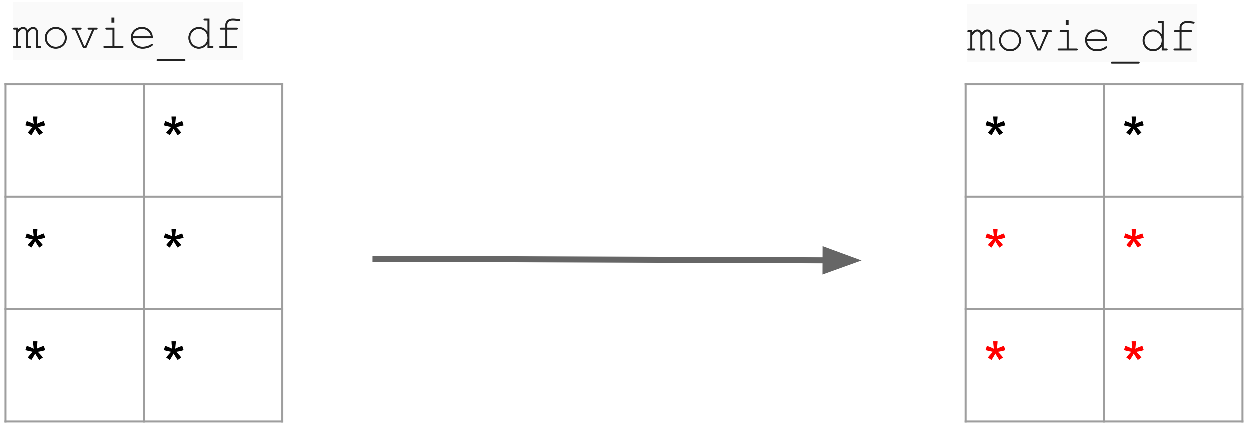
https://images.app.goo.gl/byfMyVrWQKqEsFNk9

https://images.app.goo.gl/6iiMJu9f9ohzVHcK9

Handling duplicates

Duplicates are managed using drop_duplicates()

```
movie_df.drop_duplicates( inplace = True   )
```



movie_df

*	*
*	*
*	*

movie_df

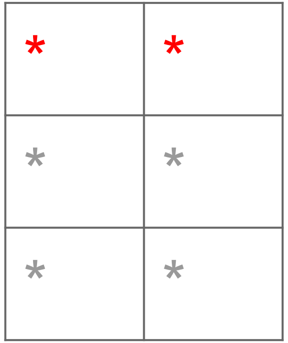
*	*
*	*
*	*

drop_duplicates() options

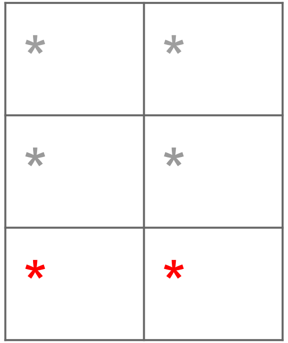
```
movie_df.drop_duplicates(keep =  ???)
```

- `first`: Drop duplicates except for the first occurrence. **(default)**
- `last`: Drop duplicates except for the last occurrence.
- `False`: Drop all duplicates.

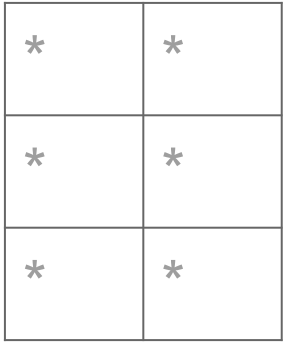
Movie_df - **first**



Movie_df - **last**



Movie_df - **False**



Get columns names with .columns

IN:

	Popularity	Movie_type	Screening
Title			
Kati kati	2	Drama	2016
The letter	1	Documentary	2019

```
.columns
```

OUT:

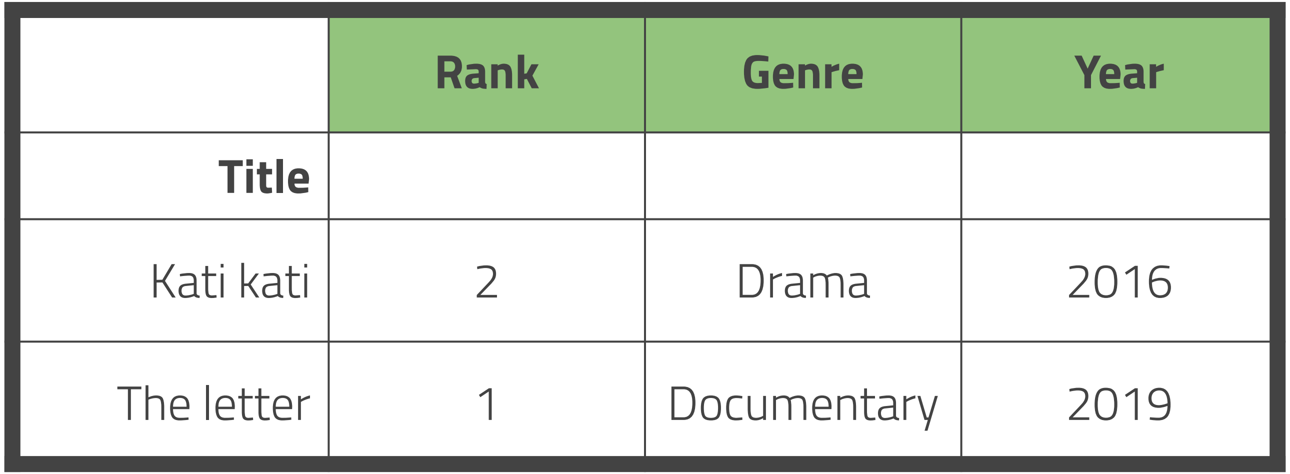
```
Index([’Popularity’, 'Movie_type', 'Screening’], dtype='object')
```

Rename columns

```
1.    movies_df.rename(columns={
                ‘Popularity': 'Rank',
                'Movie_type': 'Genre'
                ‘Screening’: ‘Year’ },inplace=True)
```

Or …

```
2.    movies_df.columns =     [‘Rank’, ‘Genre’, ‘Year’]
```

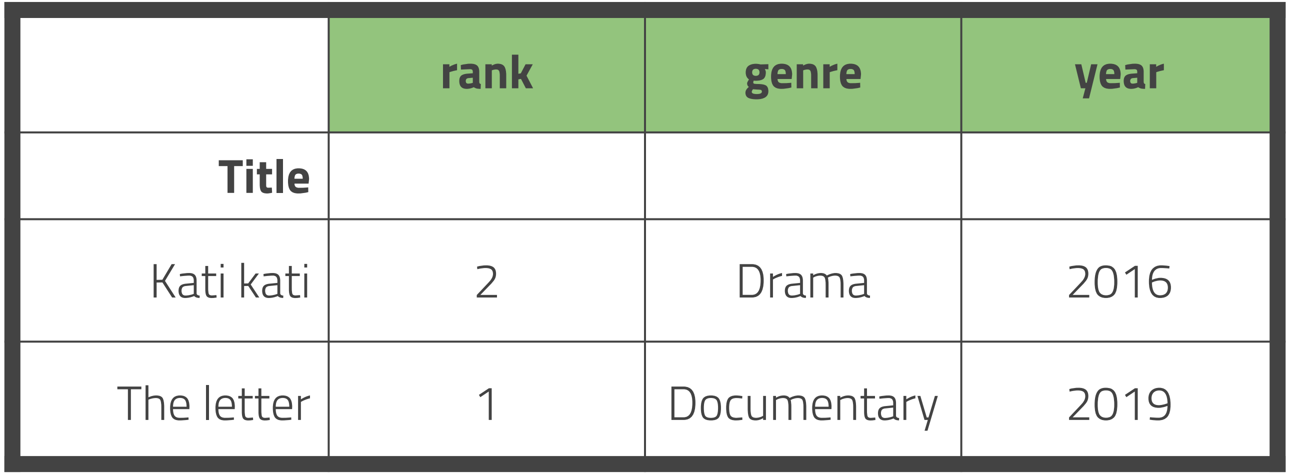


	Rank	Genre	Year
Title			
Kati kati	2	Drama	2016
The letter	1	Documentary	2019

Using list comprehension

You can iterate over an iterable, to modify a member object using an expression

```
movies_df.columns = [col.lower() for col in movies_df]
```

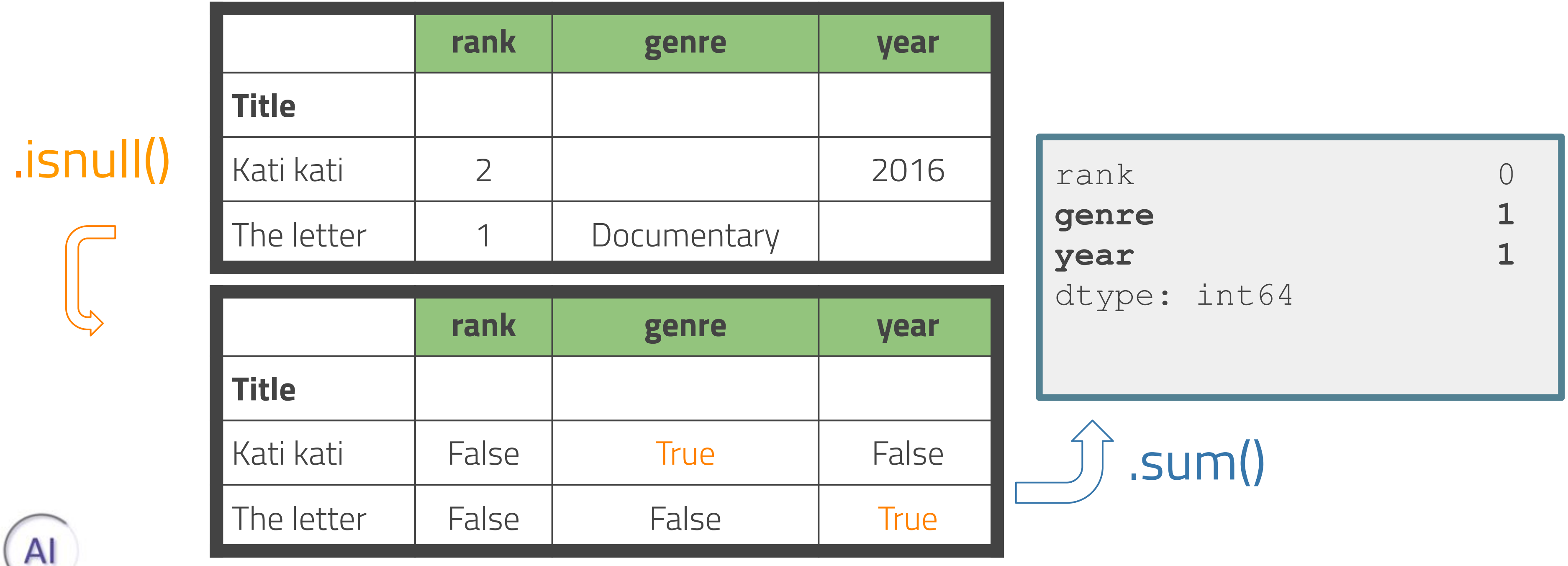


	rank	genre	year
Title			
Kati kati	2	Drama	2016
The letter	1	Documentary	2019

Identify missing values

Where are the missing values?

movies_df.isnull().sum()



	rank	genre	year
Title			
Kati kati	2		2016
The letter	1	Documentary	

	rank	genre	year
Title			
Kati kati	False	True	False
The letter	False	False	True

```
rank         0
genre        1
year         1
dtype: int64
```

Removing missing values in rows

Before .dropna():

	rank	genre	year
Title			
Kati kati	2		
The letter	1	Documentary	2019

.dropna()

After .dropna():

	rank	genre	year
Title			
The letter	1	Documentary	2019

.dropna() is equal to .dropna(axis=0). ‘axis’ refers to the shape: 0=row, 1=column

Removing missing values in columns

Before .dropna(axis = 1):

	rank	**genre**	**year**
Title			
Kati kati	2		
The letter	1	Documentary	2019

.dropna(axis=1)

After .dropna(axis = 1):

	rank
Title	
Kati kati	2
The letter	1

Filling missing values (Imputation)

	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9
Nairobi half-life	3	Drama	2012	
Soul boy	4	Drama	2010	1.2

col

col.mean() = 5.23

```
col = movie_df[‘revenues_m$’]
mean=col.mean()
col.fillna(mean, inplace = True)
```

```
col.fillna(movie_df[‘revenues_m$’] .mean(), inplace=True)
```

⇨

revenues_m$
5.6
8.9
5.23
1.2

Series and dataframes

```
movies_df['genre']
```

```
movies_df[['genre']]
```

```
type: series
```

```
type: dataframe
```

	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9
Nairobi half-life	3	Drama	2012	4.3
Soul boy	4	Drama	2010	1.2

Extracting columns

```
movies_df[['genre','year']]
type: dataframe
```

	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9
Nairobi half-life	3	Drama	2012	4.3
Soul boy	4	Drama	2010	1.2

	genre	year
Title		
Kati kati	Drama	2016
The letter	Documentary	2019
Nairobi half-life	Drama	2012
Soul boy	Drama	2010

Extract a row by name

	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9
Nairobi half-life	3	Drama	2012	4.3
Soul boy	4	Drama	2010	1.2

```
movie_df.loc['The letter']
```

	rank	genre	year	revenues_m$
Title				
The letter	1	Documentary	2019	8.9

Extract a row by index

	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9
Nairobi half-life	3	Drama	2012	4.3
Soul boy	4	Drama	2010	1.2

```
movie_df.iloc[1]
```

	rank	genre	year	revenues_m$
Title				
The letter	1	Documentary	2019	8.9

Extract multiple rows

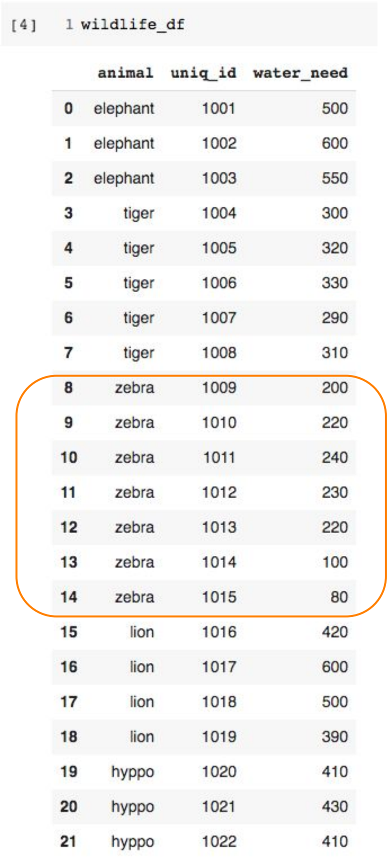
	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9
Nairobi half-life	3	Drama	2012	4.3

```
movie_df.iloc[:1]
```

```
movie_df.loc[‘Kati kati’, ‘The letter’]
```

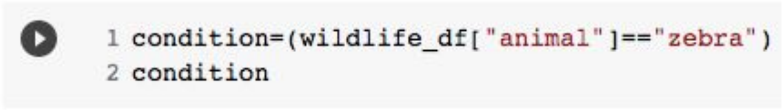
	rank	genre	year	revenues_m$
Title				
Kati kati	2	Drama	2016	5.6
The letter	1	Documentary	2019	8.9

Conditional selections dataframes

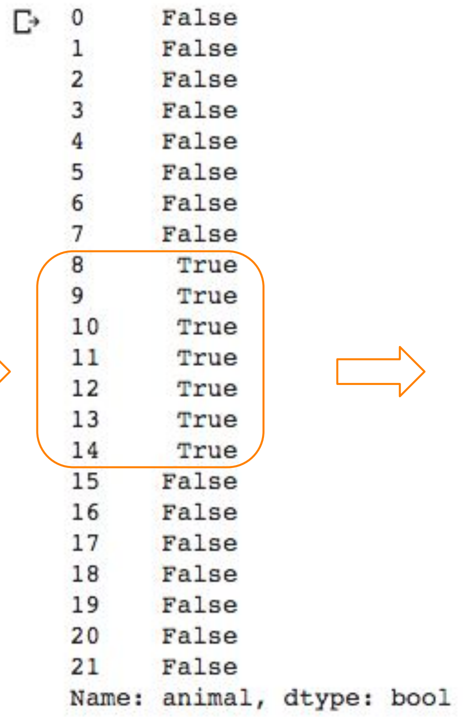


```
[4]  wildlife_df
```

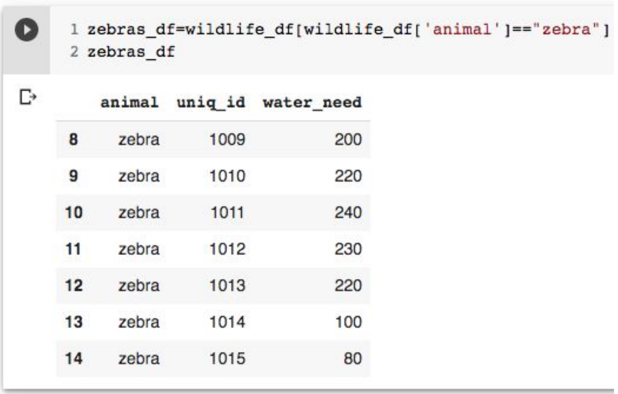
	animal	uniq_id	water_need
0	elephant	1001	500
1	elephant	1002	600
2	elephant	1003	550
3	tiger	1004	300
4	tiger	1005	320
5	tiger	1006	330
6	tiger	1007	290
7	tiger	1008	310
8	zebra	1009	200
9	zebra	1010	220
10	zebra	1011	240
11	zebra	1012	230
12	zebra	1013	220
13	zebra	1014	100
14	zebra	1015	80
15	lion	1016	420
16	lion	1017	600
17	lion	1018	500
18	lion	1019	390
19	hyppo	1020	410
20	hyppo	1021	430
21	hyppo	1022	410



```
condition=(wildlife_df["animal"]=="zebra")
condition
```



```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8       True
9       True
10      True
11      True
12      True
13      True
14      True
15     False
16     False
17     False
18     False
19     False
20     False
21     False
Name: animal, dtype: bool
```

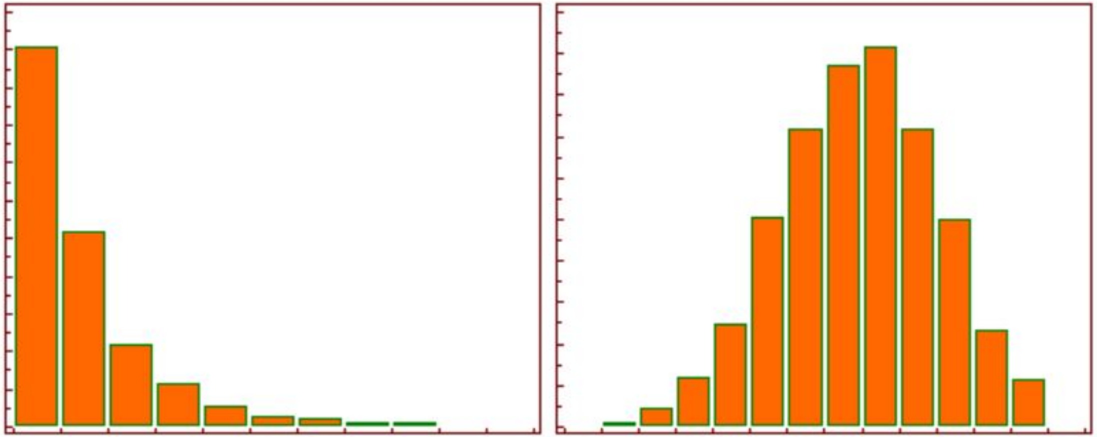


```
zebras_df=wildlife_df[wildlife_df['animal']=="zebra"]
zebras_df
```

	animal	uniq_id	water_need
8	zebra	1009	200
9	zebra	1010	220
10	zebra	1011	240
11	zebra	1012	230
12	zebra	1013	220
13	zebra	1014	100
14	zebra	1015	80

Data Transformation

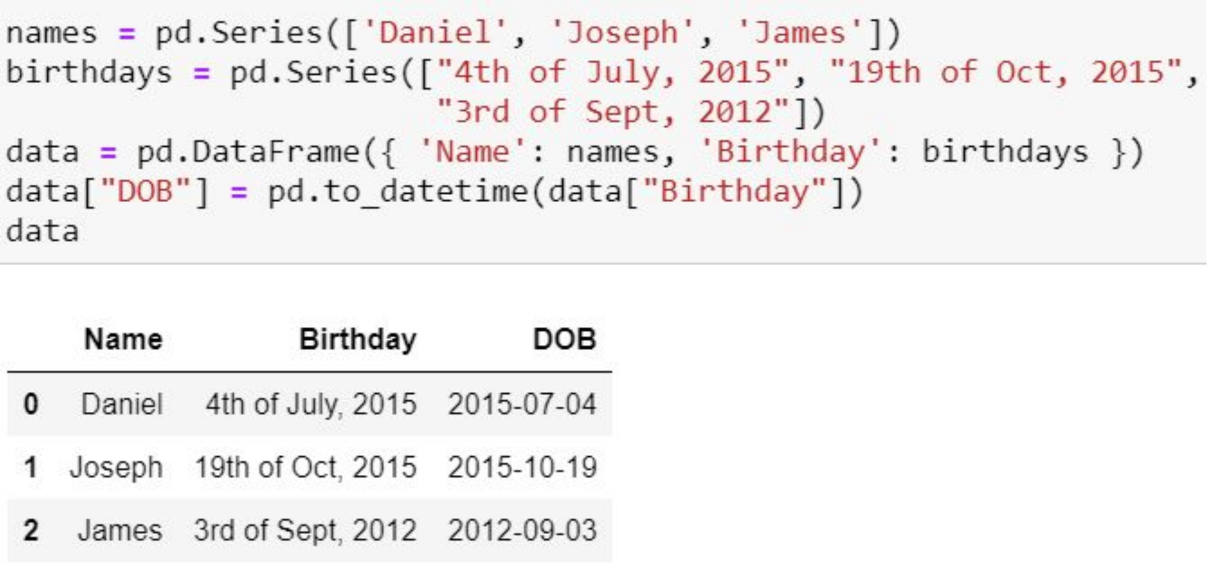
To make the data normally distributed and make them able to meet the assumptions of parametric statistical tests.



The Effect of Log Transformation. Source: https://www.medcalc.org/manual/log_transformation.php

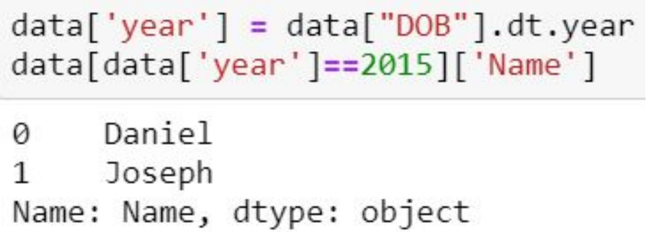
Time Data

Convert time data to a usable format for manipulations.



```
names = pd.Series(['Daniel', 'Joseph', 'James'])
birthdays = pd.Series(["4th of July, 2015", "19th of Oct, 2015",
                       "3rd of Sept, 2012"])
data = pd.DataFrame({ 'Name': names, 'Birthday': birthdays })
data["DOB"] = pd.to_datetime(data["Birthday"])
data
```

	Name	Birthday	DOB
0	Daniel	4th of July, 2015	2015-07-04
1	Joseph	19th of Oct, 2015	2015-10-19
2	James	3rd of Sept, 2012	2012-09-03



```
data['year'] = data["DOB"].dt.year
data[data['year']==2015]['Name']
```

```
0     Daniel
1     Joseph
Name: Name, dtype: object
```

Questions?

Thank you for your attention!

https://images.app.goo.gl/TAaHUsDu1aVW2qc58

References and further readings

- https://pythongeeks.net/python-tutorials/python-pandas-for-beginners-a-complete-guide/
- https://towardsdatascience.com/python-for-data-science-basics-of-pandas-5f8d9680617e
- https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html
- https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
- https://www.dataquest.io/blog/pandas-cheat-sheet/

Contributors

UofT AI

Elias Williams

UNIVERSIDAD DE CHILE

Jose Ignacio Diaz

Université de Montréal

Pavithra Rajasekar

AI Commons

Gilles Fayad