Toward Data Science
Pandas

Outline

- What is Pandas? Why using it?
- Getting started – Data and Installation
- Data manipulation
- Question

What is Pandas?

https://www.kaggle.com/datasets/abcsds/pokemon?resource=download

What is Pandas?

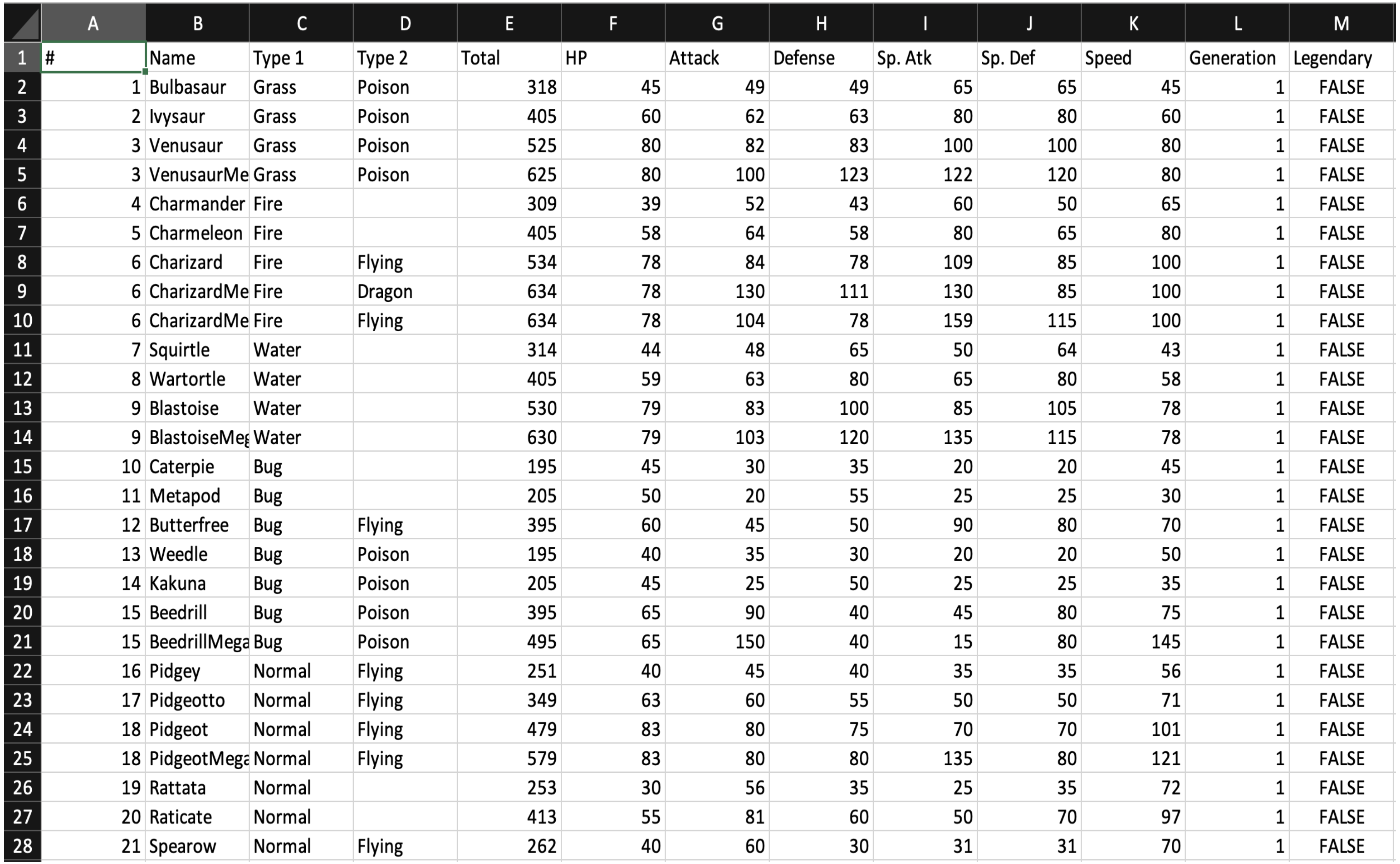
A Python library

Exploring

Manipulating

Cleaning

Analyzing



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
2	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	FALSE
3	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	FALSE
4	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	FALSE
5	3	VenusaurMe	Grass	Poison	625	80	100	123	122	120	80	1	FALSE
6	4	Charmander	Fire		309	39	52	43	60	50	65	1	FALSE
7	5	Charmeleon	Fire		405	58	64	58	80	65	80	1	FALSE
8	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	FALSE
9	6	CharizardMe	Fire	Dragon	634	78	130	111	130	85	100	1	FALSE
10	6	CharizardMe	Fire	Flying	634	78	104	78	159	115	100	1	FALSE
11	7	Squirtle	Water		314	44	48	65	50	64	43	1	FALSE
12	8	Wartortle	Water		405	59	63	80	65	80	58	1	FALSE
13	9	Blastoise	Water		530	79	83	100	85	105	78	1	FALSE
14	9	BlastoiseMeg	Water		630	79	103	120	135	115	78	1	FALSE
15	10	Caterpie	Bug		195	45	30	35	20	20	45	1	FALSE
16	11	Metapod	Bug		205	50	20	55	25	25	30	1	FALSE
17	12	Butterfree	Bug	Flying	395	60	45	50	90	80	70	1	FALSE
18	13	Weedle	Bug	Poison	195	40	35	30	20	20	50	1	FALSE
19	14	Kakuna	Bug	Poison	205	45	25	50	25	25	35	1	FALSE
20	15	Beedrill	Bug	Poison	395	65	90	40	45	80	75	1	FALSE
21	15	BeedrillMega	Bug	Poison	495	65	150	40	15	80	145	1	FALSE
22	16	Pidgey	Normal	Flying	251	40	45	40	35	35	56	1	FALSE
23	17	Pidgeotto	Normal	Flying	349	63	60	55	50	50	71	1	FALSE
24	18	Pidgeot	Normal	Flying	479	83	80	75	70	70	101	1	FALSE
25	18	PidgeotMega	Normal	Flying	579	83	80	80	135	80	121	1	FALSE
26	19	Rattata	Normal		253	30	56	35	25	35	72	1	FALSE
27	20	Raticate	Normal		413	55	81	60	50	70	97	1	FALSE
28	21	Spearow	Normal	Flying	262	40	60	30	31	31	70	1	FALSE

Why Using Pandas

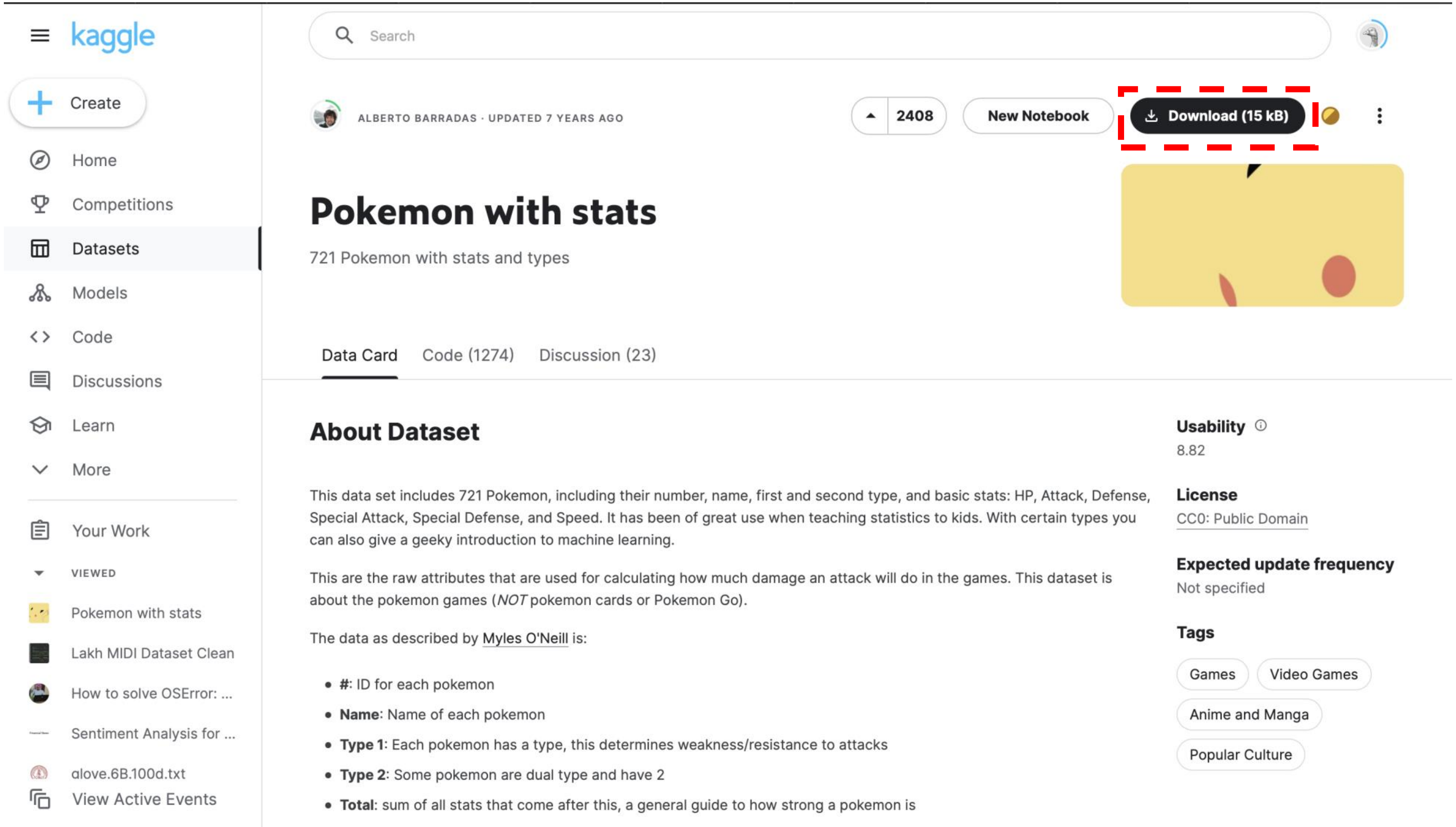
X	pandas
Ease of Use	**Performance and Scalability**
Visualization and Reporting	**Automation and Reproducibility**
Learning Curve	**Customization and Flexibility**
Quick Prototyping	**Integration with Ecosystem**
Data Entry and Formatting	**Version Control**
Compatibility	**Reproducibility and Portability**

Getting Started

https://www.kaggle.com/datasets/abcsds/pokemon?resource=download

Getting started

Data link for today lesson



ALBERTO BARRADAS · UPDATED 7 YEARS AGO

Pokemon with stats

721 Pokemon with stats and types

About Dataset

This data set includes 721 Pokemon, including their number, name, first and second type, and basic stats: HP, Attack, Defense, Special Attack, Special Defense, and Speed. It has been of great use when teaching statistics to kids. With certain types you can also give a geeky introduction to machine learning.

This are the raw attributes that are used for calculating how much damage an attack will do in the games. This dataset is about the pokemon games (*NOT* pokemon cards or Pokemon Go).

The data as described by Myles O'Neill is:

- **#**: ID for each pokemon
- **Name**: Name of each pokemon
- **Type 1**: Each pokemon has a type, this determines weakness/resistance to attacks
- **Type 2**: Some pokemon are dual type and have 2
- **Total**: sum of all stats that come after this, a general guide to how strong a pokemon is

Usability
8.82

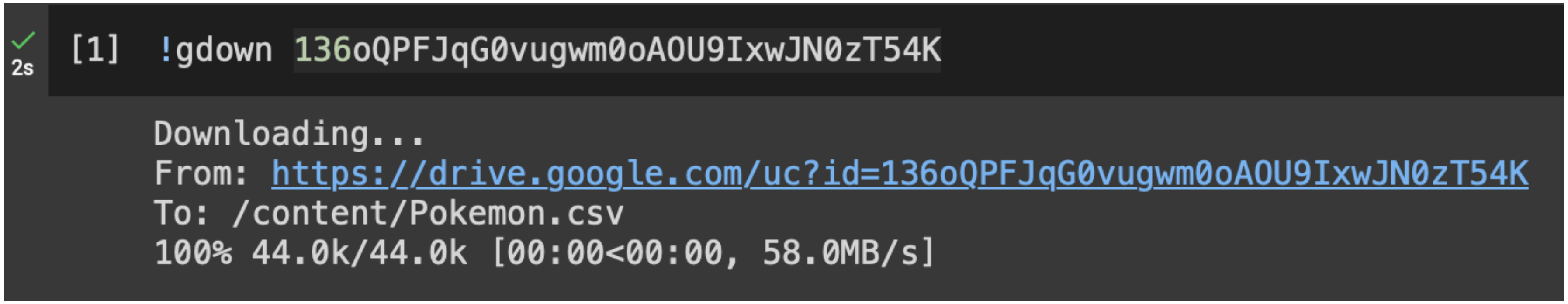
License
CC0: Public Domain

Expected update frequency
Not specified

Tags
Games
Video Games
Anime and Manga
Popular Culture

Getting started

Download data on Colab



```
[1]  !gdown 136oQPFJqG0vugwm0oAOU9IxwJN0zT54K

     Downloading...
     From: https://drive.google.com/uc?id=136oQPFJqG0vugwm0oAOU9IxwJN0zT54K
     To: /content/Pokemon.csv
     100% 44.0k/44.0k [00:00<00:00, 58.0MB/s]
```

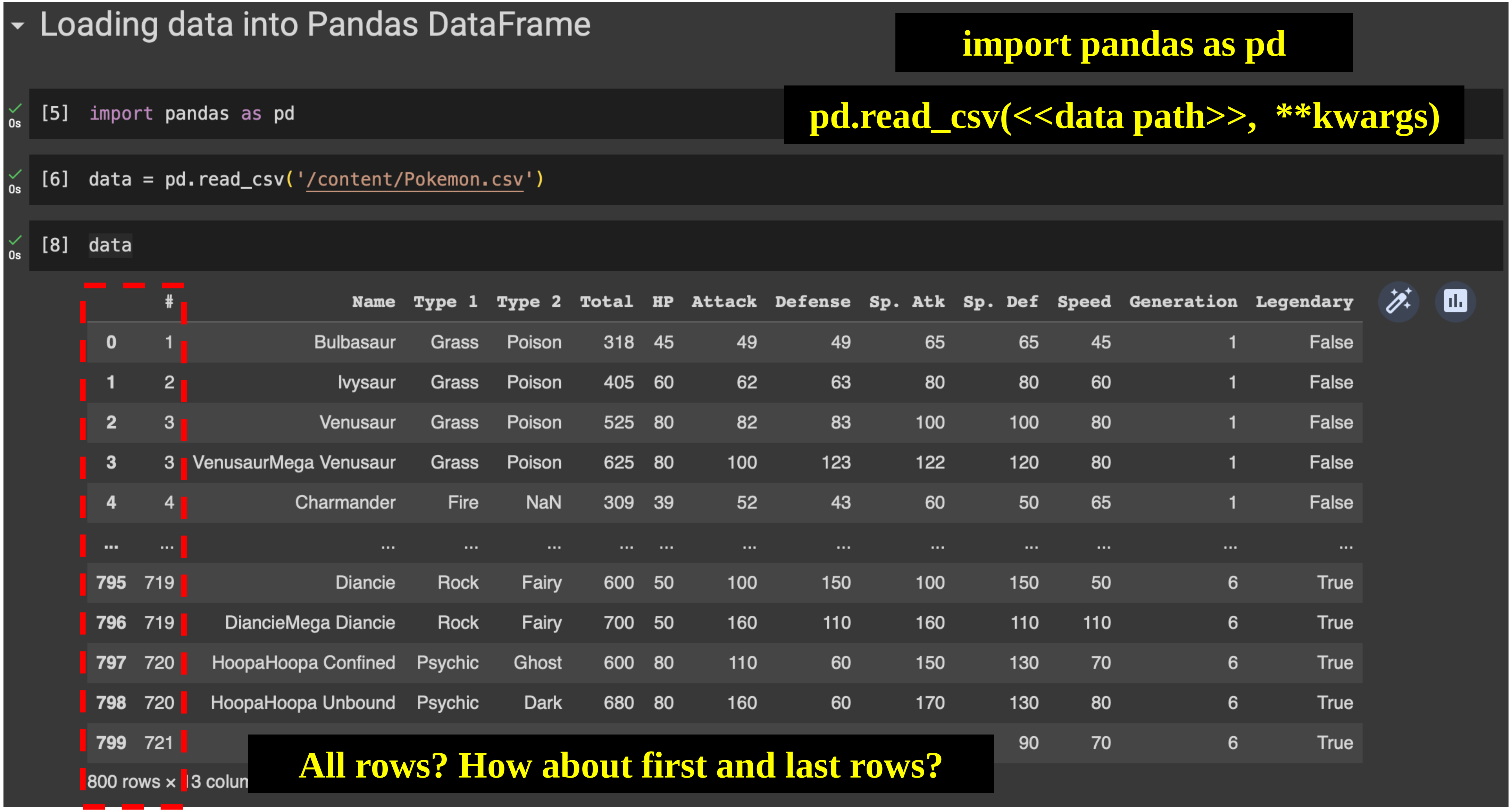
Run the line below in Colab to download the dataset

```
!gdown 136oQPFJqG0vugwm0oAOU9IxwJN0zT54K
```

Read CSV

AI VIET NAM
@aivietnam.edu.vn

Read CSV



Loading data into Pandas DataFrame

import pandas as pd

pd.read_csv(<<data path>>, **kwargs)

```
[5]  import pandas as pd
```

```
[6]  data = pd.read_csv('/content/Pokemon.csv')
```

```
[8]  data
```

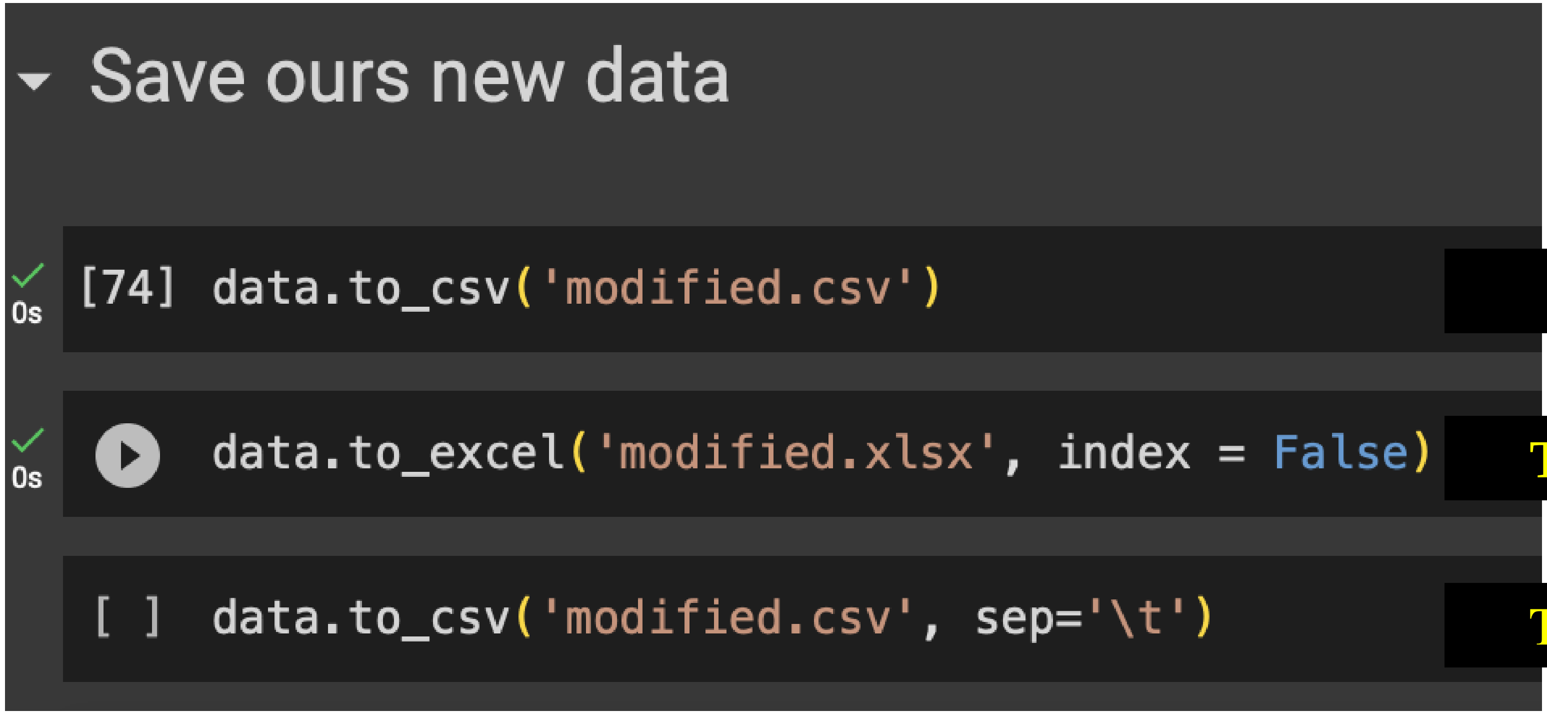
	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
799	721									90	70	6	True

800 rows × 13 colun

All rows? How about first and last rows?

To CSV

To CSV



Save ours new data

```
[74] data.to_csv('modified.csv')
```

```
data.to_excel('modified.xlsx', index = False)
```

```
[ ] data.to_csv('modified.csv', sep='\t')
```

To CSV format

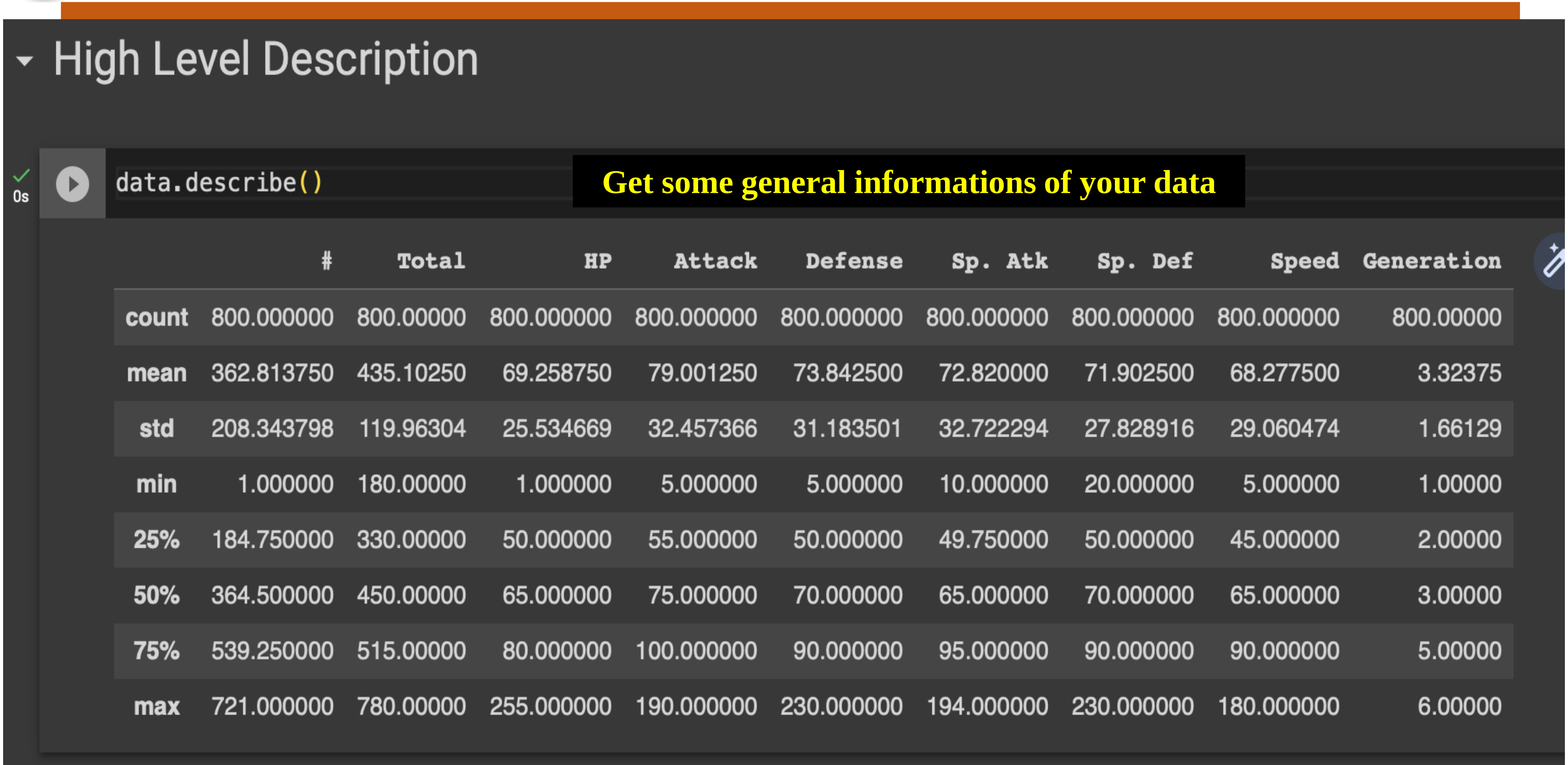
To xlsx format, remove index col

To CSV format, use ‘tab’ instead

Describe

AI VIET NAM
@aivietnam.edu.vn

Describe



High Level Description

```
data.describe()
```

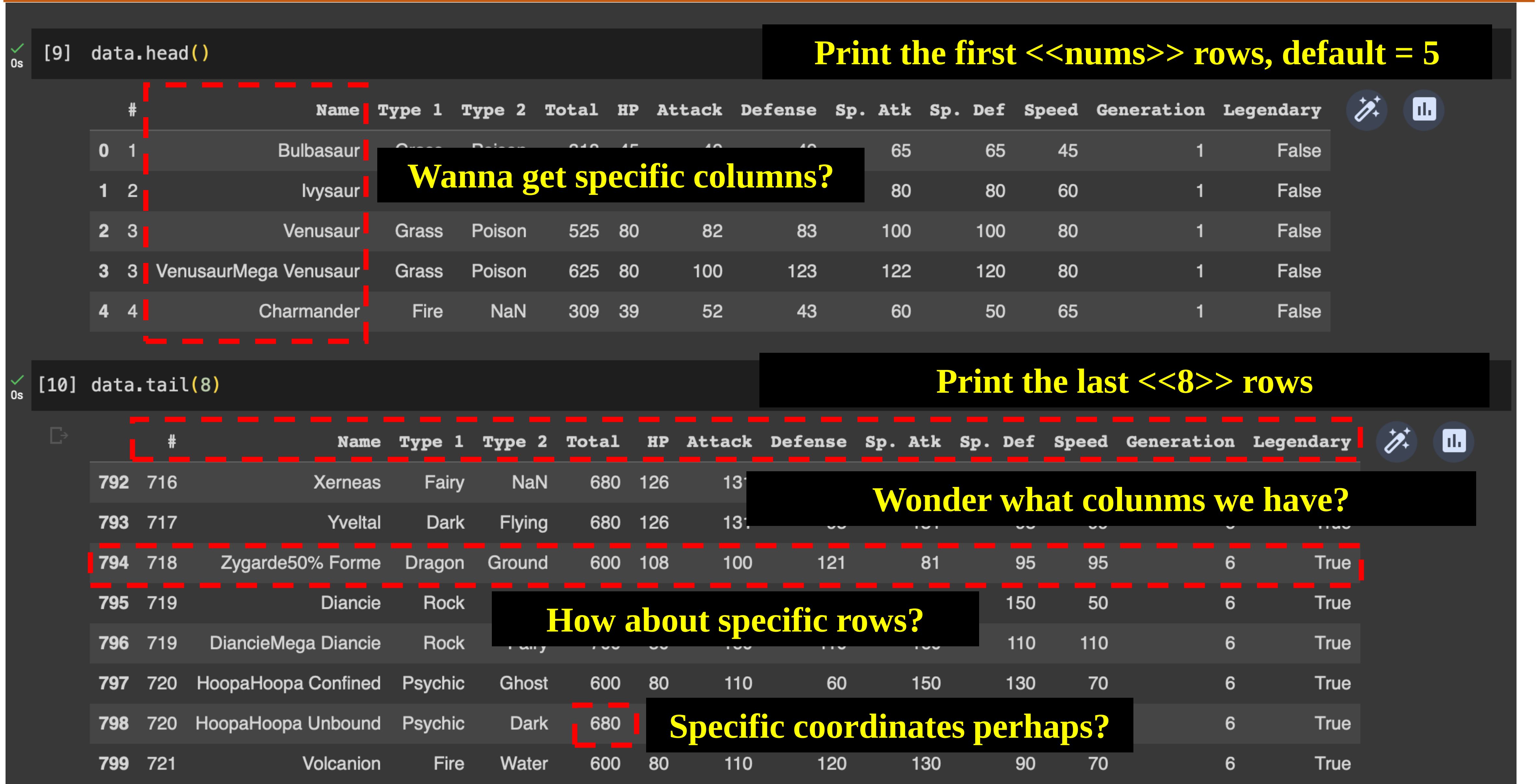
Get some general informations of your data

	#	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
count	800.000000	800.00000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.00000
mean	362.813750	435.10250	69.258750	79.001250	73.842500	72.820000	71.902500	68.277500	3.32375
std	208.343798	119.96304	25.534669	32.457366	31.183501	32.722294	27.828916	29.060474	1.66129
min	1.000000	180.00000	1.000000	5.000000	5.000000	10.000000	20.000000	5.000000	1.00000
25%	184.750000	330.00000	50.000000	55.000000	50.000000	49.750000	50.000000	45.000000	2.00000
50%	364.500000	450.00000	65.000000	75.000000	70.000000	65.000000	70.000000	65.000000	3.00000
75%	539.250000	515.00000	80.000000	100.000000	90.000000	95.000000	90.000000	90.000000	5.00000
max	721.000000	780.00000	255.000000	190.000000	230.000000	194.000000	230.000000	180.000000	6.00000

Head - Tail

AI VIET NAM
@aivietnam.edu.vn

Head - Tail



```
[9] data.head()
```

Print the first <<nums>> rows, default = 5

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

Wanna get specific columns?

```
[10] data.tail(8)
```

Print the last <<8>> rows

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
792	716	Xerneas	Fairy	NaN	680	126	131	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
793	717	Yveltal	Dark	Flying	680	126	131	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
794	718	Zygarde50% Forme	Dragon	Ground	600	108	100	121	81	95	95	6	True
795	719	Diancie	Rock	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	150	50	6	True
796	719	DiancieMega Diancie	Rock	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	680	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	6	True
799	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

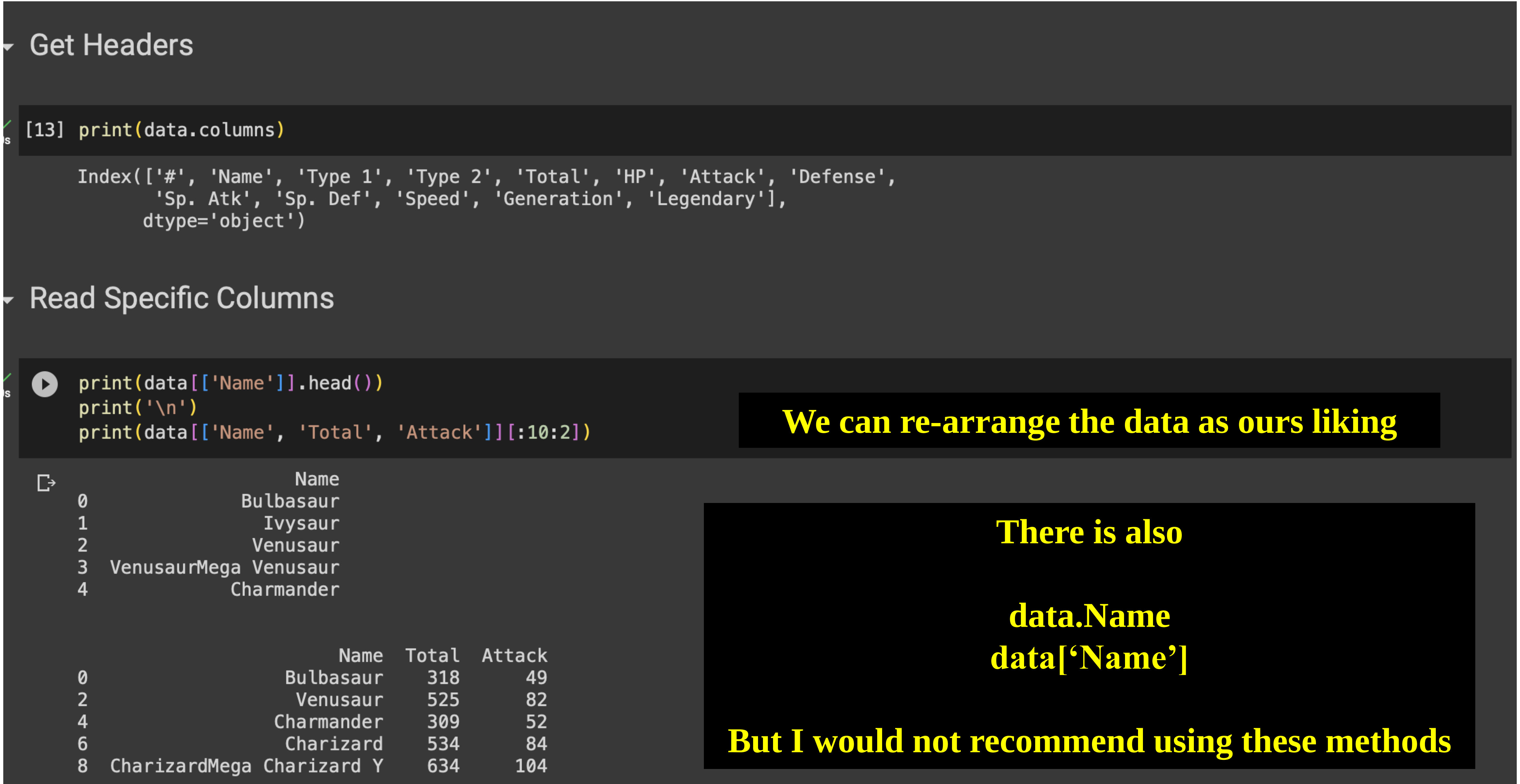
Wonder what columms we have?

How about specific rows?

Specific coordinates perhaps?

Columns

Columns



Get Headers

```
[13] print(data.columns)
```

```
Index(['#', 'Name', 'Type 1', 'Type 2', 'Total', 'HP', 'Attack', 'Defense',
       'Sp. Atk', 'Sp. Def', 'Speed', 'Generation', 'Legendary'],
      dtype='object')
```

Read Specific Columns

```
print(data[['Name']].head())
print('\n')
print(data[['Name', 'Total', 'Attack']][:10:2])
```

```
                     Name
0               Bulbasaur
1                 Ivysaur
2                Venusaur
3   VenusaurMega Venusaur
4              Charmander


                        Name  Total  Attack
0                  Bulbasaur    318      49
2                   Venusaur    525      82
4                 Charmander    309      52
6                  Charizard    534      84
8  CharizardMega Charizard Y    634     104
```

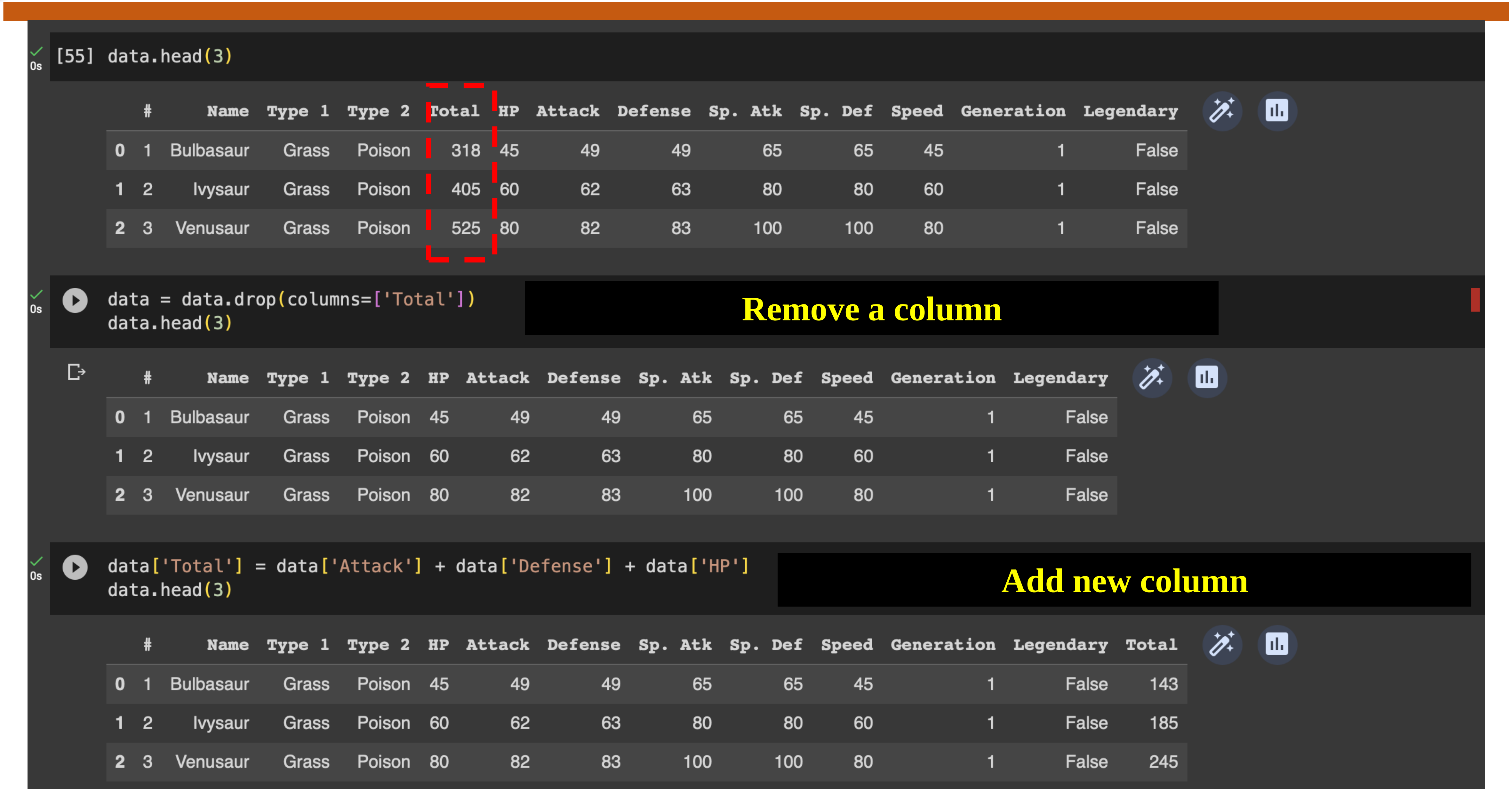
We can re-arrange the data as ours liking

There is also

data.Name
data[‘Name’]

But I would not recommend using these methods

Columns



```
[55] data.head(3)
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False

```
data = data.drop(columns=['Total'])
data.head(3)
```

Remove a column

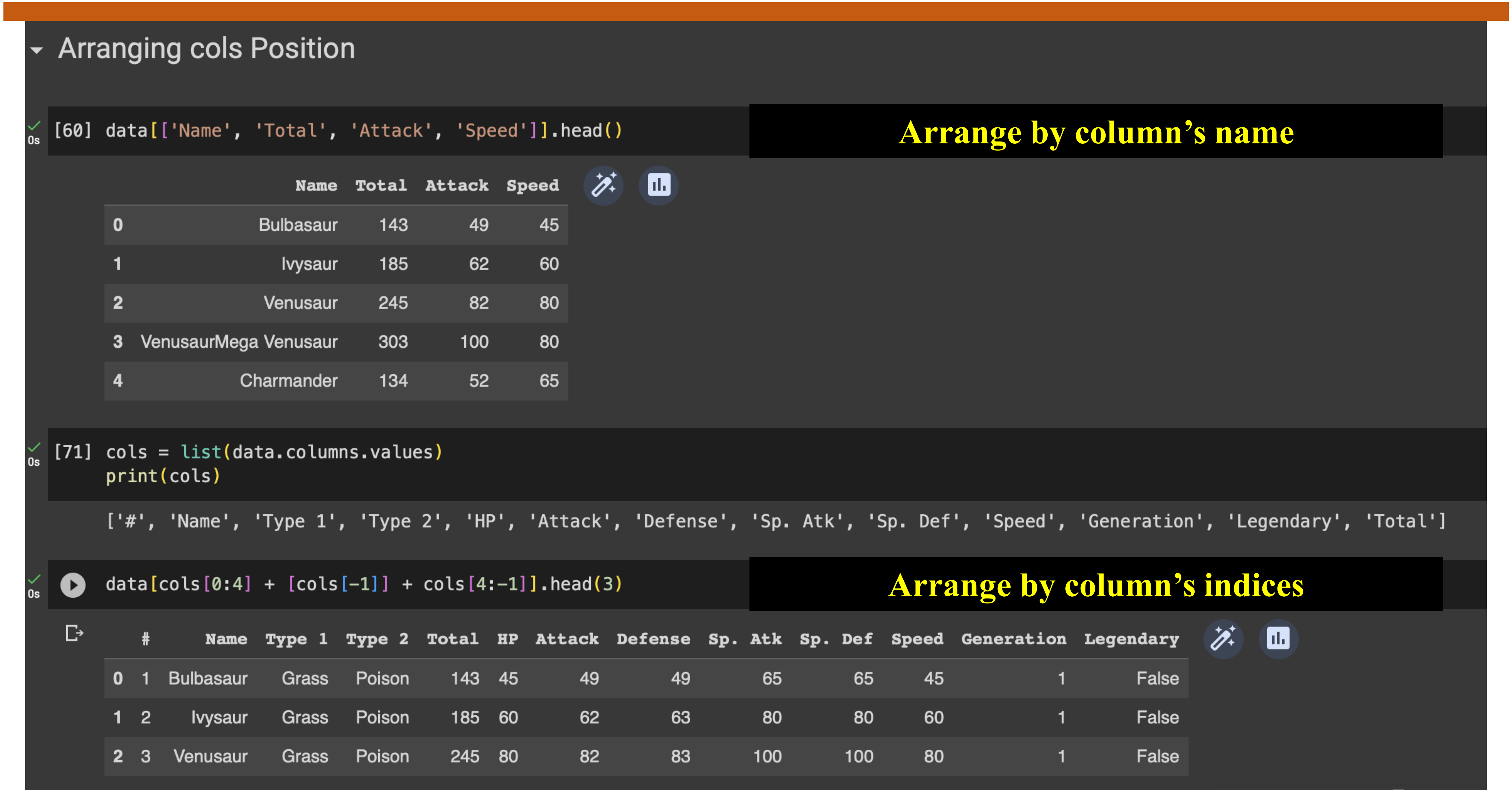
	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False

```
data['Total'] = data['Attack'] + data['Defense'] + data['HP']
data.head(3)
```

Add new column

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False	143
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False	185
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False	245

Columns



Arranging cols Position

```
[60] data[['Name', 'Total', 'Attack', 'Speed']].head()
```

Arrange by column’s name

	Name	Total	Attack	Speed
0	Bulbasaur	143	49	45
1	Ivysaur	185	62	60
2	Venusaur	245	82	80
3	VenusaurMega Venusaur	303	100	80
4	Charmander	134	52	65

```
[71] cols = list(data.columns.values)
     print(cols)
```

```
['#', 'Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation', 'Legendary', 'Total']
```

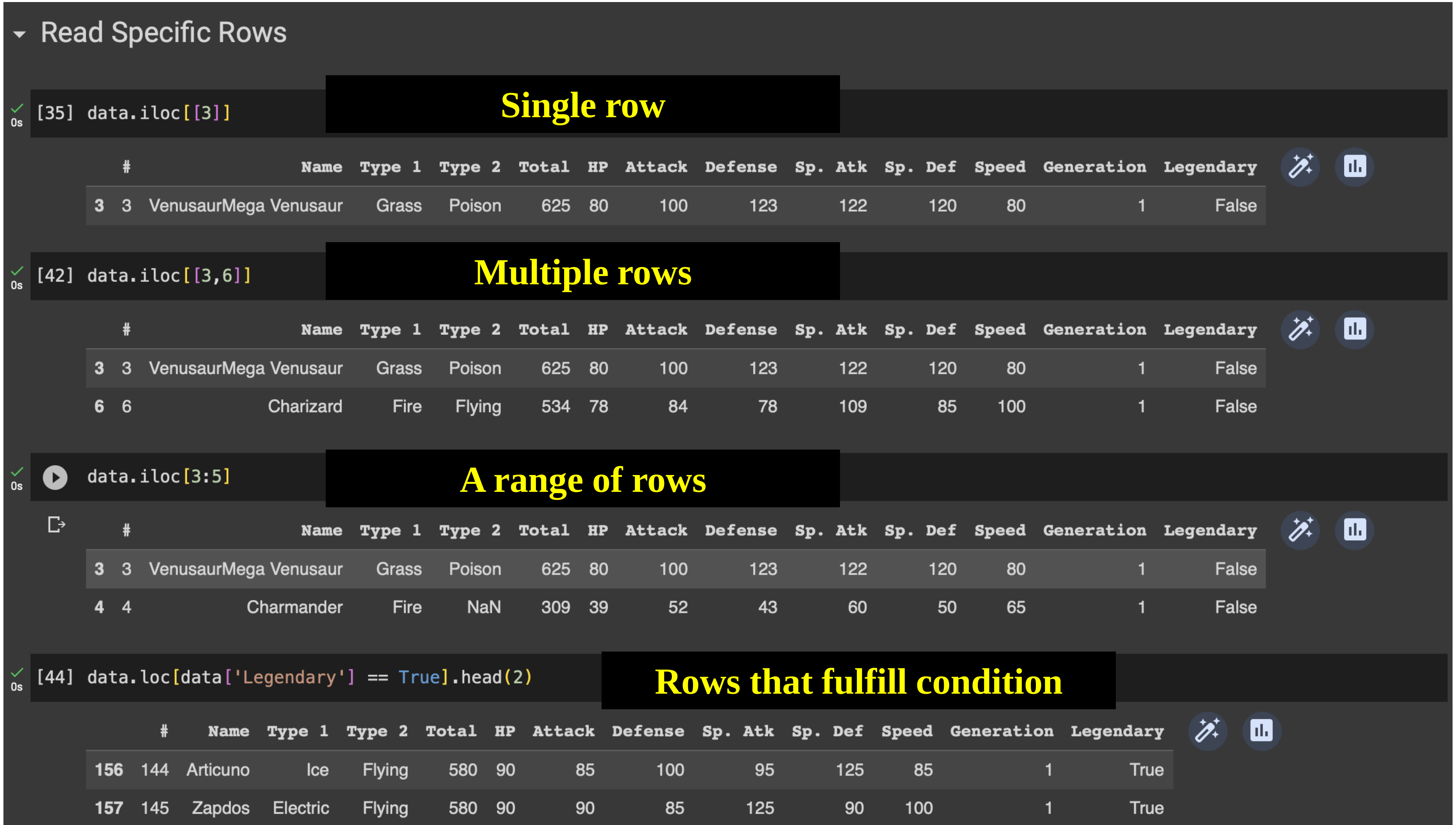
```
data[cols[0:4] + [cols[-1]] + cols[4:-1]].head(3)
```

Arrange by column’s indices

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	143	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	185	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	245	80	82	83	100	100	80	1	False

Iloc - Loc

iloc - loc



Read Specific Rows

Single row

```
[35] data.iloc[[3]]
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False

Multiple rows

```
[42] data.iloc[[3,6]]
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False

A range of rows

```
data.iloc[3:5]
```

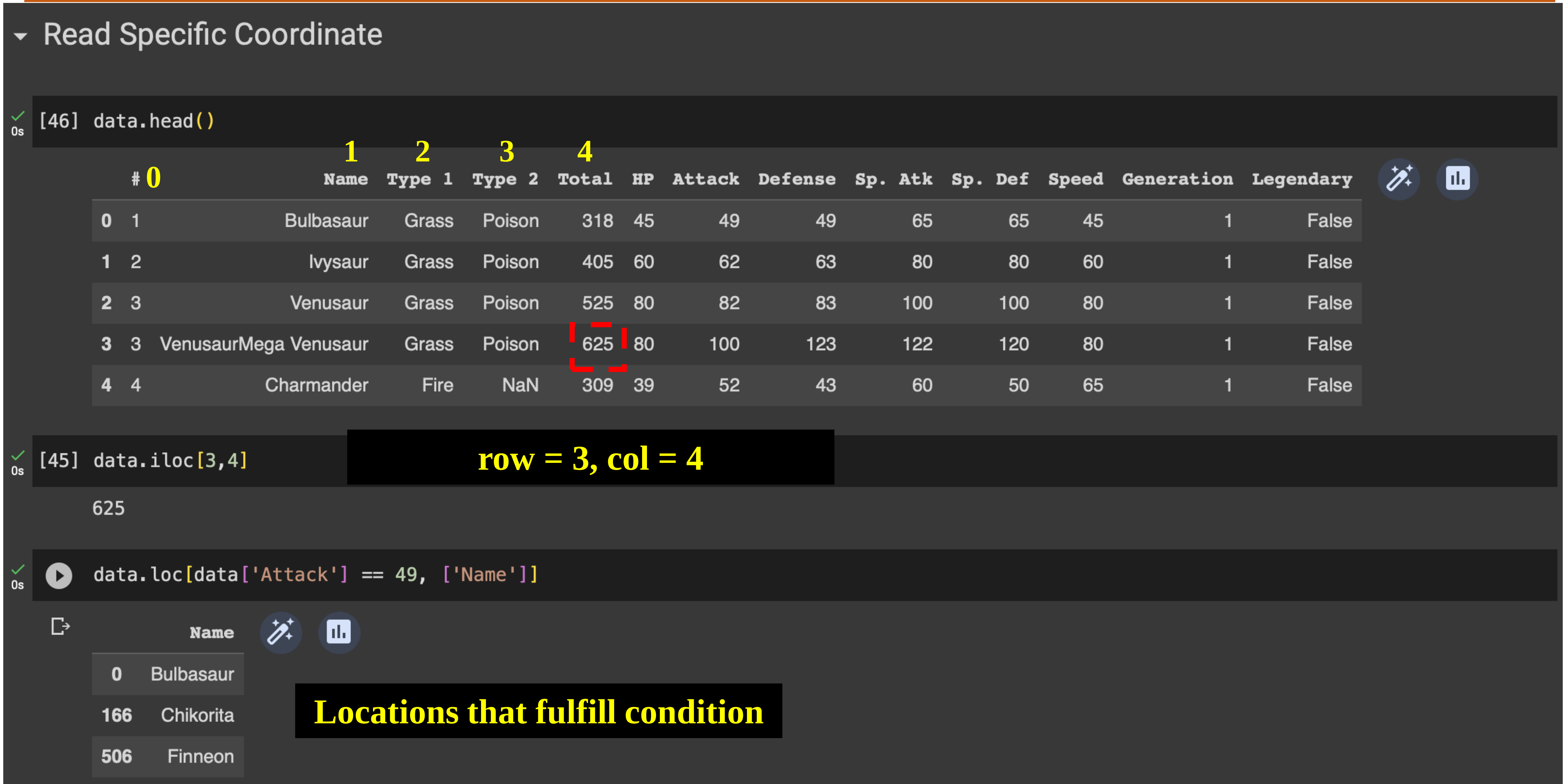
	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

Rows that fulfill condition

```
[44] data.loc[data['Legendary'] == True].head(2)
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
156	144	Articuno	Ice	Flying	580	90	85	100	95	125	85	1	True
157	145	Zapdos	Electric	Flying	580	90	90	85	125	90	100	1	True

iloc - loc



Read Specific Coordinate

```
[46] data.head()
```

	0	1	2	3	4								
	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

```
[45] data.iloc[3,4]
```

row = 3, col = 4

```
625
```

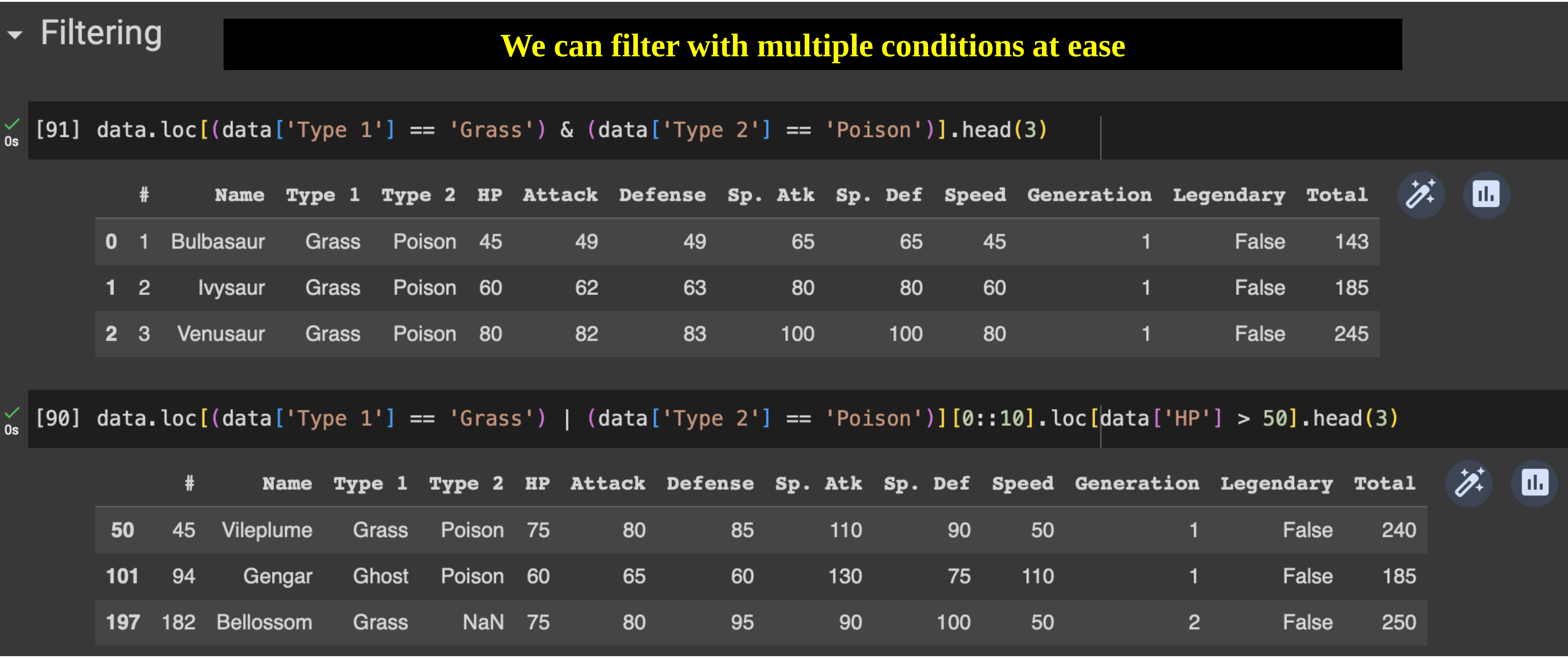
```
data.loc[data['Attack'] == 49, ['Name']]
```

	Name
0	Bulbasaur
166	Chikorita
506	Finneon

Locations that fulfill condition

Filtering Data

Filtering Data



Filtering

We can filter with multiple conditions at ease

```
[91] data.loc[(data['Type 1'] == 'Grass') & (data['Type 2'] == 'Poison')].head(3)
```

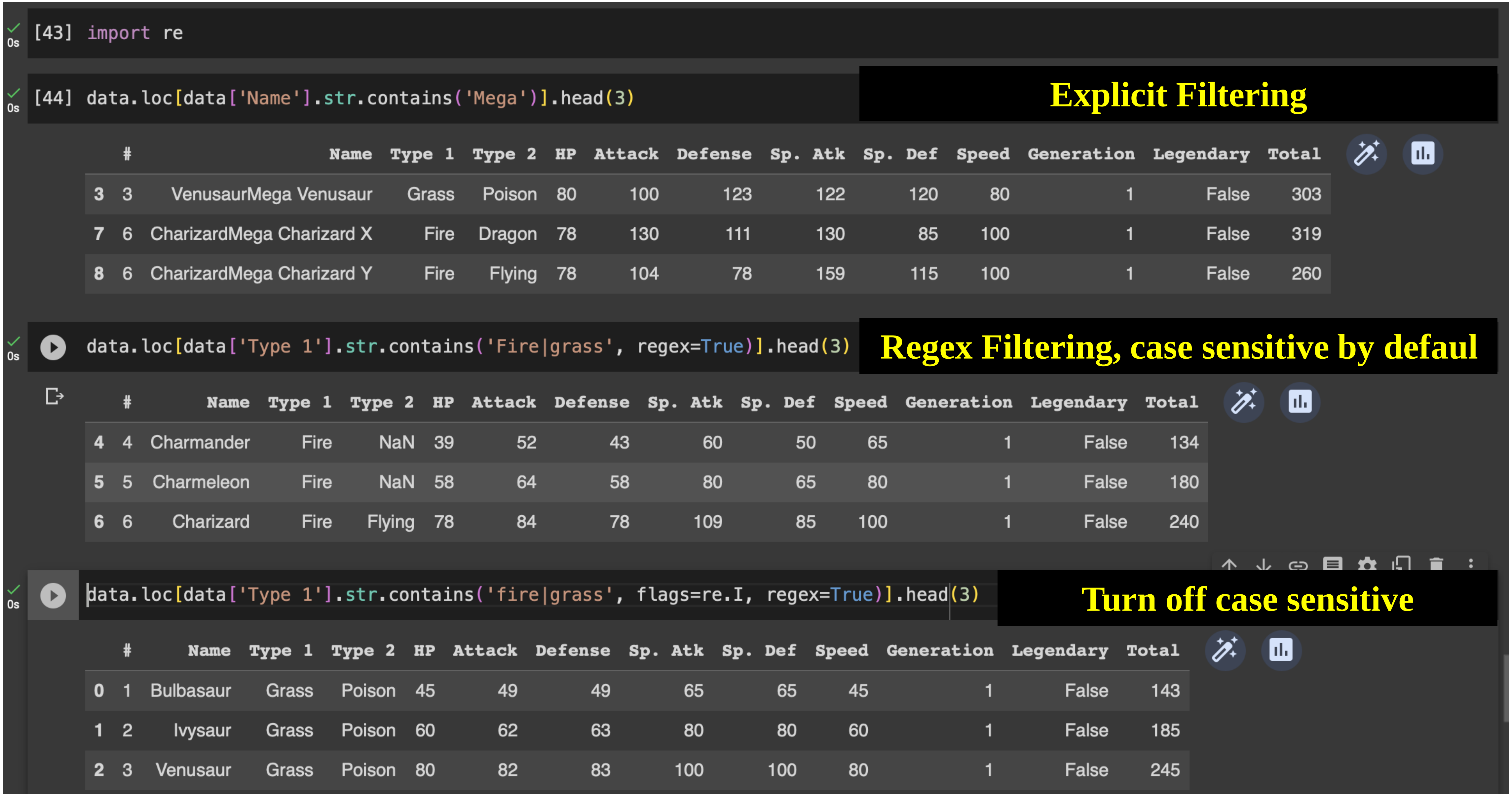
	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False	143
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False	185
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False	245

```
[90] data.loc[(data['Type 1'] == 'Grass') | (data['Type 2'] == 'Poison')][0::10].loc[data['HP'] > 50].head(3)
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
50	45	Vileplume	Grass	Poison	75	80	85	110	90	50	1	False	240
101	94	Gengar	Ghost	Poison	60	65	60	130	75	110	1	False	185
197	182	Bellossom	Grass	NaN	75	80	95	90	100	50	2	False	250

Regex Filtering

Regex Filtering



```
[43] import re
```

```
[44] data.loc[data['Name'].str.contains('Mega')].head(3)
```

Explicit Filtering

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False	303
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	False	319
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	False	260

```
data.loc[data['Type 1'].str.contains('Fire|grass', regex=True)].head(3)
```

Regex Filtering, case sensitive by defaul

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	False	134
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	1	False	180
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	1	False	240

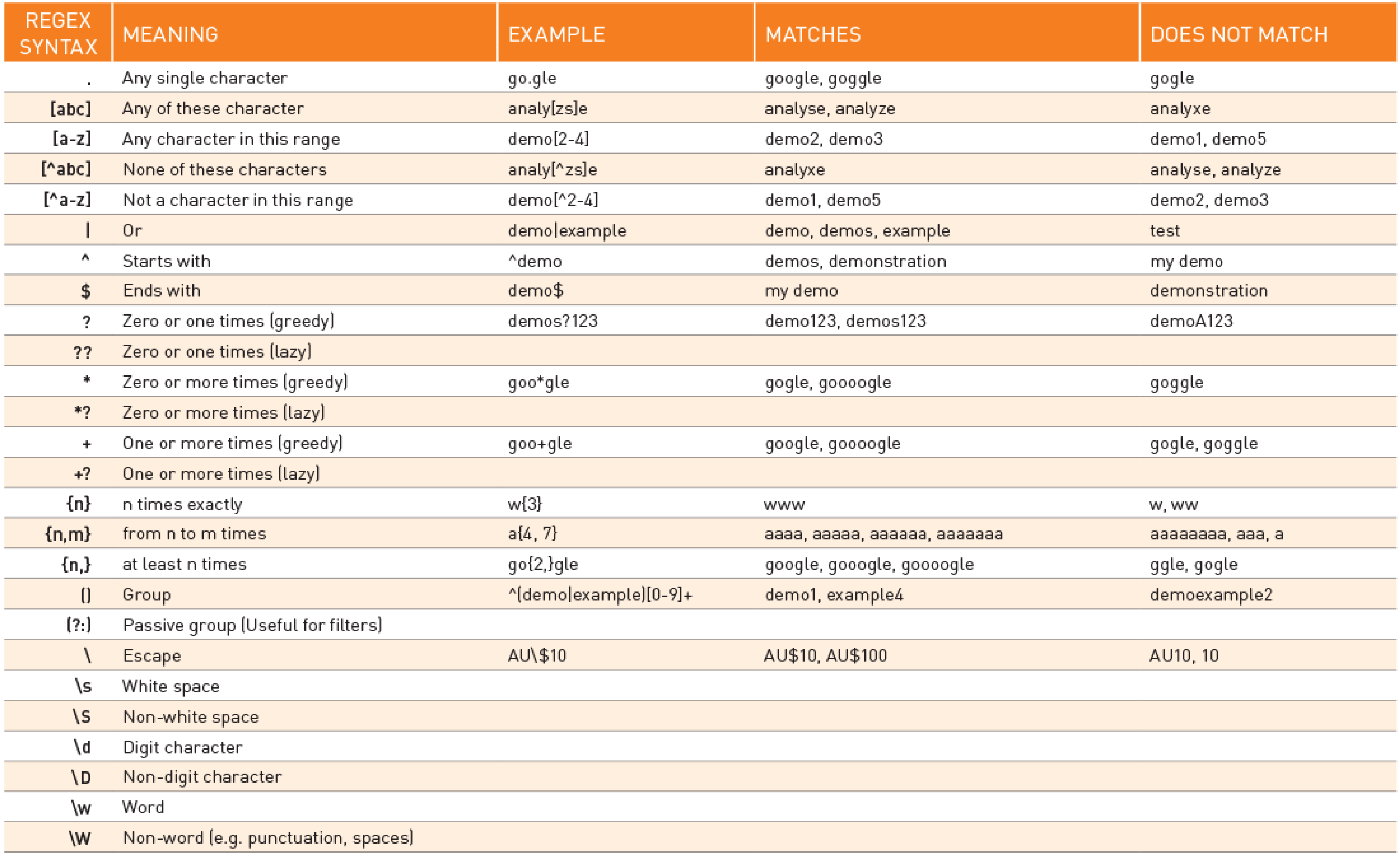
```
data.loc[data['Type 1'].str.contains('fire|grass', flags=re.I, regex=True)].head(3)
```

Turn off case sensitive

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False	143
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False	185
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False	245

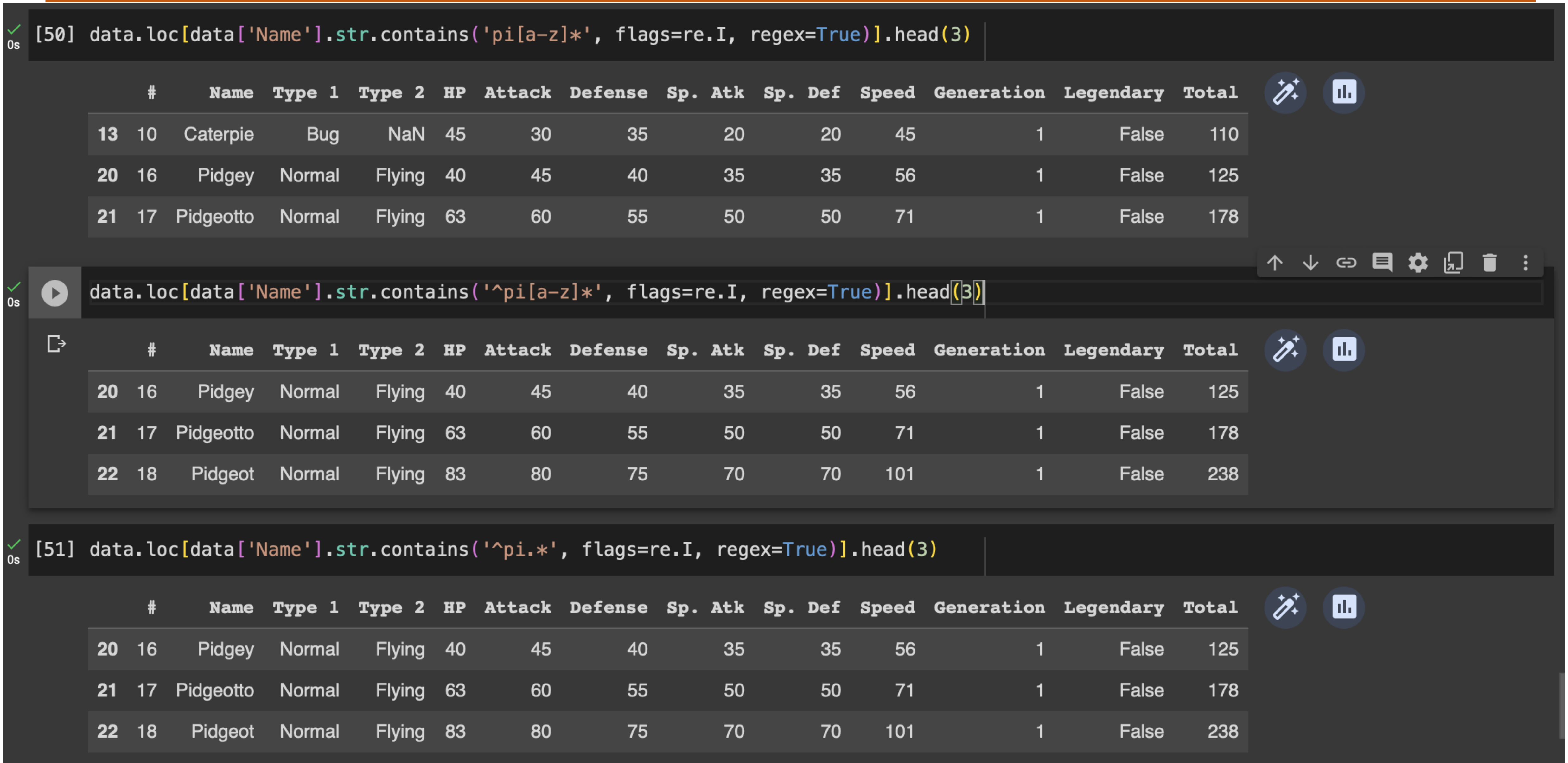
Regex Filtering

REGEX basic syntaxs



REGEX SYNTAX	MEANING	EXAMPLE	MATCHES	DOES NOT MATCH
.	Any single character	go.gle	google, goggle	gogle
[abc]	Any of these character	analy[zs]e	analyse, analyze	analyxe
[a-z]	Any character in this range	demo[2-4]	demo2, demo3	demo1, demo5
[^abc]	None of these characters	analy[^zs]e	analyxe	analyse, analyze
[^a-z]	Not a character in this range	demo[^2-4]	demo1, demo5	demo2, demo3
|	Or	demo|example	demo, demos, example	test
^	Starts with	^demo	demos, demonstration	my demo
$	Ends with	demo$	my demo	demonstration
?	Zero or one times (greedy)	demos?123	demo123, demos123	demoA123
??	Zero or one times (lazy)			
*	Zero or more times (greedy)	goo*gle	gogle, goooogle	goggle
*?	Zero or more times (lazy)			
+	One or more times (greedy)	goo+gle	google, goooogle	gogle, goggle
+?	One or more times (lazy)			
{n}	n times exactly	w{3}	www	w, ww
{n,m}	from n to m times	a{4, 7}	aaaa, aaaaa, aaaaaa, aaaaaaa	aaaaaaaa, aaa, a
{n,}	at least n times	go{2,}gle	google, gooogle, goooogle	ggle, gogle
()	Group	^(demo|example)[0-9]+	demo1, example4	demoexample2
(?:)	Passive group (Useful for filters)			
\	Escape	AU\$10	AU$10, AU$100	AU10, 10
\s	White space			
\S	Non-white space			
\d	Digit character			
\D	Non-digit character			
\w	Word			
\W	Non-word (e.g. punctuation, spaces)			

Regex Filtering



```
[50] data.loc[data['Name'].str.contains('pi[a-z]*', flags=re.I, regex=True)].head(3)
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
13	10	Caterpie	Bug	NaN	45	30	35	20	20	45	1	False	110
20	16	Pidgey	Normal	Flying	40	45	40	35	35	56	1	False	125
21	17	Pidgeotto	Normal	Flying	63	60	55	50	50	71	1	False	178

```
data.loc[data['Name'].str.contains('^pi[a-z]*', flags=re.I, regex=True)].head(3)
```

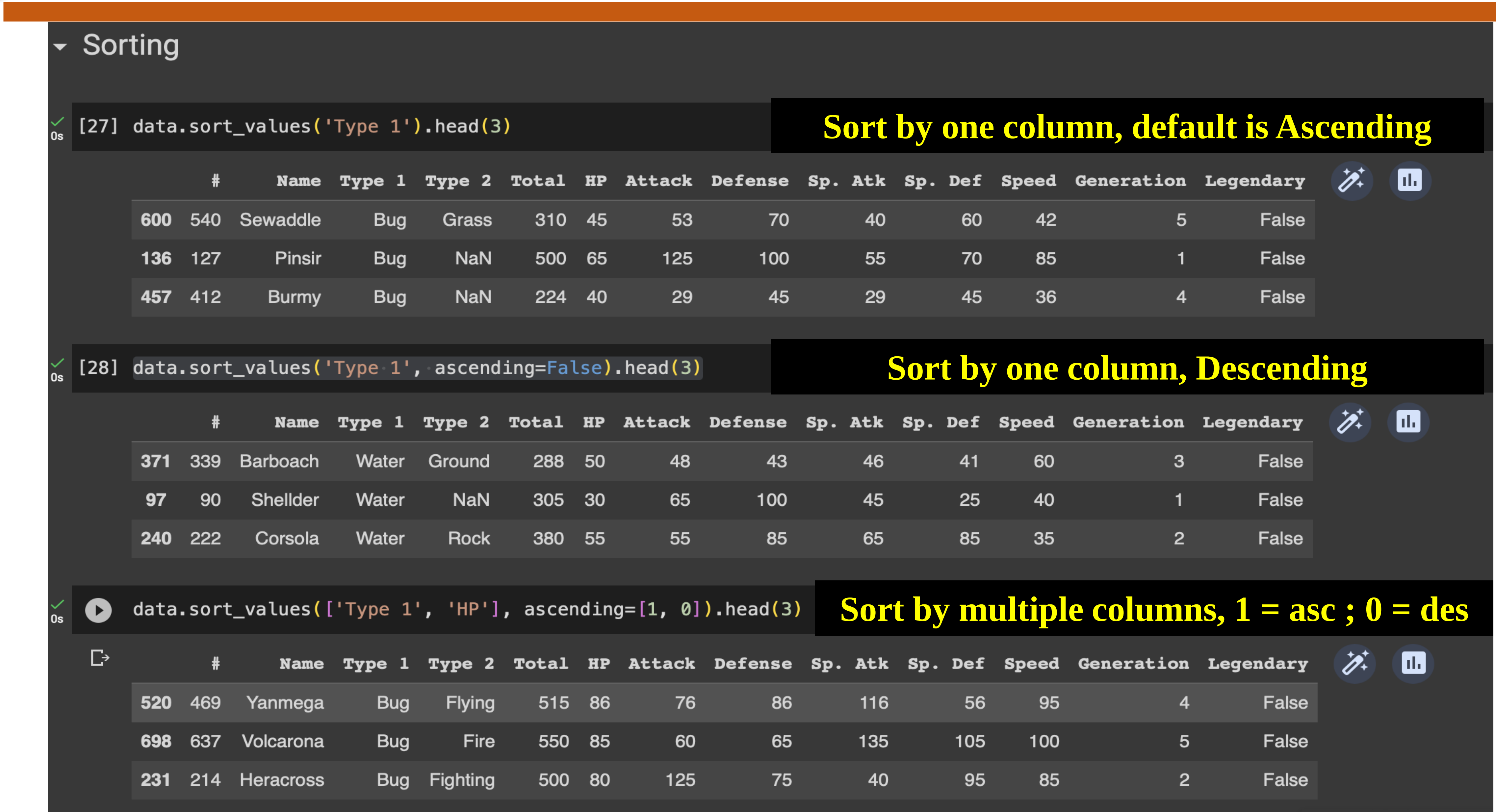
	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
20	16	Pidgey	Normal	Flying	40	45	40	35	35	56	1	False	125
21	17	Pidgeotto	Normal	Flying	63	60	55	50	50	71	1	False	178
22	18	Pidgeot	Normal	Flying	83	80	75	70	70	101	1	False	238

```
[51] data.loc[data['Name'].str.contains('^pi.*', flags=re.I, regex=True)].head(3)
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
20	16	Pidgey	Normal	Flying	40	45	40	35	35	56	1	False	125
21	17	Pidgeotto	Normal	Flying	63	60	55	50	50	71	1	False	178
22	18	Pidgeot	Normal	Flying	83	80	75	70	70	101	1	False	238

Sort values

Sort values



Sorting

```
data.sort_values('Type 1').head(3)
```

Sort by one column, default is Ascending

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
600	540	Sewaddle	Bug	Grass	310	45	53	70	40	60	42	5	False
136	127	Pinsir	Bug	NaN	500	65	125	100	55	70	85	1	False
457	412	Burmy	Bug	NaN	224	40	29	45	29	45	36	4	False

```
data.sort_values('Type 1', ascending=False).head(3)
```

Sort by one column, Descending

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
371	339	Barboach	Water	Ground	288	50	48	43	46	41	60	3	False
97	90	Shellder	Water	NaN	305	30	65	100	45	25	40	1	False
240	222	Corsola	Water	Rock	380	55	55	85	65	85	35	2	False

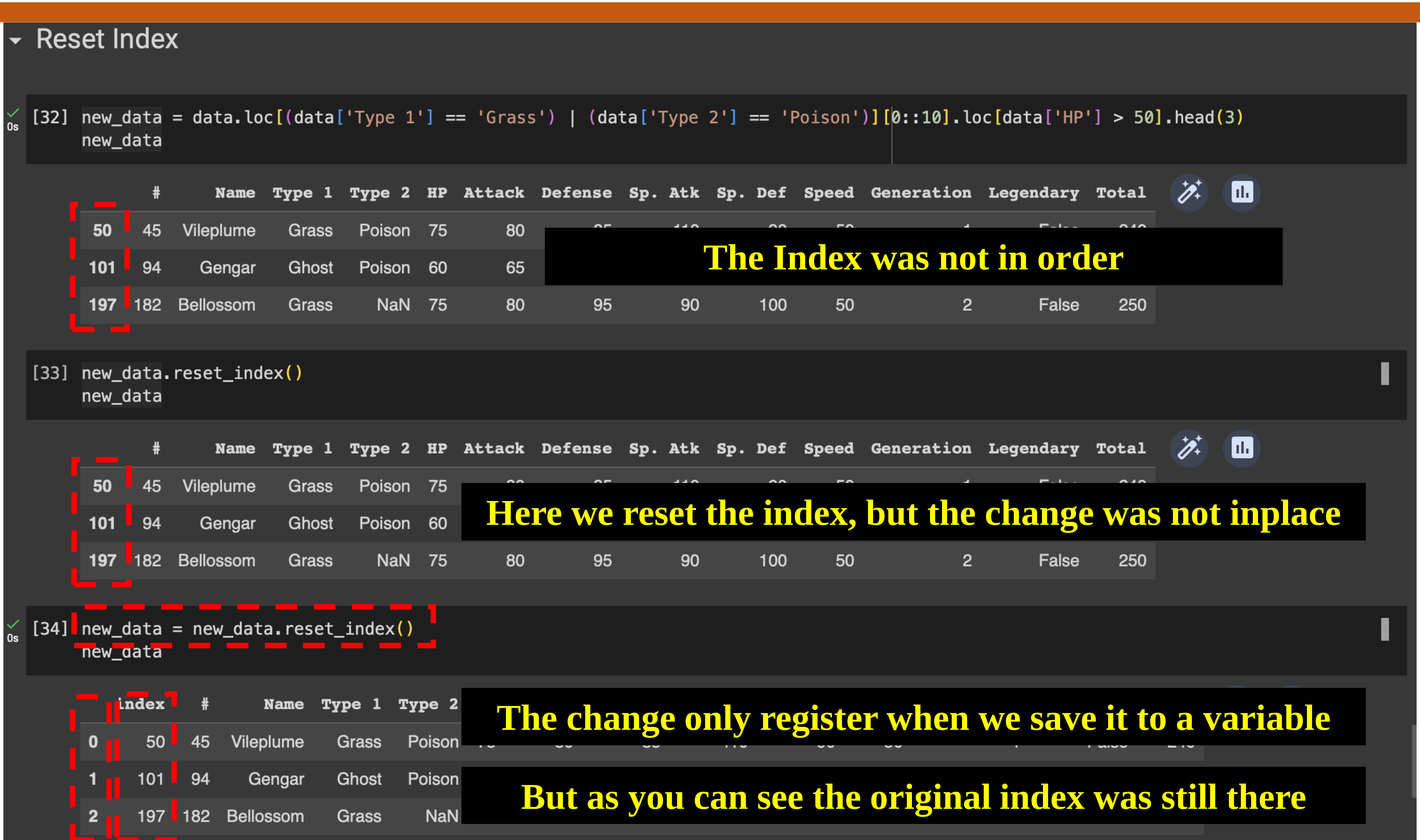
```
data.sort_values(['Type 1', 'HP'], ascending=[1, 0]).head(3)
```

Sort by multiple columns, 1 = asc ; 0 = des

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
520	469	Yanmega	Bug	Flying	515	86	76	86	116	56	95	4	False
698	637	Volcarona	Bug	Fire	550	85	60	65	135	105	100	5	False
231	214	Heracross	Bug	Fighting	500	80	125	75	40	95	85	2	False

Reset Index

Reset Index



Reset Index

```
[32] new_data = data.loc[(data['Type 1'] == 'Grass') | (data['Type 2'] == 'Poison')][0::10].loc[data['HP'] > 50].head(3)
     new_data
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
50	45	Vileplume	Grass	Poison	75	80	85	110	90	50	1	False	240
101	94	Gengar	Ghost	Poison	60	65	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
197	182	Bellossom	Grass	NaN	75	80	95	90	100	50	2	False	250

The Index was not in order

```
[33] new_data.reset_index()
     new_data
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
50	45	Vileplume	Grass	Poison	75	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
101	94	Gengar	Ghost	Poison	60	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
197	182	Bellossom	Grass	NaN	75	80	95	90	100	50	2	False	250

Here we reset the index, but the change was not inplace

```
[34] new_data = new_data.reset_index()
     new_data
```

	index	#	Name	Type 1	Type 2
0	50	45	Vileplume	Grass	Poison
1	101	94	Gengar	Ghost	Poison
2	197	182	Bellossom	Grass	NaN

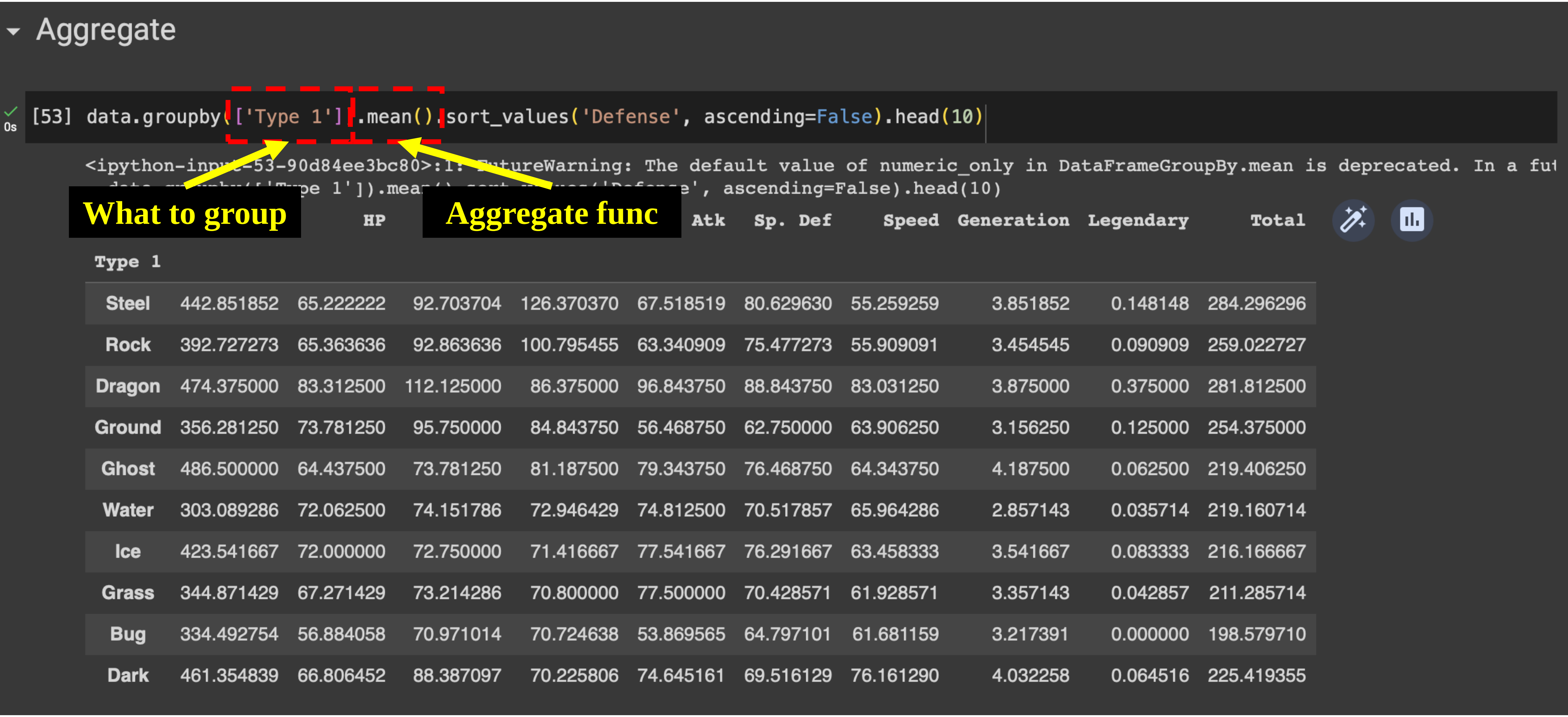
The change only register when we save it to a variable

But as you can see the original index was still there

Group by

AI VIET NAM
@aivietnam.edu.vn

Group by



Aggregate

```
[53] data.groupby(['Type 1']).mean().sort_values('Defense', ascending=False).head(10)
```

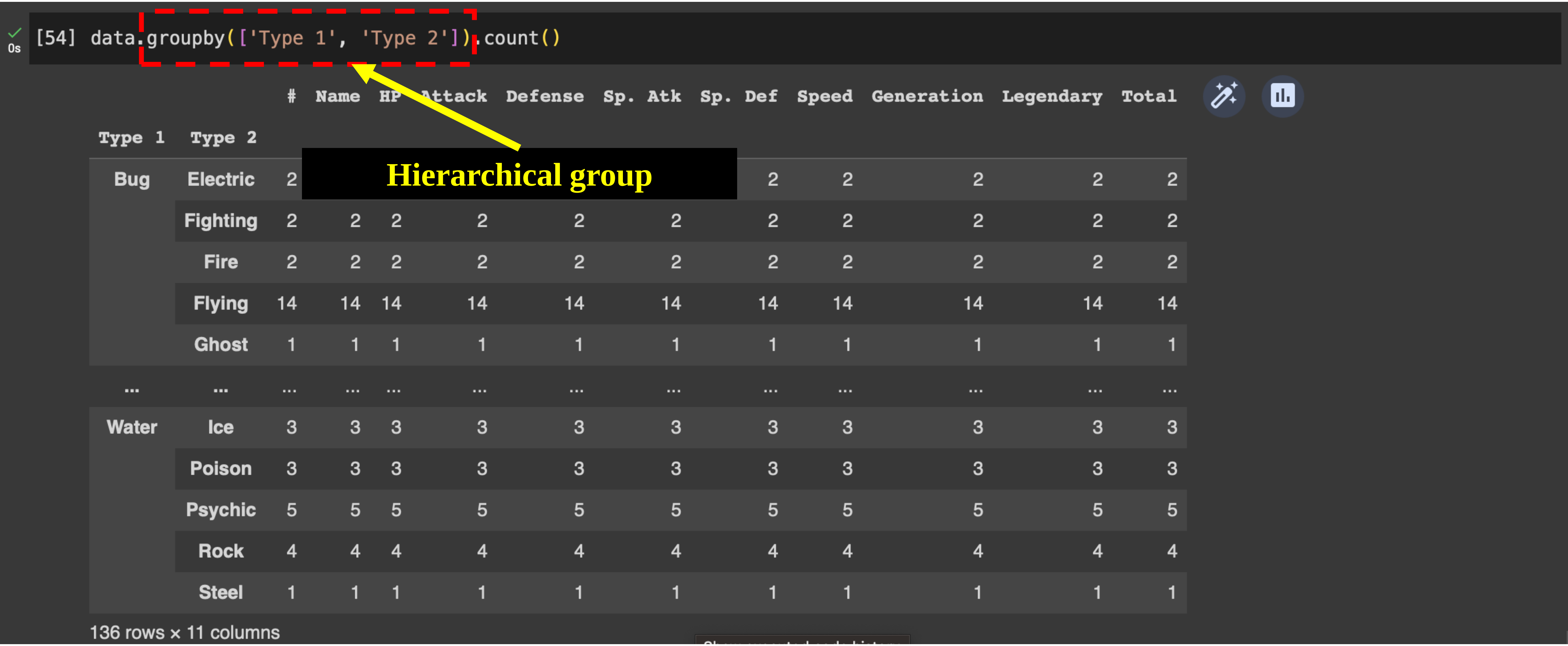
```
<ipython-input-53-90d84ee3bc80>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a fut
  data.groupby(['Type 1']).mean().sort_values('Defense', ascending=False).head(10)
```

What to group

Aggregate func

		HP			Atk	Sp. Def	Speed	Generation	Legendary	Total
Type 1										
Steel	442.851852	65.222222	92.703704	126.370370	67.518519	80.629630	55.259259	3.851852	0.148148	284.296296
Rock	392.727273	65.363636	92.863636	100.795455	63.340909	75.477273	55.909091	3.454545	0.090909	259.022727
Dragon	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250	3.875000	0.375000	281.812500
Ground	356.281250	73.781250	95.750000	84.843750	56.468750	62.750000	63.906250	3.156250	0.125000	254.375000
Ghost	486.500000	64.437500	73.781250	81.187500	79.343750	76.468750	64.343750	4.187500	0.062500	219.406250
Water	303.089286	72.062500	74.151786	72.946429	74.812500	70.517857	65.964286	2.857143	0.035714	219.160714
Ice	423.541667	72.000000	72.750000	71.416667	77.541667	76.291667	63.458333	3.541667	0.083333	216.166667
Grass	344.871429	67.271429	73.214286	70.800000	77.500000	70.428571	61.928571	3.357143	0.042857	211.285714
Bug	334.492754	56.884058	70.971014	70.724638	53.869565	64.797101	61.681159	3.217391	0.000000	198.579710
Dark	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290	4.032258	0.064516	225.419355

Group by

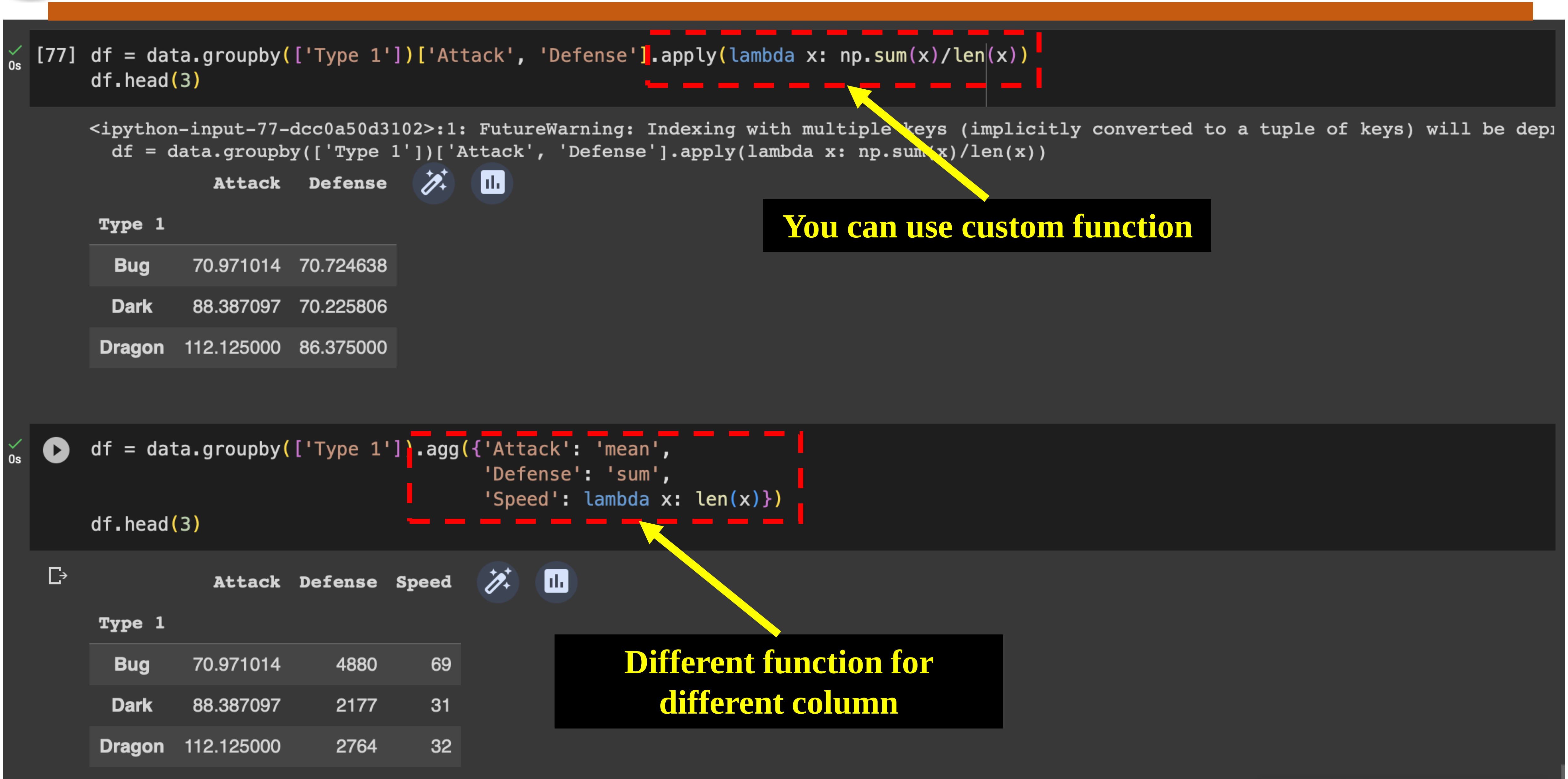


```
[54] data.groupby(['Type 1', 'Type 2']).count()
```

Hierarchical group

```
                     #  Name  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed  Generation  Legendary  Total
Type 1  Type 2
Bug     Electric     2                                          2      2         2          2        2
        Fighting     2     2   2       2        2        2      2      2         2          2        2
        Fire         2     2   2       2        2        2      2      2         2          2        2
        Flying      14    14  14      14       14       14     14     14        14         14       14
        Ghost        1     1   1       1        1        1      1      1         1          1        1
...     ...        ...   ... ...     ...      ...      ...    ...    ...       ...        ...      ...
Water   Ice          3     3   3       3        3        3      3      3         3          3        3
        Poison       3     3   3       3        3        3      3      3         3          3        3
        Psychic      5     5   5       5        5        5      5      5         5          5        5
        Rock         4     4   4       4        4        4      4      4         4          4        4
        Steel        1     1   1       1        1        1      1      1         1          1        1
136 rows × 11 columns
```

Group by



```
[77] df = data.groupby(['Type 1'])['Attack', 'Defense'].apply(lambda x: np.sum(x)/len(x))
     df.head(3)
```

```
<ipython-input-77-dcc0a50d3102>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be dep
  df = data.groupby(['Type 1'])['Attack', 'Defense'].apply(lambda x: np.sum(x)/len(x))
```

	Attack	Defense
Type 1		
Bug	70.971014	70.724638
Dark	88.387097	70.225806
Dragon	112.125000	86.375000

You can use custom function

```
df = data.groupby(['Type 1']).agg({'Attack': 'mean',
                                   'Defense': 'sum',
                                   'Speed': lambda x: len(x)})

df.head(3)
```

	Attack	Defense	Speed
Type 1			
Bug	70.971014	4880	69
Dark	88.387097	2177	31
Dragon	112.125000	2764	32

Different function for different column

Quizzes

AI VIET NAM
@aivietnam.edu.vn

QUIZZES

Use what we have learned about Pandas to solve these questions below.

1/ How can you read a CSV file named "pokemon_data.csv" into a Pandas DataFrame?
2/ How do you display the first 5 rows of the DataFrame?
3/ How can you display the last 3 rows of the DataFrame?
4/ What method is used to see the names of all columns in the DataFrame?
5/ How can you select the rows from index 10 to 15 (inclusive)?
6/ How do you select the row with index 50?
7/ How can you filter the DataFrame to only show rows where the 'Type 1' column is 'Grass’?
8/ How can you filter the DataFrame to only show rows where the 'HP' column is greater than 100?
9/ How can you filter the DataFrame to only show rows where the 'Name' column contains the substring 'chu’?
10/ How can you sort the DataFrame based on the 'Attack' column in descending order?
11/ How do you reset the index of the DataFrame?
12/ How can you select specific columns 'Name', 'Type 1', and 'HP' from the DataFrame?
13/ Filter the DataFrame to only show rows where 'Type 1' is 'Grass' and 'Type 2' is 'Poison’?
14/ How can you find the mean HP of Legendary Pokémon?
15/ How can you create a new DataFrame containing only the rows with even-numbered indices?
16/ How can you write the filtered DataFrame to a new CSV file named "filtered_pokemon.csv"?

QUIZZES

Use what we have learned about Pandas to solve these questions below.

17/ Display the first 10 rows of the DataFrame for columns 'Name', 'Type 1', and 'Attack’?
18/ How do you find the maximum Defense value in the DataFrame?
19/ How can you create a new DataFrame with rows where 'Speed' is above the mean Speed value?
20/ How can you select the first 3 rows and the columns 'Name' and 'Attack’?
21/ How can you find the number of Pokémon of each 'Type 1' in the DataFrame?
22/ How can you display the rows where 'Generation' is 3 or 'Generation' is 5?
23/ How can you replace all occurrences of 'Fire' in the 'Type 1' column with 'Flame’?
24/ How can you display the rows where 'Name' starts with the letter 'P’?
25/ How can you find the mean 'Attack' value for each 'Type 1’?
26/ How can you sort the DataFrame based on 'Type 1' in ascending and 'Attack' in descending order?
27/ How can you find the median 'Defense' value for each 'Type 1’?
28/ How can you filter the DataFrame to show only Legendary Pokémon with 'Attack’ > 100?
29/ Create a new DataFrame with only 'Name' and 'Total' columns, and rename 'Total' to 'Overall’?
30/ How can you display the rows where 'Type 1' is 'Water' and 'Attack' is at least 80?
31/ Create a new DataFrame with only the 'Name' and 'Defense' columns for Generation 4 Pokémon?
32/ How can you find the mean 'Total' value for each 'Type 1' and 'Type 2' combination?
33/ Filter the DataFrame to show only Pokémon with a 'Total' between 400 and 500 (inclusive)?