

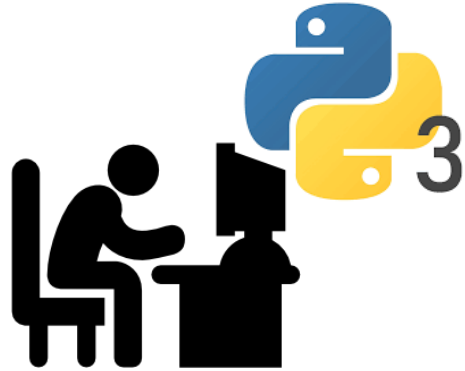


## **DATA ANÁLISIS IN PYTHON PANDAS & SEABORN**

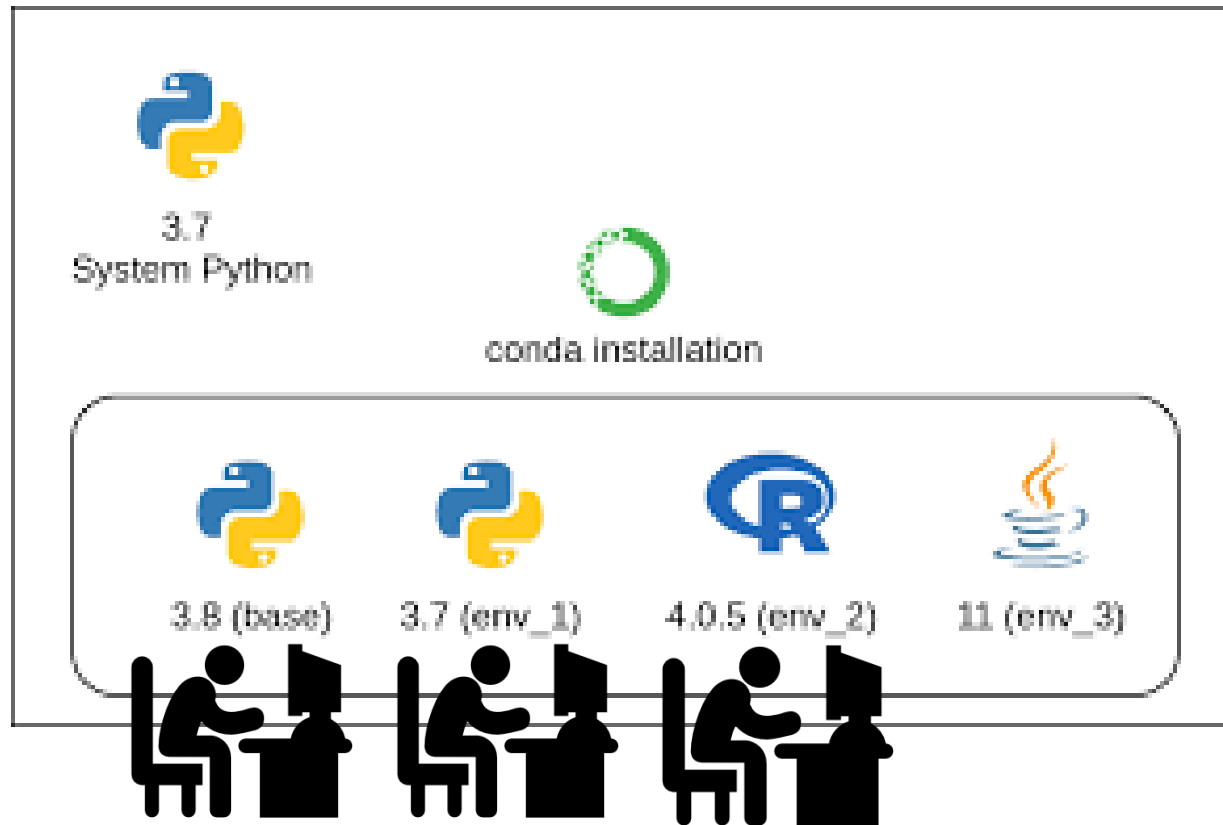
Virginia Domínguez García  
Seminarios Ecoinformática AEET  
2022



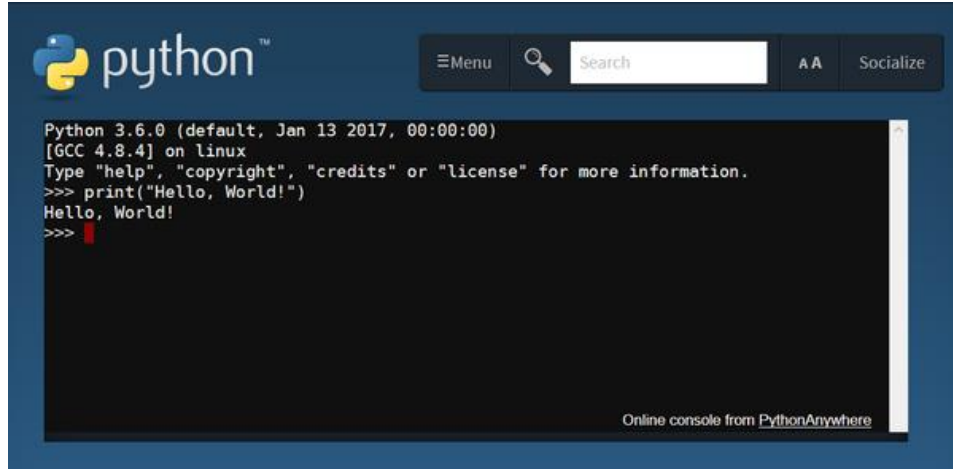
# 1 - ¿Por qué Python en Anaconda?



# 1 - ¿Por qué Python en Anaconda?



## 2 - ¿Cómo trabajar en Python?



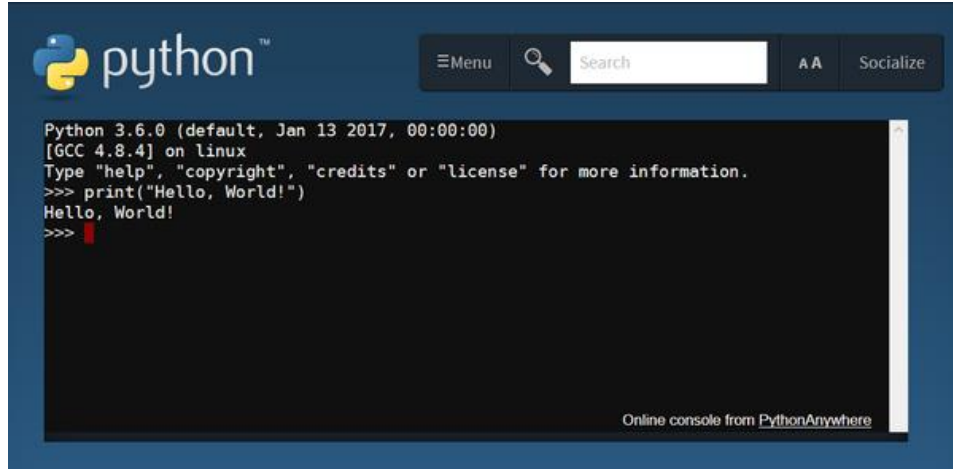
The image shows a screenshot of the PythonAnywhere online console. The interface has a dark blue header with the Python logo and the word "python" in white. To the right of the logo are buttons for "Menu", a search bar, "A A" (font size), and "Socialize". Below the header is a black terminal window with white text. The text in the terminal shows the Python 3.6.0 shell prompt, the system information "[GCC 4.8.4] on linux", and a user typing a print statement which has been executed successfully, outputting "Hello, World!". At the bottom right of the terminal window, there is a small text link: "Online console from [PythonAnywhere](#)".

```
python™
Menu Search A A Socialize

Python 3.6.0 (default, Jan 13 2017, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>>
```

Online console from [PythonAnywhere](#)

## 2 - ¿Cómo trabajar en Python?



The screenshot shows the PythonAnywhere online console interface. At the top, there's a header with the Python logo, the word "python", and navigation links for "Menu", "Search", "A A", and "Socialize". The main area is a terminal window for Python 3.6.0 (default, Jan 13 2017, 00:00:00) on Linux. It shows the prompt "Type 'help', 'copyright', 'credits' or 'license' for more information." followed by the execution of a simple program: `>>> print("Hello, World!")` which outputs "Hello, World!". The prompt `>>>` is shown again at the bottom. A small link "Online console from PythonAnywhere" is visible in the bottom right corner of the terminal area.

```
python
Menu Search A A Socialize

Python 3.6.0 (default, Jan 13 2017, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>>
```

Online console from [PythonAnywhere](#)

```
def parse_arguments():
    """Read arguments from a command line."""
    parser = argparse.ArgumentParser(description='Arguments get parsed via --commands')
    parser.add_argument('-v', metavar='verbosity', type=int, default=2,
                        help='Verbosity of logging: 0 -critical, 1 -warning, 2 -info, 3 -debug')

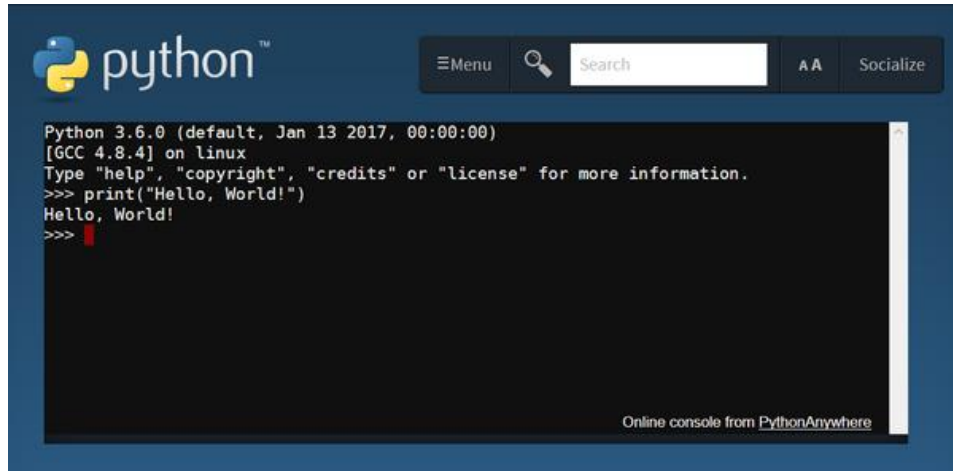
    args = parser.parse_args()
    verbose = {0: logging.CRITICAL, 1: logging.WARNING, 2: logging.INFO, 3: logging.DEBUG}
    logging.basicConfig(format='%(message)s', level=verbose[args.v], stream=sys.stdout)

    return args

def main():
    pass

if __name__ == '__main__':
    args = parse_arguments()
    main()
```

# 2 - ¿Cómo trabajar en Python?



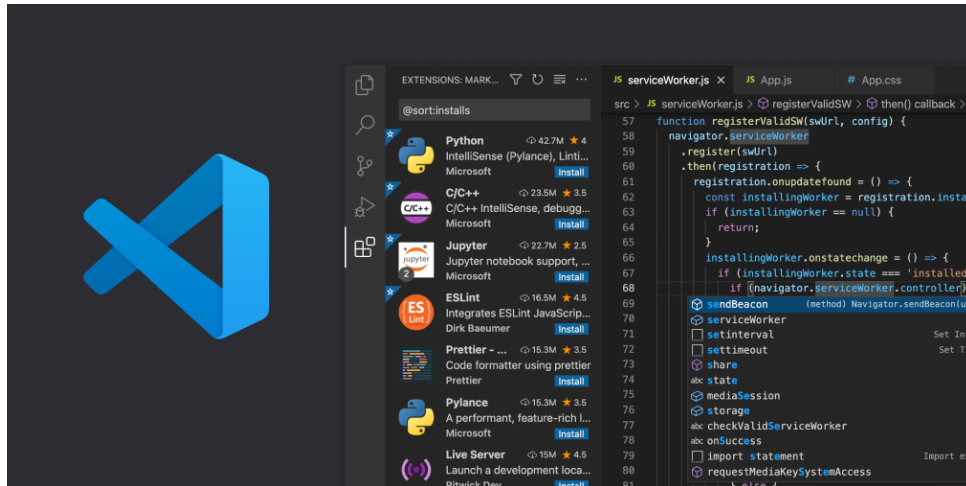
```
def parse_arguments():
    """Read arguments from a command line."""
    parser = argparse.ArgumentParser(description='Arguments get parsed via --commands')
    parser.add_argument('-v', metavar='verbosity', type=int, default=2,
                        help='Verbosity of logging: 0 -critical, 1 -warning, 2 -info, 3 -debug')

    args = parser.parse_args()
    verbose = {0: logging.CRITICAL, 1: logging.WARNING, 2: logging.INFO, 3: logging.DEBUG}
    logging.basicConfig(format='%(message)s', level=verbose[args.v], stream=sys.stdout)

    return args

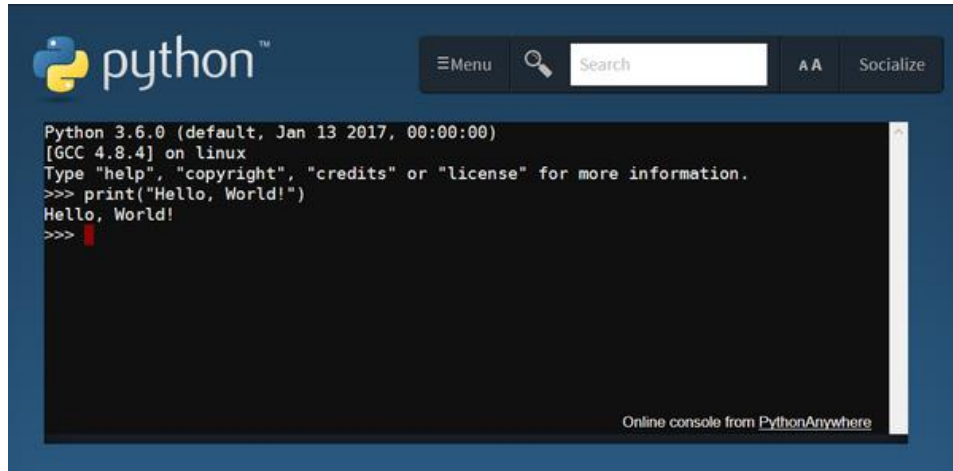
def main():
    pass

if __name__ == '__main__':
    args = parse_arguments()
    main()
```



<https://code.visualstudio.com/>

# 2 - ¿Cómo trabajar en Python?



The screenshot shows the Python Anywhere online console interface. At the top, there's a 'python' logo and a search bar. Below, it displays the Python version (3.6.0) and the GCC version (4.8.4) on Linux. The console shows a simple Python script being executed: `>>> print("Hello, World!")` followed by the output `Hello, World!`. A link to the 'Online console from PythonAnywhere' is visible at the bottom right.

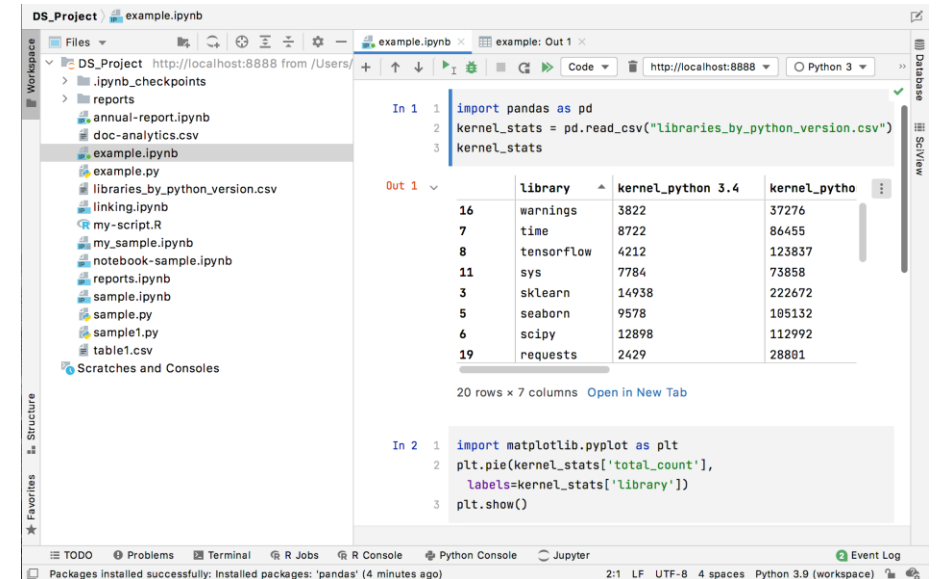
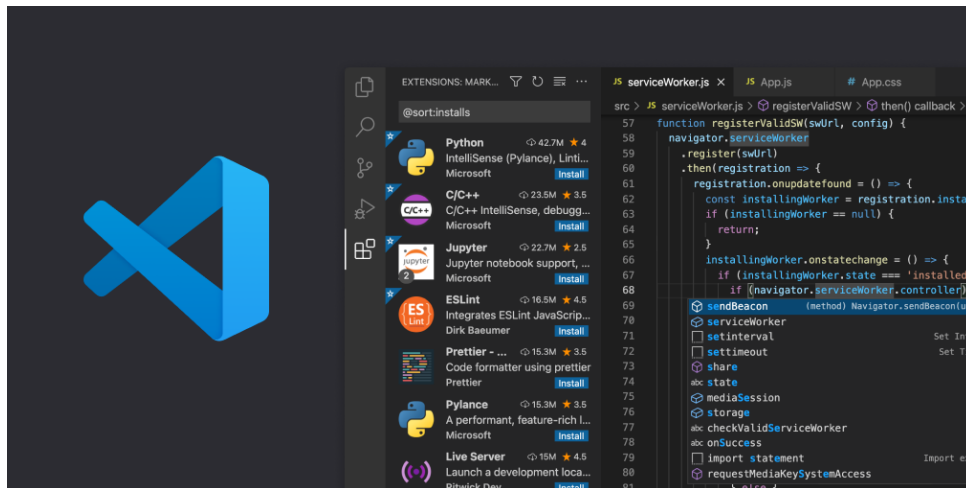
```
def parse_arguments():
    """Read arguments from a command line."""
    parser = argparse.ArgumentParser(description='Arguments get parsed via --commands')
    parser.add_argument('-v', metavar='verbosity', type=int, default=2,
                        help='Verbosity of logging: 0 -critical, 1 -warning, 2 -info, 3 -debug')

    args = parser.parse_args()
    verbose = {0: logging.CRITICAL, 1: logging.WARNING, 2: logging.INFO, 3: logging.DEBUG}
    logging.basicConfig(format='%(message)s', level=verbose[args.v], stream=sys.stdout)

    return args

def main():
    pass

if __name__ == '__main__':
    args = parse_arguments()
    main()
```

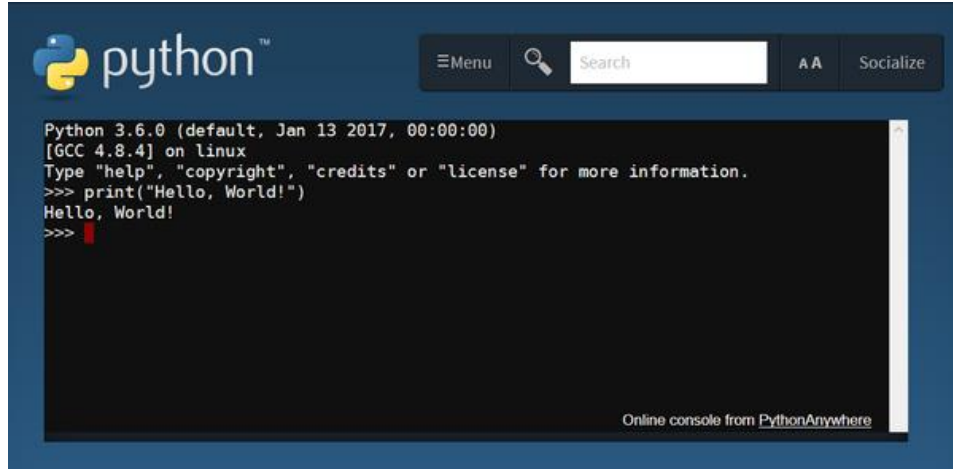


The screenshot shows the Visual Studio Code editor with a Jupyter notebook open. The notebook has two cells. The first cell contains code to read a CSV file using pandas: `import pandas as pd; kernel_stats = pd.read_csv("libraries_by_python_version.csv"); kernel_stats`. The output of this cell is a table with 20 rows and 7 columns. The second cell contains code to create a pie chart using matplotlib: `import matplotlib.pyplot as plt; plt.pie(kernel_stats['total_count'], labels=kernel_stats['library']); plt.show()`. The status bar at the bottom indicates that packages were installed successfully.

	library	kernel_python 3.4	kernel_pytho
16	warnings	3822	37276
7	time	8722	86455
8	tensorflow	4212	123837
11	sys	7784	73858
3	sklearn	14938	222672
5	seaborn	9578	105132
6	scipy	12898	112992
19	requests	2429	28801

<https://code.visualstudio.com/>

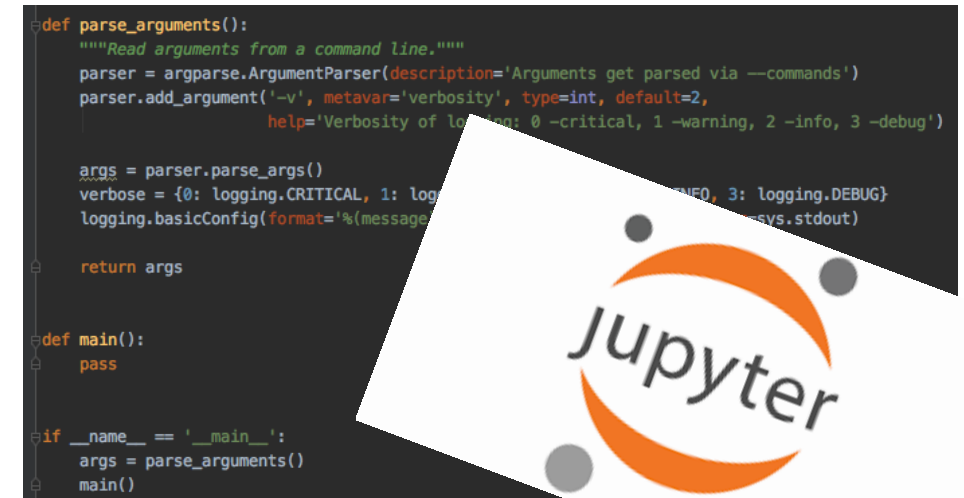
# 2 - ¿Cómo trabajar en Python?



The screenshot shows the Python Anywhere online console interface. At the top, there's a 'python' logo and a search bar. Below, a terminal window displays the following text:

```
Python 3.6.0 (default, Jan 13 2017, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>>
```

At the bottom right, it says "Online console from PythonAnywhere".



The screenshot shows a Jupyter Notebook interface with a dark theme. The code in the cell is as follows:

```
def parse_arguments():
    """Read arguments from a command line."""
    parser = argparse.ArgumentParser(description='Arguments get parsed via --commands')
    parser.add_argument('-v', metavar='verbosity', type=int, default=2,
                        help='Verbosity of logging: 0 -critical, 1 -warning, 2 -info, 3 -debug')

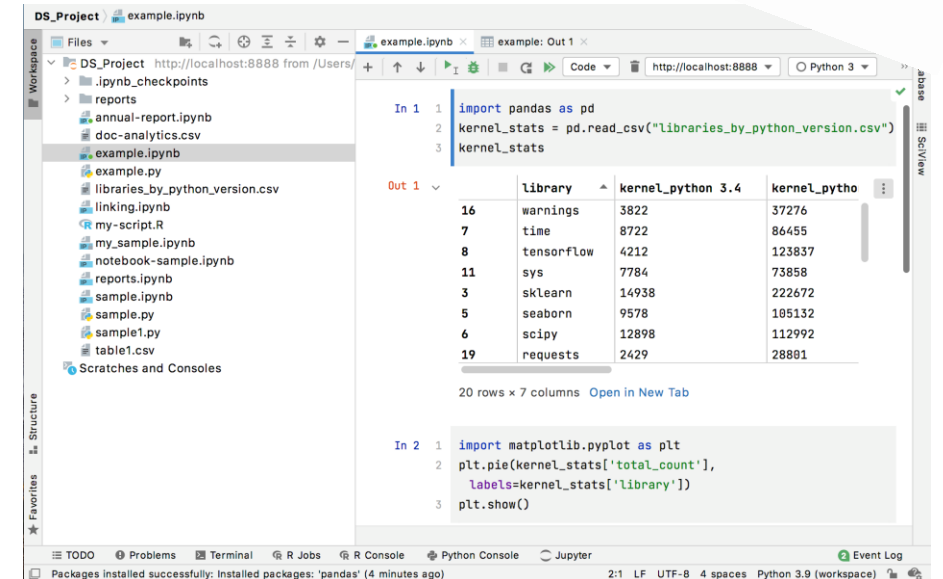
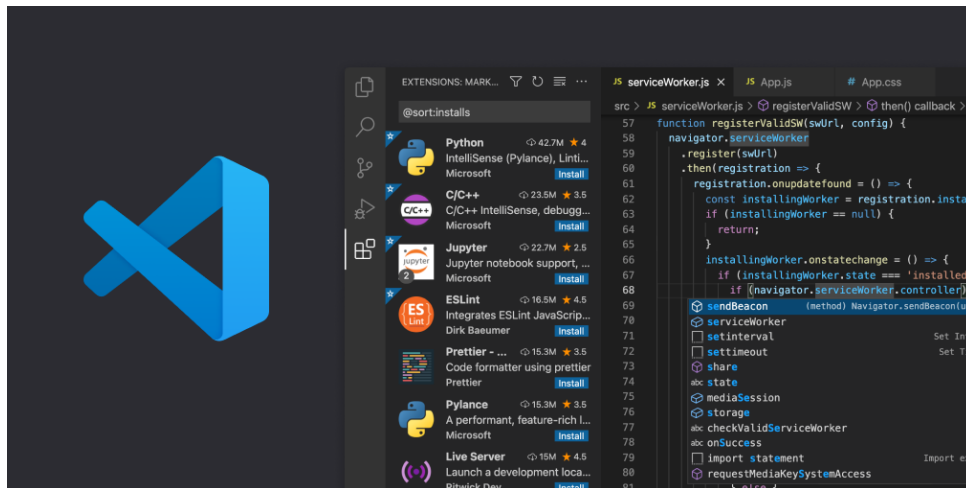
    args = parser.parse_args()
    verbose = {0: logging.CRITICAL, 1: logging.WARNING, 2: logging.INFO, 3: logging.DEBUG}
    logging.basicConfig(format='%(message)s', level=verbose.get(args.verbosity, logging.INFO),
                        stream=sys.stdout)

    return args

def main():
    pass

if __name__ == '__main__':
    args = parse_arguments()
    main()
```

A Jupyter logo is overlaid on the right side of the code.



The screenshot shows a Jupyter Notebook interface with a light theme. The code in the cell is as follows:

```
In 1: import pandas as pd
      kernel_stats = pd.read_csv("libraries_by_python_version.csv")
      kernel_stats
```

The output is a DataFrame with 20 rows and 7 columns. The first few rows are:

	library	kernel_python 3.4	kernel_pytho
16	warnings	3822	37276
7	time	8722	86455
8	tensorflow	4212	123837
11	sys	7784	73858
3	sklearn	14938	222672
5	seaborn	9578	105132
6	scipy	12898	112992
19	requests	2429	28801

Below the table, it says "20 rows x 7 columns Open in New Tab".

The second cell contains the following code:

```
In 2: import matplotlib.pyplot as plt
      plt.pie(kernel_stats['total_count'],
              labels=kernel_stats['library'])
      plt.show()
```

<https://code.visualstudio.com/>



## 2 – Instalar jupyter notebook

<https://towardsdatascience.com/how-to-set-up-anaconda-and-jupyter-notebook-the-right-way-de3b7623ea4a>

*Install the notebook*

```
conda install -c conda-forge notebook
```

```
conda install -c conda-forge nb_conda_kernels
```

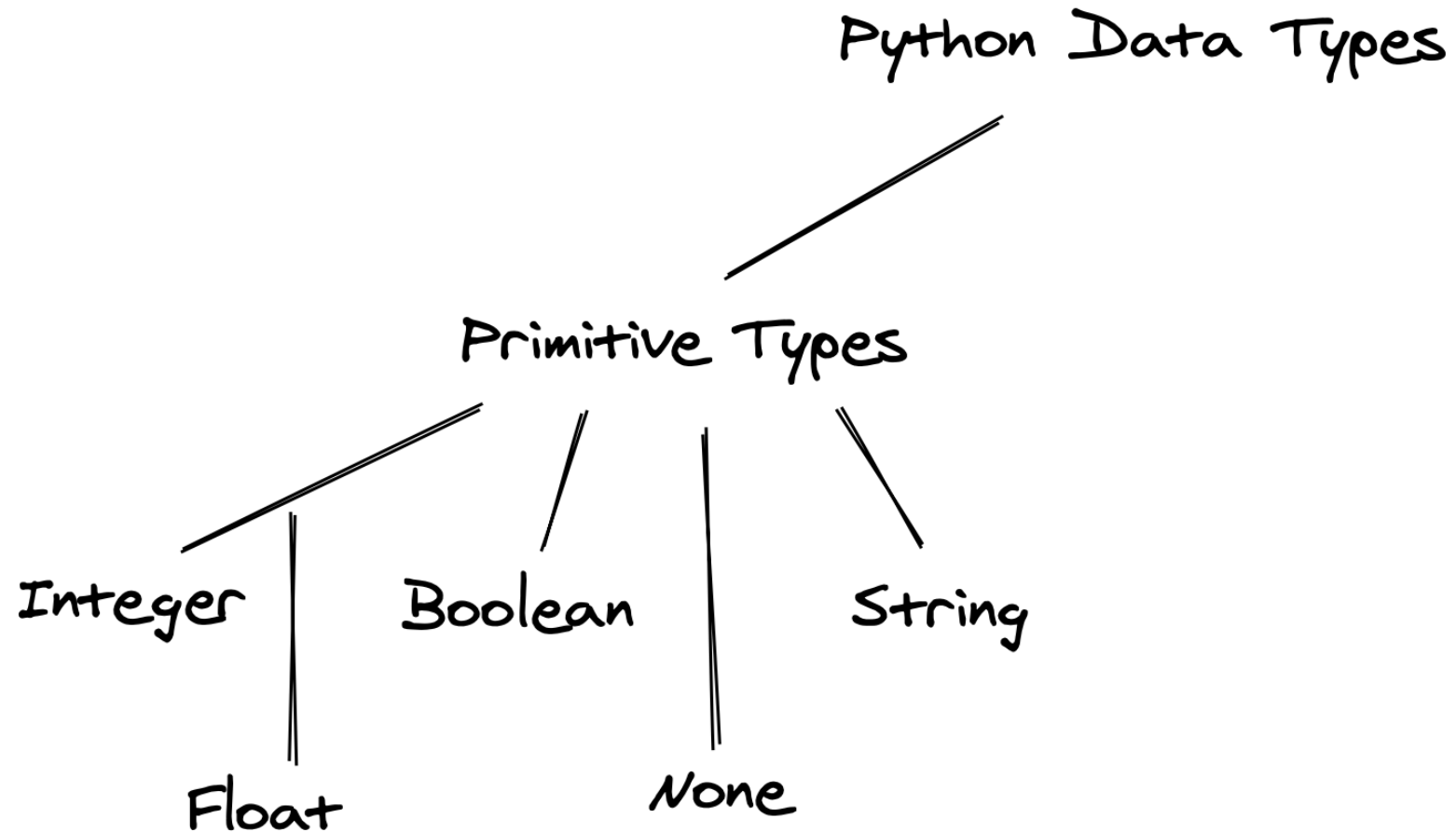
Add environment kernel to notebook

```
conda activate cenv
```

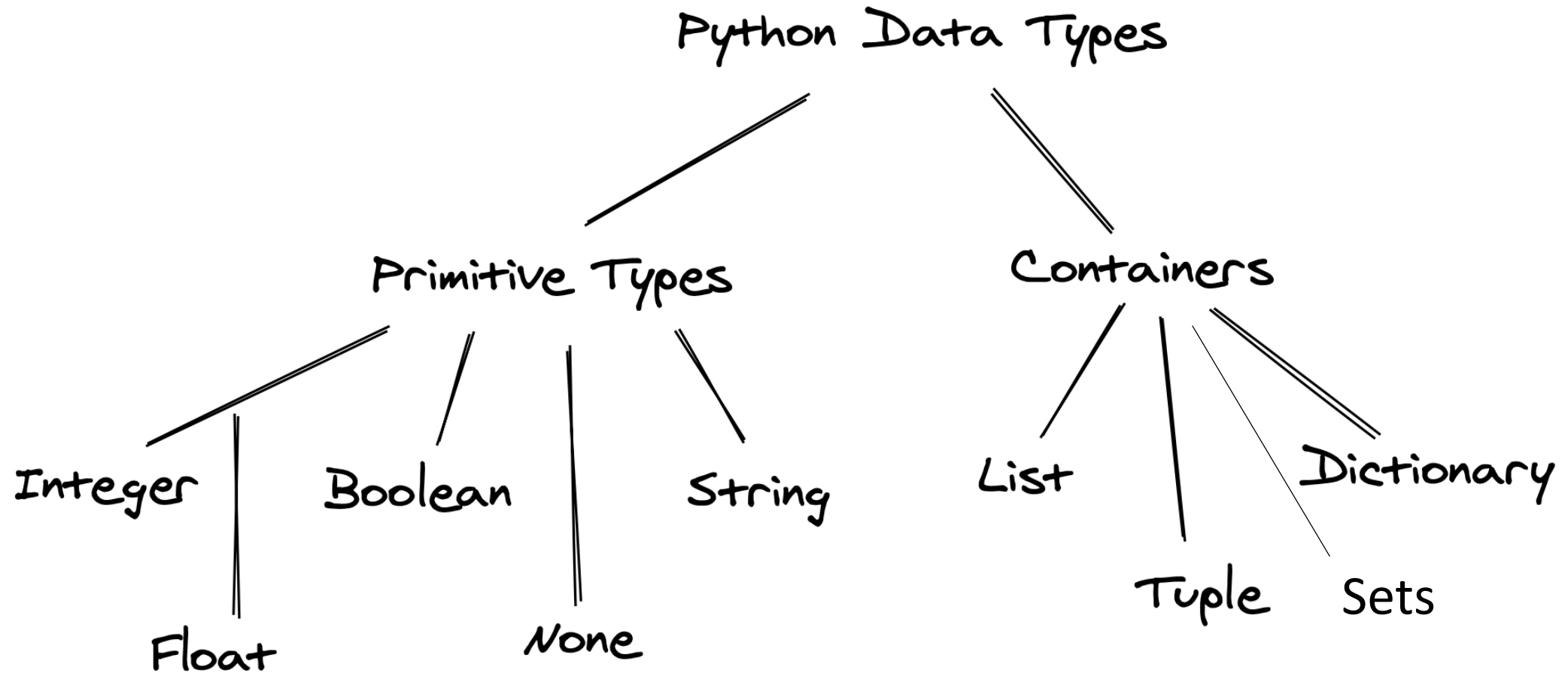
```
(cenv)$ conda install ipykernel
```

```
(cenv)$ ipython kernel install --user --name=<any_name_for_kernel>
```

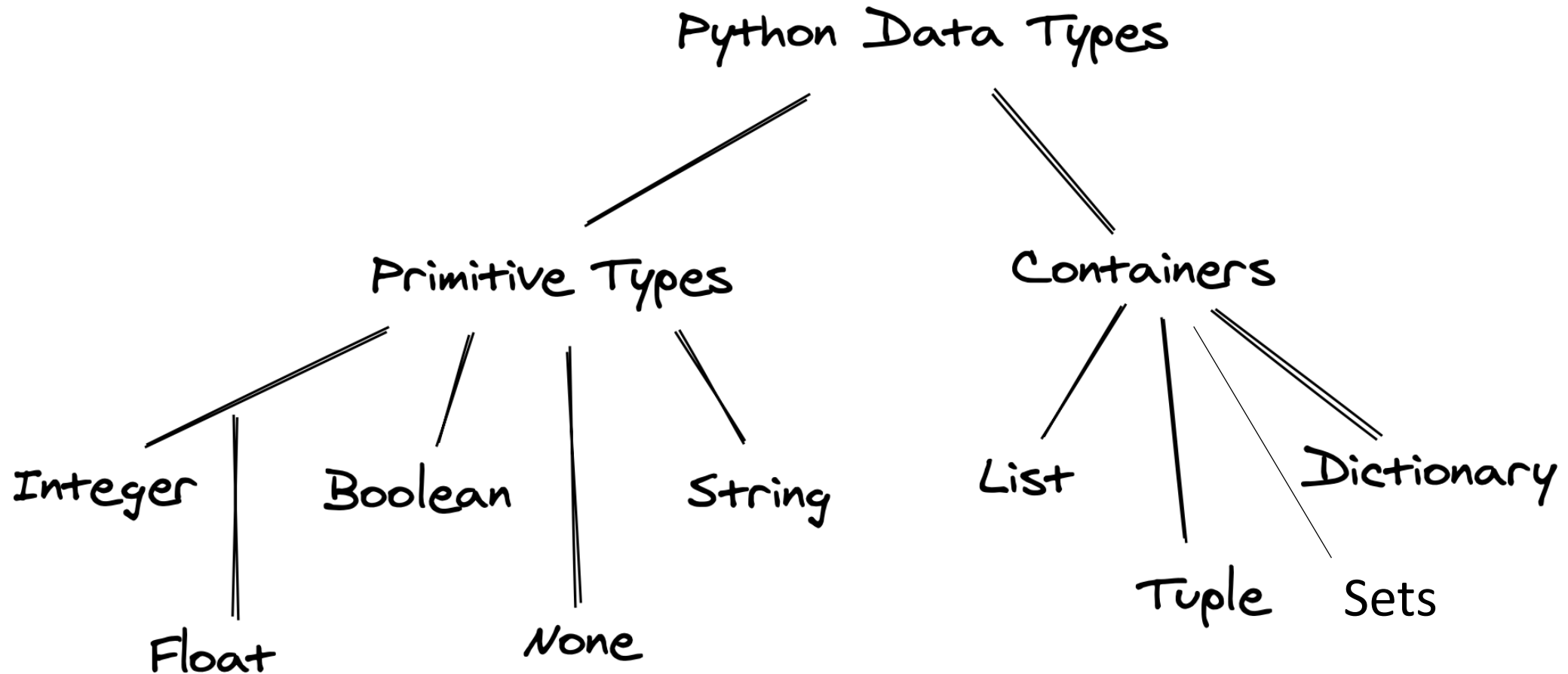
### 3 – Tipos de datos en Python



### 3 – Tipos de datos en Python

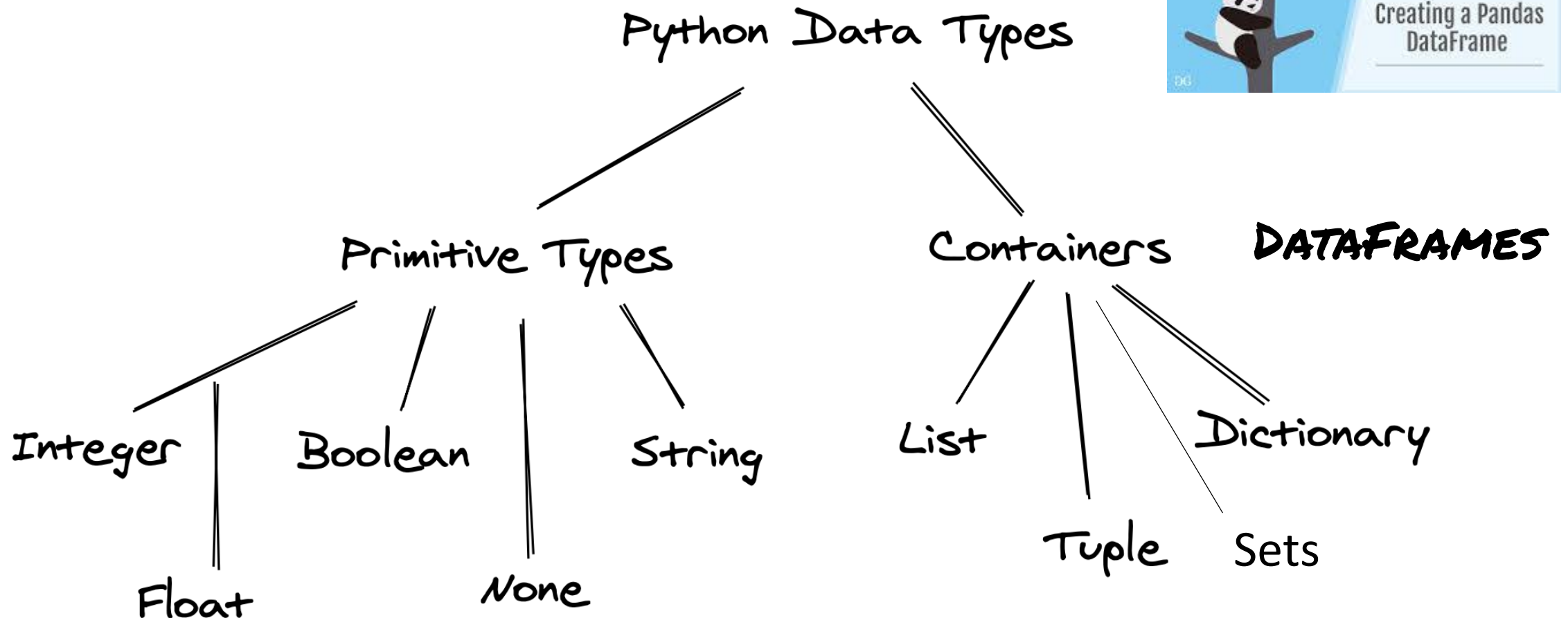


# 3 – Tipos de dados em Python



Acceso a elemento: `container.element`  
Método: `container.method.()`

# 3 – Tipos de dados em Python

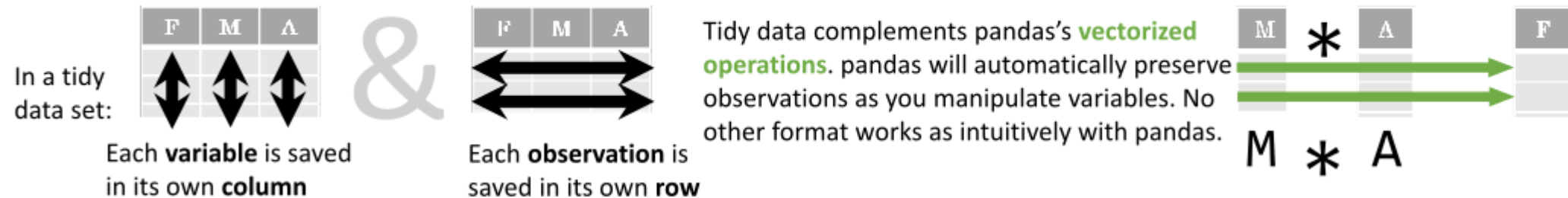


Acceso a elemento: `container.element`  
Método: `container.method.()`

# 3 – Tipos de datos en Python



## Tidy Data – A foundation for wrangling in pandas



Acceso a elemento: `container.element`  
Método: `container.method.()`

# **3 – Creando un DataFrame**

df.info()  
pokemon df.hist()

# 4 – Data Overview

## Consultar Inicio y Final

df.head(5)

df.tail(5)

## Recuperar el número de filas y columnas (shape)

df.shape

## Información general

df.info() -> tipos de datos

df.describe() -> estadística básica

df.hist() -> histograma



# 4 – Data Overview

## Consultar Inicio y Final

`df.head(5)`

`df.tail(5)`

## Información general

`df.info()` -> tipos de datos

`df.describe()` -> estadística básica

## Información de distribución de datos y correlaciones

`df.hist()` -> histograma

`df.corr()` -> correlación

## Recuperar el número de filas y columnas (shape)

`df.shape`

pandas provides a large set of [summary functions](#) that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

[`sum\(\)`](#)

Sum values of each object.

[`count\(\)`](#)

Count non-NA/null values of each object.

[`median\(\)`](#)

Median value of each object.

[`quantile\(\[0.25,0.75\]\)`](#)

Quantiles of each object.

[`apply\(function\)`](#)

Apply function to each object.

[`min\(\)`](#)

Minimum value in each object.

[`max\(\)`](#)

Maximum value in each object.

[`mean\(\)`](#)

Mean value of each object.

[`var\(\)`](#)

Variance of each object.


[`std\(\)`](#)

Standard deviation of each object.

# 5 – Selección de datos

Selección **Booleana**: Atendiendo a los valores de las celdas

Pandas Filter or Select Rows Based on Column Values



	C1		
	10		
	2		
	20		

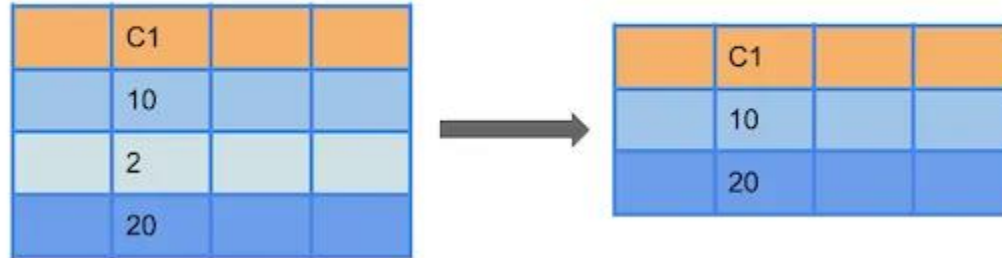
	C1		
	10		
	20		

Selección **Indexada**: Atendiendo a las etiquetas que hayamos usado para indexar

# 5 – Selección de datos

Selección **Booleana**: Atendiendo a los valores de las celdas

Pandas Filter or Select Rows Based on Column Values



Selección **Indexada**: Atendiendo a las etiquetas que hayamos usado para indexar

The diagram shows a large table with 20 rows and 4 columns (Year, Rank, Name). The rows are indexed from 0 to 19. Rows 8, 12, 15, and 17 are highlighted with red boxes. An arrow points to a smaller table with 4 rows and 4 columns, containing the data from the highlighted rows.

	Year	Rank	Name
0	1984	5	METTLE
1	1985	16	DESSON
2	1983	6	ANANDA
3	1983	10	POWELL
4	1987	18	TROIANO
5	2013	5	SETHI
6	1989	14	BEAN
7	1983	16	WATSON
8	1983	16	CHRISTIAN
9	1989	24	DELL
10	2005	2	PERLEY
11	2009	20	BLAIR
12	1987	16	CHRISTIAN
13	1978	10	BEAN
14	2007	24	ALEXA
15	1983	2	BLAIR
16	1975	14	PERLEY
17	1984	20	DELL
18	2009	16	PERLEY
19	1985	2	ARCHER


	Year	Rank	Name
8	1983	16	CHRISTIAN
12	1987	16	CHRISTIAN
15	1983	2	BLAIR
17	1984	20	DELL

# 5 – Selección de datos

Selección **Booleana**: Atendiendo a los valores de las celdas

Pandas Filter or Select Rows Based on Column Values

	C1		
	10		
	2		
	20		




	C1		
	10		
	20		

Selección **Indexada**: Atendiendo a las etiquetas que hayamos usado para indexar

Selección por tipo de dato

	Year	Rank	Name
0	1984	5	METHEL
1	1985	16	DESSIN
2	1983	6	ARABIA
3	1983	10	POWELL
4	1987	18	TROIANO
5	1988	5	SETHI
6	1988	14	BEAN
7	1983	16	WATSON
8	1983	16	CHRISTIAN
9	1988	24	DELL
10	2005	2	PERLEY
11	2008	20	BLAIR
12	1987	18	CHRISTIAN
13	1988	16	BEAN
14	2007	24	ALEXA
15	1983	2	BLAIR
16	1988	14	BEAN
17	1988	16	DELL
18	2008	16	CHRISTIAN
19	1988	2	ARCHER



	Year	Rank	Name
8	1983	16	CHRISTIAN
12	1987	18	CHRISTIAN
15	1983	2	BLAIR
17	1988	16	DELL

# 6 – Limpieza de datos

Información que falta: NaN

- \* Los localizamos con **.isna()**
- \* Eliminamos las filas con NaN con **.dropna()**
- \* Rellenamos los valores que faltan con **.fillna()**

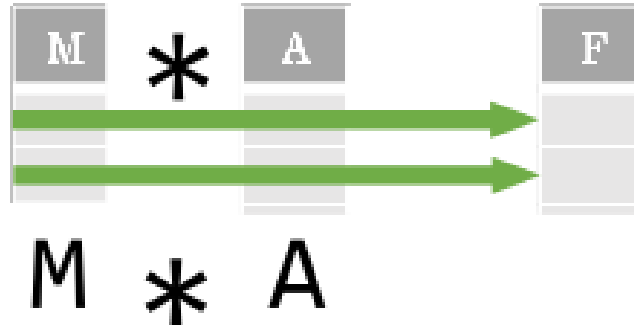
Renombrando columnas y mapeando valores:

Cambiar nombres **de** columnas: **df.rename(columns=diccionario)**

Cambiar valores **en** la columns: **df['column'].map(diccionario)**

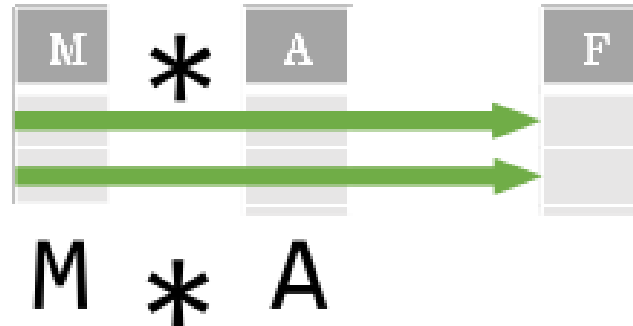
# 7 – Operando en la DataFrame

Las Operaciones que hagamos van a ser vectoriales por defecto (afectan a toda la df)



# 7 – Operando en la DataFrame

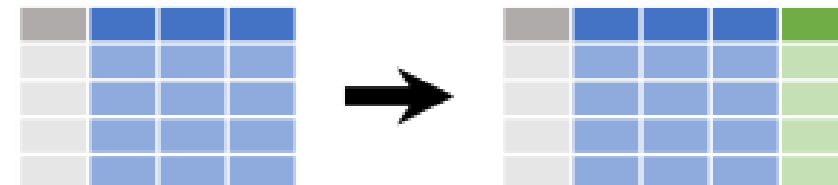
Las Operaciones que hagamos van a ser vectoriales por defecto (afectan a toda la df)



Para guardar la información **creamos nuevas columnas**:

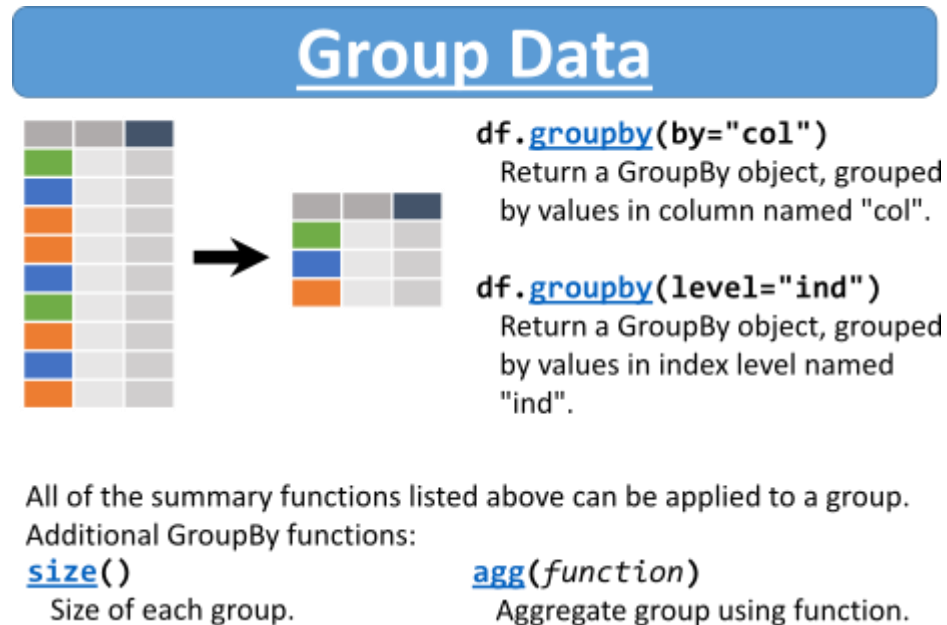
`df.loc[:, "Name"] = operation on rows`

Make New Columns



# 8 – Agrupando información y resumiendo

Agrupamos por categorías y aplicamos agregación con las summary functions



pandas provides a large set of [summary functions](#) that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`

Sum values of each object.

`count()`

Count non-NA/null values of each object.

`median()`

Median value of each object.

`quantile([0.25,0.75])`

Quantiles of each object.

`apply(function)`

Apply function to each object.

`min()`

Minimum value in each object.

`max()`

Maximum value in each object.

`mean()`

Mean value of each object.

`var()`

Variance of each object.

`std()`

Standard deviation of each object.



# 9 – Correlaciones

Podemos usar métodos solamente numéricos : `df.corr()`

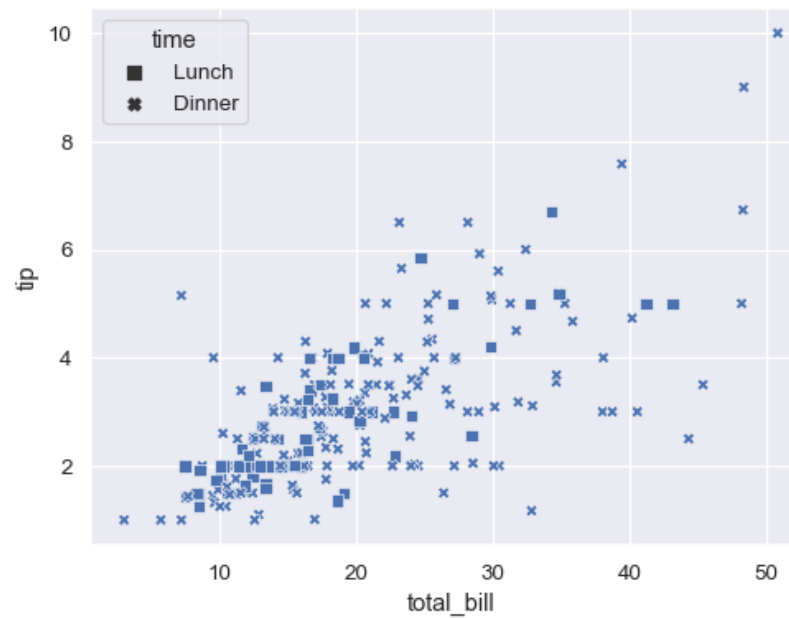


Podemos usar métodos más gráficos

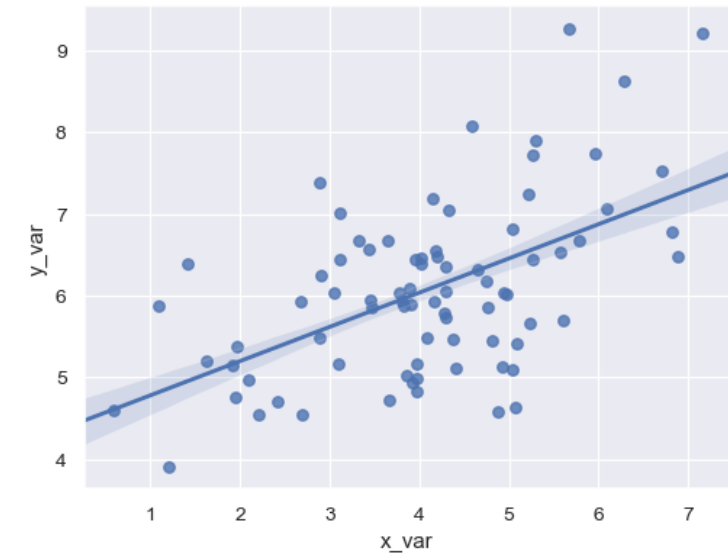
`sns.heatmap(df.corr())`



`sns.scatterplot()`



`sns.regplot()`



# 10 – Sorting and Ranking

Podemos usar métodos solamente numéricos : **df.corr()**

## Ranking

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

[shift\(1\)](#)

Copy with values shifted by 1.

[rank\(method='dense'\)](#)

Ranks with no gaps.

[rank\(method='min'\)](#)

Ranks. Ties get min rank.

[rank\(pct=True\)](#)

Ranks rescaled to interval [0, 1].

[rank\(method='first'\)](#)

Ranks. Ties go to first value.

[shift\(-1\)](#)

Copy with values lagged by 1.

[cumsum\(\)](#)

Cumulative sum.

[cummax\(\)](#)

Cumulative max.

[cummin\(\)](#)

Cumulative min.

[cumprod\(\)](#)

# 11 – Combinando DataFrames: merge y concat

Merge: Une dos DataFrames con al menos un índice en común

## Combine Data Sets

adf		bdf		
x1	x2	x1	x3	
A	1	A	T	+
B	2	B	F	
C	3	D	T	
				=

### Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`  
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`  
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`  
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`  
Join data. Retain all values, all rows.

### Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`  
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`  
All rows in adf that do not have a match in bdf.

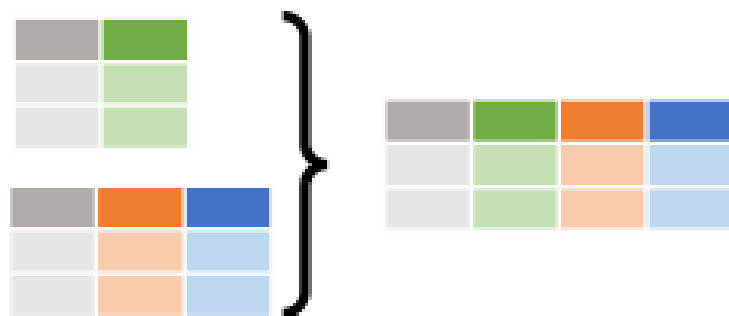
# 11 – Combinando DataFrames: merge y concat

**Merge:** Une dos DataFrames con al menos un índice en común

**Concat:** Combina **varias** DataFrames en una nueva



`pd.concat([df1, df2])`  
Append rows of DataFrames



`pd.concat([df1, df2], axis=1)`  
Append columns of DataFrames

## Combine Data Sets

adf		bdf		
x1	x2	x1	x3	
A	1	A	T	+
B	2	B	F	
C	3	D	T	
				=

### Standard Joins

x1	x2	x3	pd.merge(adf, bdf, how='left', on='x1')
A	1	T	Join matching rows from bdf to adf.
B	2	F	
C	3	NaN	

x1	x2	x3	pd.merge(adf, bdf, how='right', on='x1')
A	1.0	T	Join matching rows from adf to bdf.
B	2.0	F	
D	NaN	T	

x1	x2	x3	pd.merge(adf, bdf, how='inner', on='x1')
A	1	T	Join data. Retain only rows in both sets.
B	2	F	

x1	x2	x3	pd.merge(adf, bdf, how='outer', on='x1')
A	1	T	Join data. Retain all values, all rows.
B	2	F	
C	3	NaN	
D	NaN	T	

### Filtering Joins

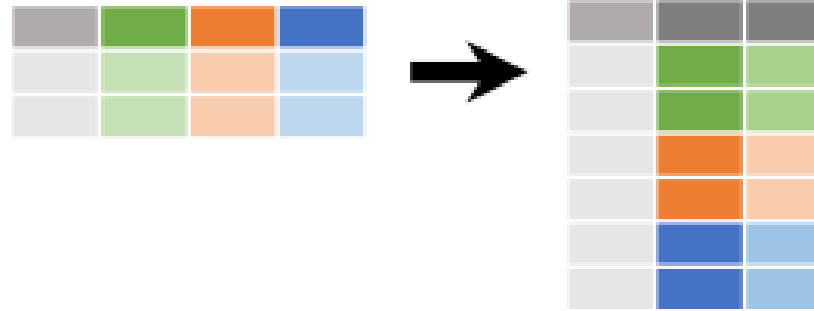
x1	x2	adf[adf.x1.isin(bdf.x1)]
A	1	All rows in adf that have a match in bdf.
B	2	

x1	x2	adf[~adf.x1.isin(bdf.x1)]
C	3	All rows in adf that do not have a match in bdf.

# 12 – Reestructurar datos: .melt() y .pivot()

Hay casos en los que estaremos obligados a cambiar la estructura de la base de datos, porque por ejemplo no sea tidy.

`pd.melt()` nos permite hacerlo



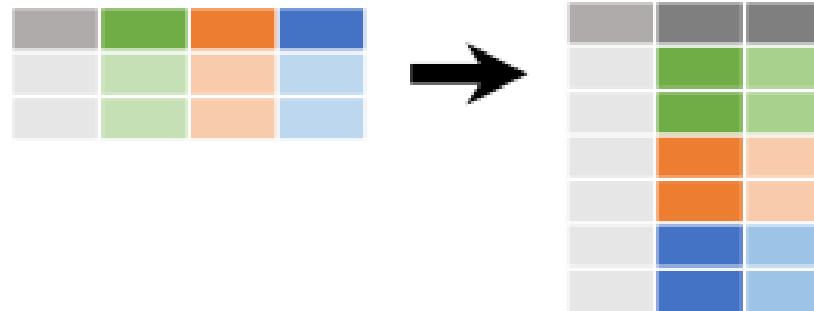
`pd.melt(df)`

Gather columns into rows.

# 12 – Reestructurar datos: .melt() y .pivot()

Hay casos en los que estaremos obligados a cambiar la estructura de la base de datos, porque por ejemplo no sea tidy.

`pd.melt()` nos permite hacerlo ->



`pd.melt(df)`

Gather columns into rows.



`df.pivot(columns='var', values='val')`

Spread rows into columns.

En otros casos nos interesará pasar de una estructura tidy a una table cruzada, por ejemplo para convertir una lista de adyacencias de una red en una matriz.

# ¡Gracias por tu atención!

[https://pandas.pydata.org/docs/user\\_guide/index.html](https://pandas.pydata.org/docs/user_guide/index.html)

¡Muchos ejercicios!

<https://www.w3resource.com/python-exercises/pandas/index.php>

