¿Jupyter Notebooks en producción? ¿Es ello posible?

Rodolfo Ferro

Developer Advocate @ Ploomber
(Twitter/Instagram: @rodo_ferro)

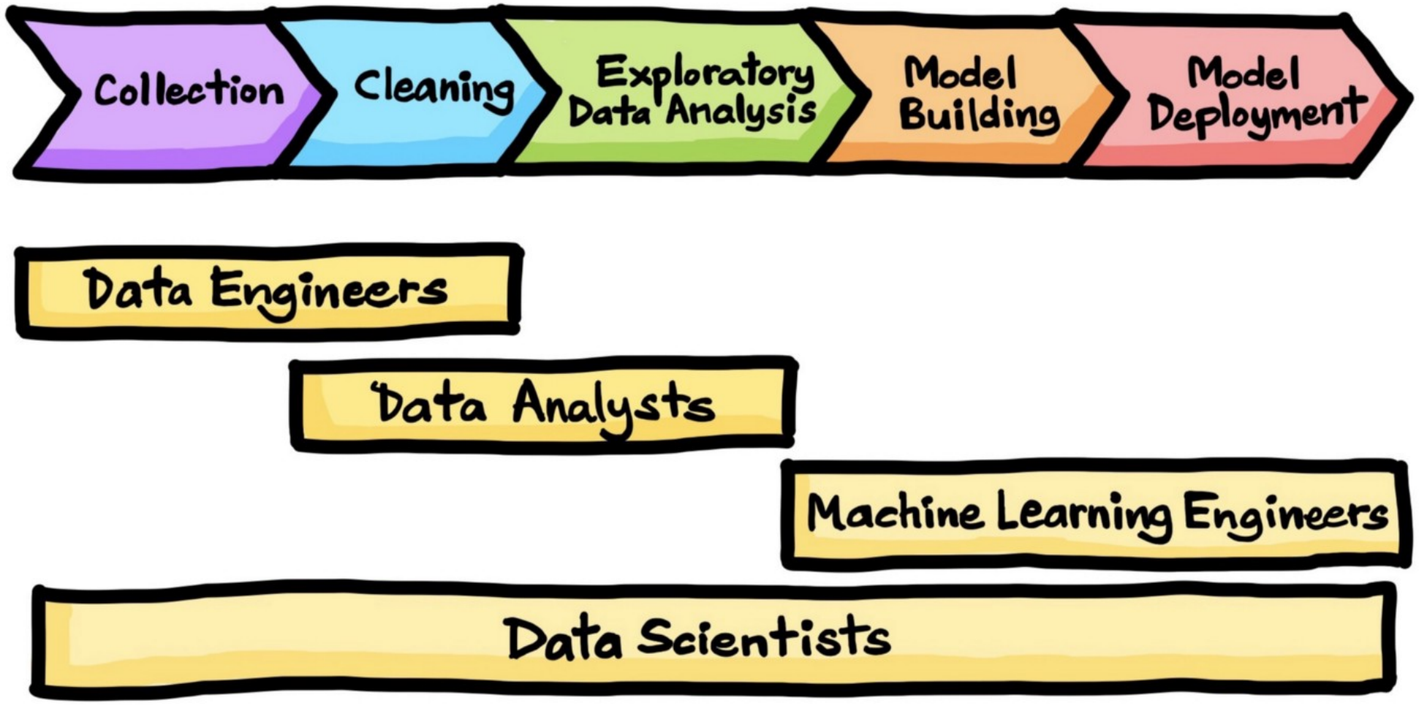
ploomber.io

¿Es ello posible? [1] [2]

[1] Fuente: Unsplash

[2] Prueba: Evidation Health (ploomber.io/blog/evidation)

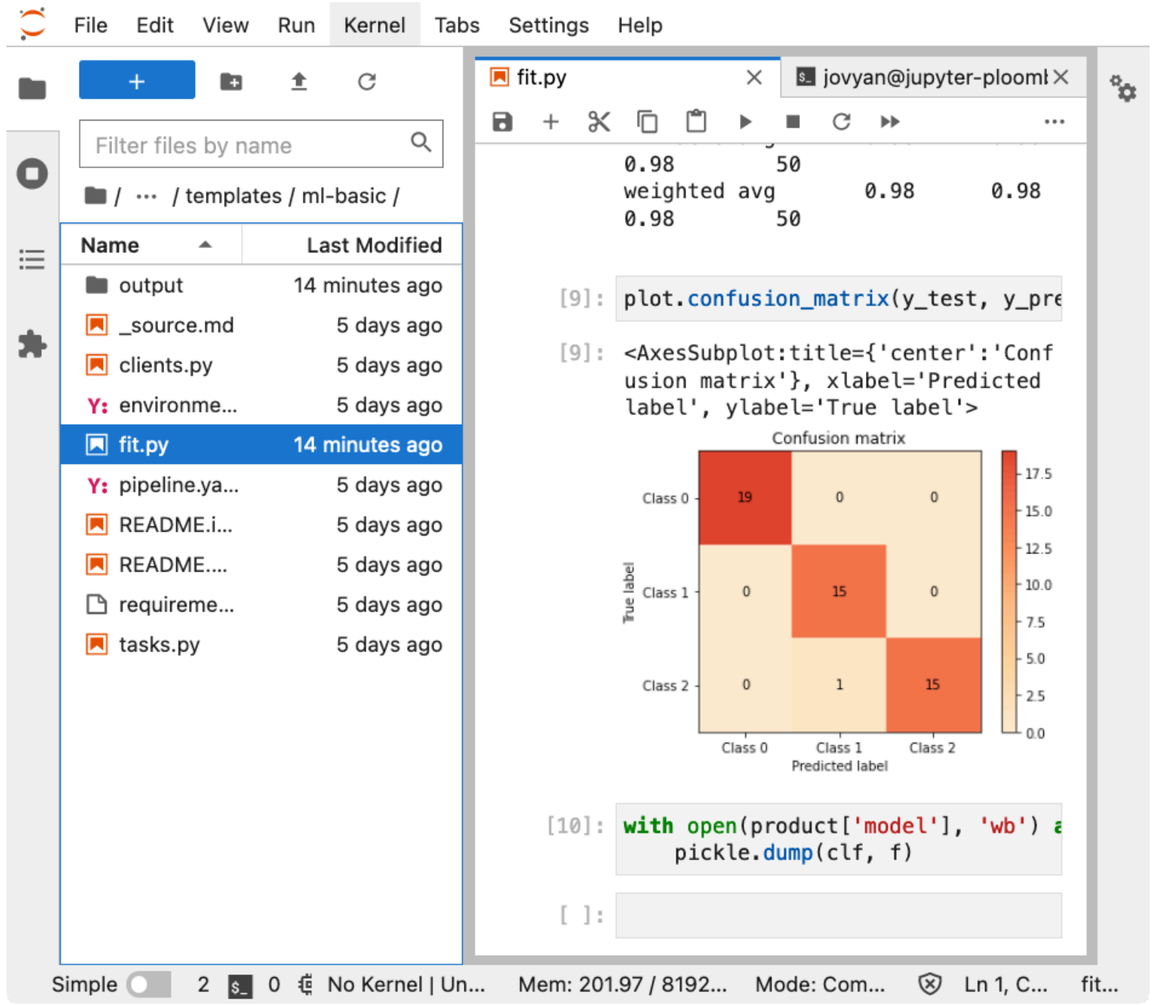
Trabajar con datos es un proceso **robusto** [3]



[3] Fuente: The Data Science Process

Jupyter: un **formato** y una **plataforma**

```
{
  "metadata" : {
    "kernel_info": {
        "name" : "name of the kernel"
    },
    "language_info": {
        "name" : "programming language",
        "version": "version of the language",
    }
  },
  "nbformat": 4,
  "nbformat_minor": 0,
  "cells" : [
  ],
}
```



Formato: nbformat

Truco: ¡Cambiar el formato! [4]

1. Mejor integracion con `git`
2. Mejor interoperabilidad con otros ambientes de desarollo (VSCode, Spyder, PyCharm)

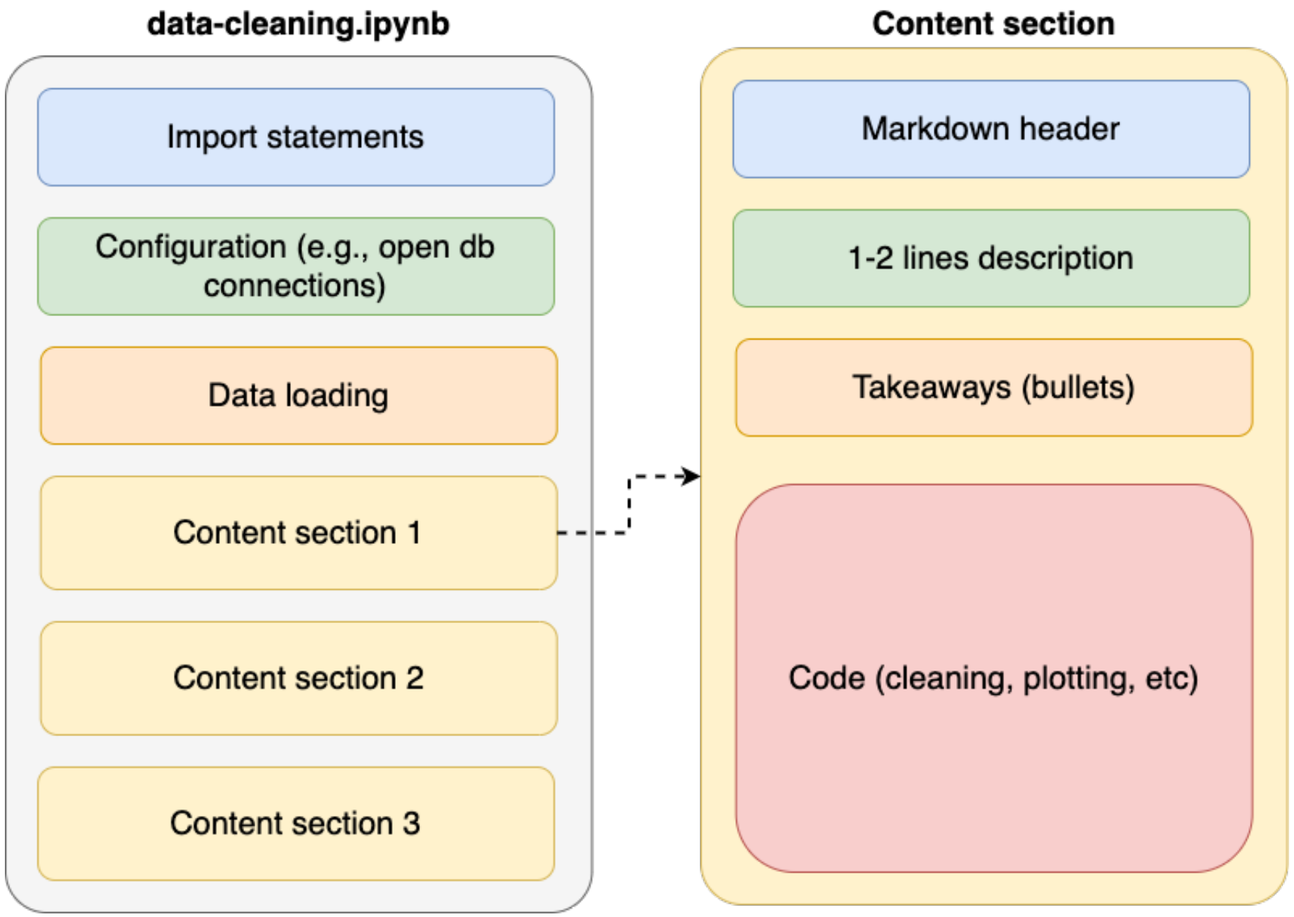
```
# data-cleaning.py
import pandas as pd

# %%
# one cell
df = pd.read_csv('my-data.csv')

# %%
# another cell
df['new-column'] = df['column'] + 1
```

[4] Paquete: jupytext

Parte I: Cuadernos limpios  [5]

[5] Lectura recomendada: ploomber.io/blog/clean-nbs (On Writing Clean Notebooks)

Los **cuadernos limpios** son cuadernos cortos



Los **cuadernos limpios** tienen operaciones sin efectos laterales

Efecto lateral: **¡evitar!** ⚠️

```
import pandas as pd

def add_one(df):
    df['zeros'] = df['zeros'] + 1

df = pd.DataFrame({'zeros': [0, 0, 0]})

# more code...

# a few dozen cells below...
add_one(df)

df
#    zeros
# 0      1
# 1      1
# 2      1
```

¡Sin efectos! ✅

```
def add_one(df):
    # copying the data frame!
    another = df.copy()
    another['zeros'] = another['zeros'] + 1
    return another

df = pd.DataFrame({'zeros': [0, 0, 0]})

ones = add_one(df)

df
#    zeros
# 0      0
# 1      0
# 2      0
```

Los **cuadernos limpios** declaran dependencias 🔒

1. Actualizaciones de paquetes pueden romper tu proyecto
2. Hará más difícil correr el cuaderno en el futuro

```
# requirements.lock.txt
package-a==1.1
package-b==2.4
...
package-z==0.4
```

Después de instalar las dependencies:

```
pip freeze > requirements.lock.txt
```

Para volver a crear el ambiente:

```
pip install -r requirements.lock.txt
```

Los **cuadernos limpios** siguen estándares [6]

Antes 👀

```
[ ]: df = pd.DataFrame({"x": np.random.rand(10),
"y": np.random.rand(10),
        "x": np.random.rand(10),})
```

```
[ ]: something = True
another = False

if something
\or another:
    print("at least one is True!")
```

Después 

```
[ ]: df = pd.DataFrame(
    {
        "x": np.random.rand(10),
        "y": np.random.rand(10),
        "x": np.random.rand(10),
    }
)
```

```
[ ]: something = True
another = False

if something or another:
    print("at least one is True!")
```

Comando: `soorgeon clean nb.ipynb`

---

[6] Paquete Soorgeon (Ploomber)

Los **cuadernos limpios** separan la **lógica** de la **narrativa** 📊

1. **Lógica:** funciones, clases
2. **Narrativa:** gráficas, transformaciones de datos

Define la lógica en un archivo:

```python
# transformations.py
def add_one(series):
    return series + 1
```

```
[1]: import pandas as pd
     import numpy as np

[2]: # import from transformations.py
     from transformations import add_one

[3]: df = pd.DataFrame({"x": np.random.rand(10)})

[4]: df["y"] = add_one(df.x)

[5]: df.head(3)
```

[5]:

	x	y
0	0.010910	1.010910
1	0.201756	1.201756
2	0.404869	1.404869

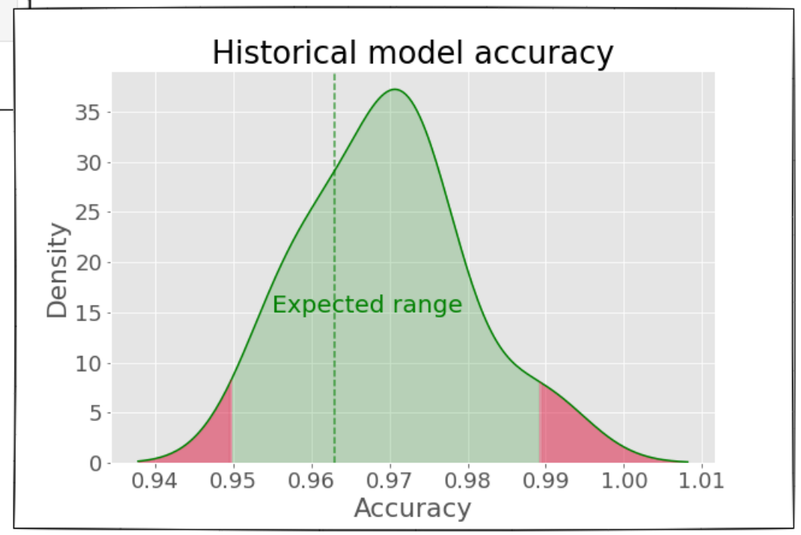
Parte II: Probando cuadernos (*testing*) 🧪 [7]

[7] Lectura recomendada: ploomber.io/blog/ci-for-ds y ploomber.io/blog/ml-testing-i

Probando cuadernos: Salidas de las celdas [8]

```
[7]: y_pred = clf.predict(X_test)

[8]: acc = accuracy_score(y_test,
                          y_pred)
     print(f'{acc:.3f}')

     0.963
```

```
nbsnapshot test nb.ipynb

Testing nb.ipynb...
Checking "accuracy"...
Value within expected range!
No issues found.
```



[8] Paquete: nbsnapshot (por Ploomber)

Probando cuadernos: Archivos de salida [9]

- **Ejemplo:** Pruebas de calidad de datos

```
# assume your notebook produces df
def check_data_quality(df):
    # no nas
    assert df['column'].isna().sum() == 0
    # no negative numbers
    assert (df['column'] < 0).sum() == 0
```

[9] Paquete: Ploomber

Probando cuadernos: Definiciones

Narrativa y lógica separada [10]

```python
from transformations import add_one

def test_add_one():
    assert add_one(1, 2) == 3
```

Lógica embebida [11]

```python
from testbook import testbook


@testbook('path/to/nb.ipynb')
def test_func(tb):
    add_one = tb.get("add_one")

    assert add_one(1, 2) == 3
```

---

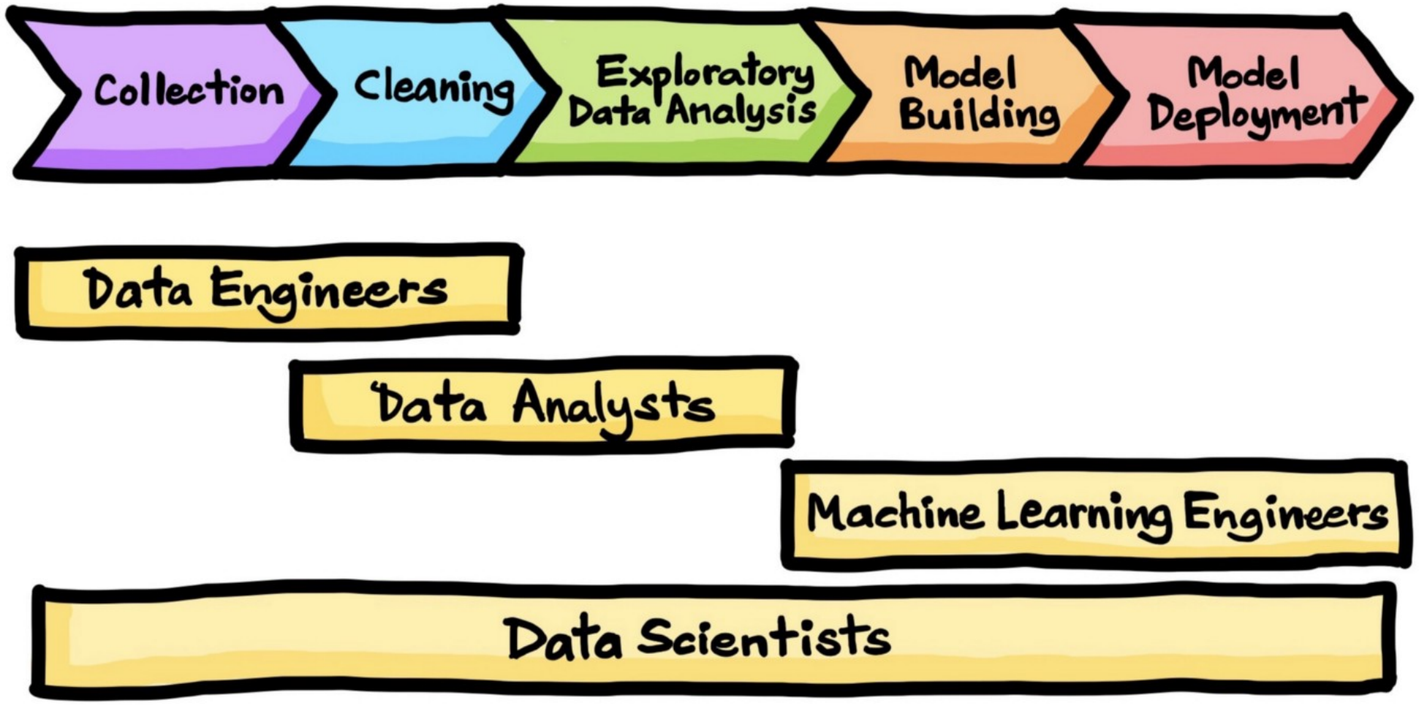
[10] Paquete: pytest

[11] Paquete: testbook

Parte III: Construyendo *pipelines* de datos 🛠️ [12]

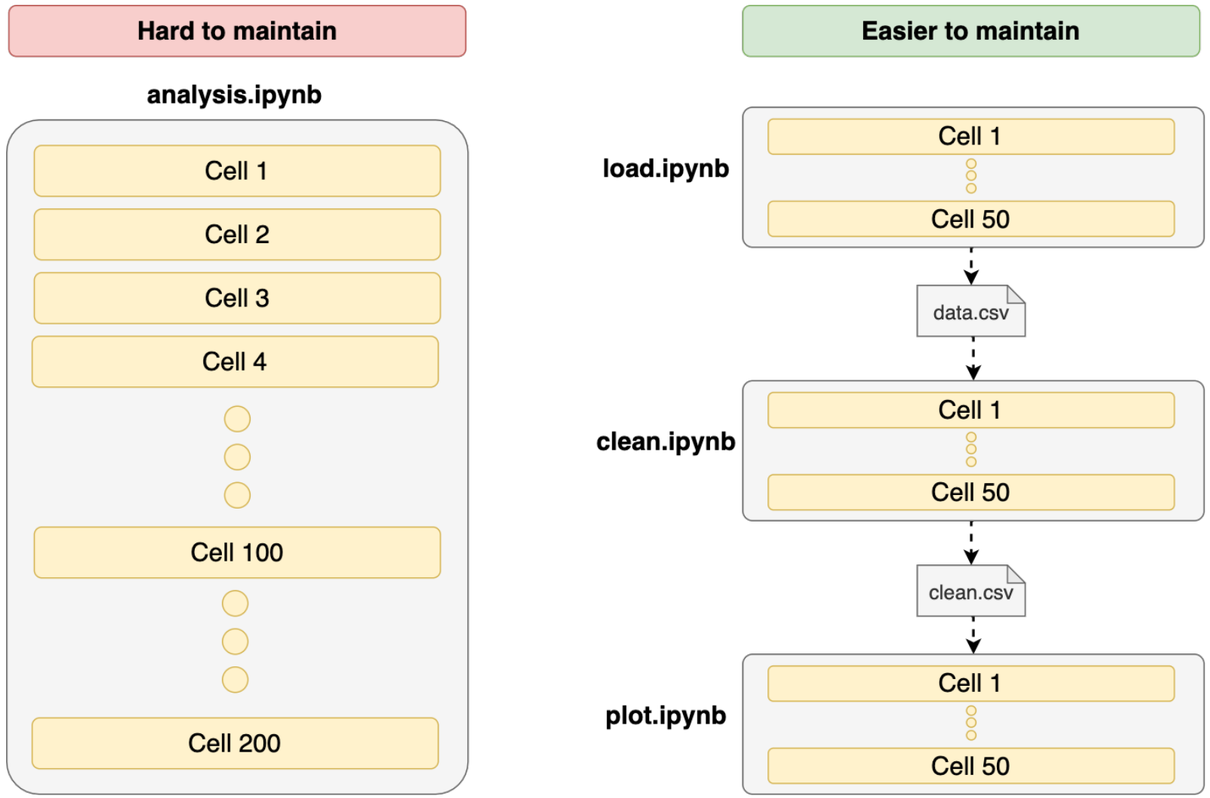
[12] Lectura recomendada: ploomber.io/blog/clean-pipelines

Trabajar con datos es un proceso **robusto** [13]



[13] Fuente: The Data Science Process

Construyendo *pipelines* de datos [14]

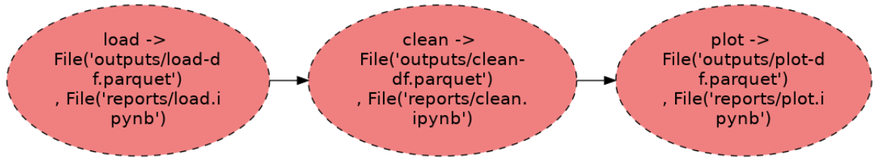


[14] Construyendo pipelines: Ploomber. Convertir cuadernos existentes: Soorgeon (por Ploomber)

Construyendo *pipelines* de datos

```yaml
# pipeline.yaml
tasks:
    - source: tasks/load.py
      product:
          df: outputs/load-df.parquet
          nb: reports/load.ipynb

    - source: tasks/clean.py
      product:
          df: outputs/clean-df.parquet
          nb: reports/clean.ipynb

    - source: tasks/plot.py
      product:
          df: outputs/plot-df.parquet
          nb: reports/plot.ipynb
```

El archivo `pipeline.yaml` es utilizado para construir tus pipelines[15]:



Con el CLI de Ploomber:

```
ploomber scaffold
ploomber plot --include-products
ploomber build
```

[15] Lectura sugerida: Tu primer *pipeline* con Python

Recursos

- Curso gratuito: notebooks.academy
- Presentación (con ligas): rodolfoferro.xyz/assets/talks/calzada-code-ploomber.pdf
- Comunidad: ploomber.io/community

¡Gracias! 🎉

Contacto: rodo@ploomber.io

Redes: @ploomber
(Síguenos, publicamos memes los viernes.)