

¿Jupyter Notebooks en
producción? ¿Es ello
posible?

Rodolfo Ferro

Developer Advocate @ Ploomber
(Twitter/Instagram: [@odo_ferro](#))



ploomber.io

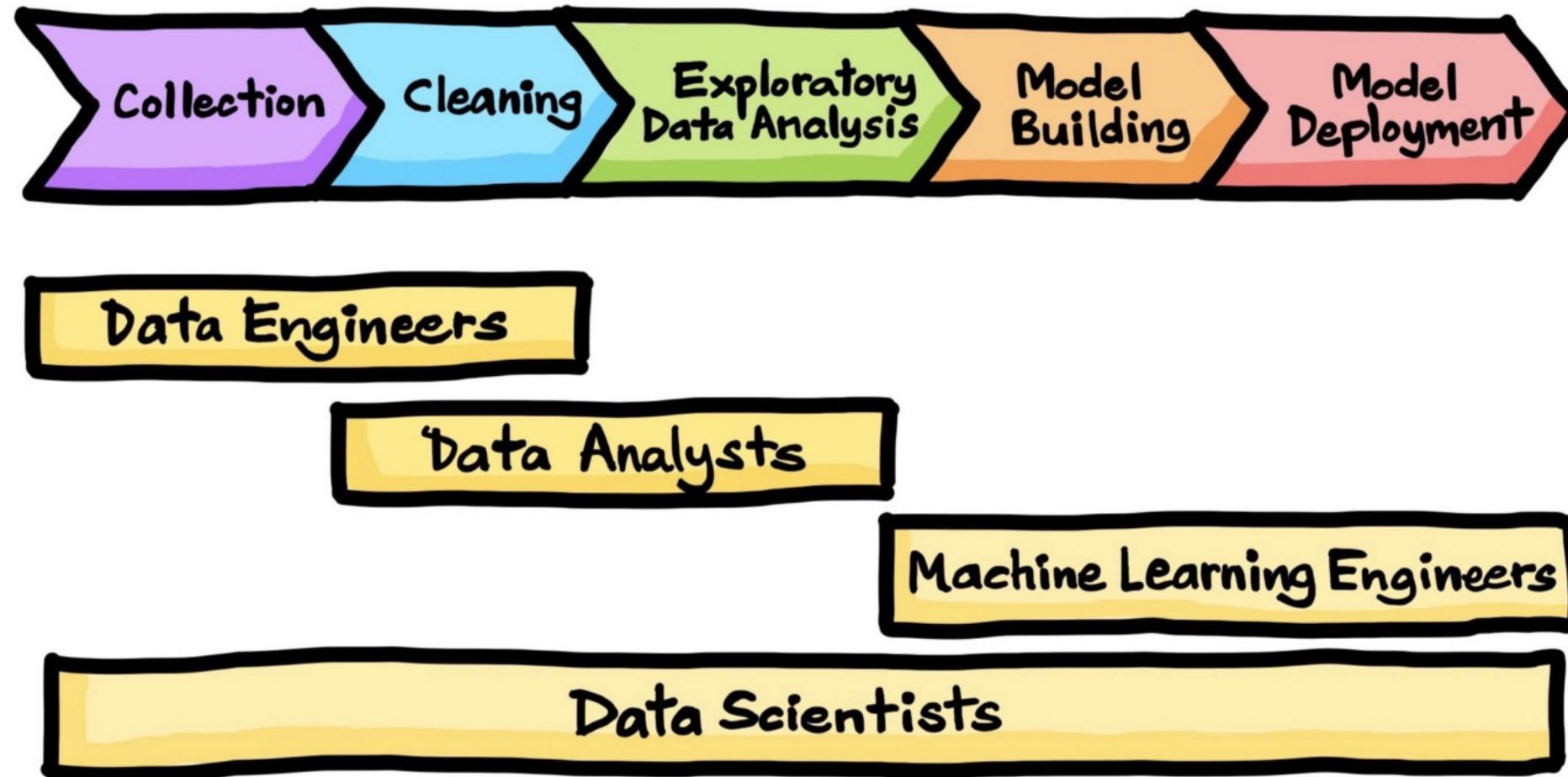
¿Es ello posible? ¹ ²

¹ Fuente: [Unsplash](#)

² Prueba: [Evidation Health \(ploomber.io/blog/evidation\)](https://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": [  
    ]  
}
```

The screenshot shows a Jupyter Notebook interface with the following components:

- File Explorer:** On the left, it shows a file tree under the path `/.../templates/ml-basic/`. The files listed are: output (14 minutes ago), _source.md (5 days ago), clients.py (5 days ago), environment (5 days ago), fit.py (14 minutes ago), pipeline.yaml (5 days ago), README.i... (5 days ago), README.... (5 days ago), requirements (5 days ago), and tasks.py (5 days ago).
- Code Editor:** In the top right, there is a code editor tab for `fit.py` which contains the following code:

```
0.98      50  
weighted avg      0.98      0.98  
0.98      50
```

```
[9]: plot.confusion_matrix(y_test, y_pred)
```

```
[9]: <AxesSubplot:title={'center':'Confusion matrix'}, xlabel='Predicted label', ylabel='True label'>
```

A confusion matrix plot titled "Confusion matrix" is displayed below the code. The x-axis is labeled "Predicted label" and the y-axis is labeled "True label", both with categories Class 0, Class 1, and Class 2. The matrix values are:

		Class 0	Class 1	Class 2
True label	Class 0	19	0	0
	Class 1	0	15	0
	Class 2	0	1	15

A color scale bar on the right indicates values from 0.0 (light yellow) to 17.5 (dark red).

```
[10]: with open(product['model'], 'wb') as f:  
    pickle.dump(clf, f)
```

```
[ ]:
```
- Kernel Status:** At the bottom, it shows "No Kernel" and "Mem: 201.97 / 8192...".

Formato: **nbformat**

Truco: ¡Cambiar el formato! ⁴

1. Mejor integracion con git

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

2. Mejor interoperabilidad con
otros ambientes de desarollo
(VSCode, Spyder, PyCharm)

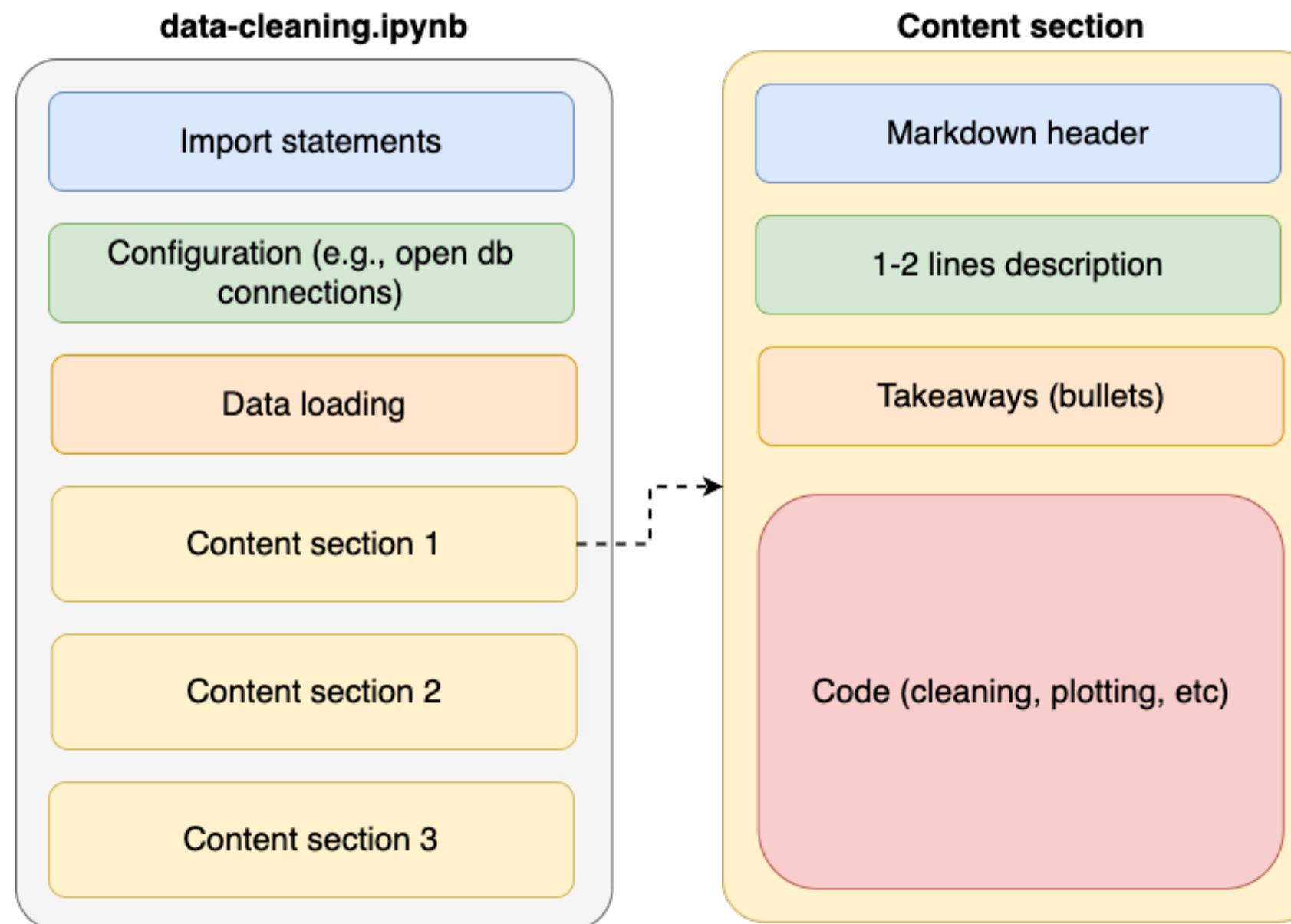
```
# %%  
# one cell  
df = pd.read_csv('my-data.csv')  
  
# %%  
# another cell  
df['new-column'] = df['column'] + 1
```

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
```

```
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
```

¡Sin efectos! 

Los cuadernos limpios declaran dependencias

1. Actualizaciones de paquetes
pueden romper tu proyecto

Después de instalar las
dependencies:

2. Hará más difícil correr el
cuaderno en el futuro

pip freeze > requirements.lock.txt

Para volver a crear el ambiente:

pip install -r requirements.lock.txt

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

Los cuadernos limpios siguen estándares⁶

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: `sourgeon clean nb.ipynb`

⁶ Paquete [Sourgeon](#) (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:

```
# 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

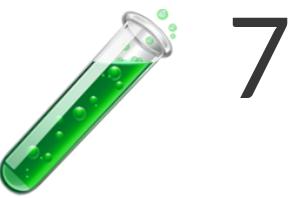
[3]: df = pd.DataFrame({"x": np.random.rand(10)})

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

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

	x	y
0	0.010910	1.010910
1	0.201756	1.201756
2	0.404869	1.404869

Parte II: Probando cuadernos (*testing*)



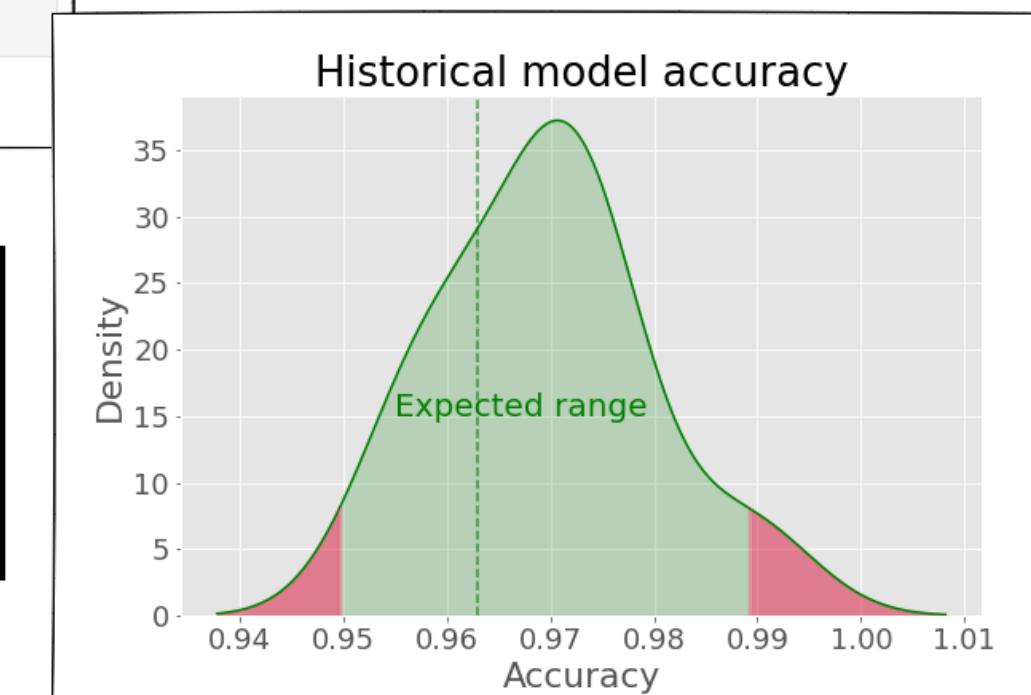
7

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

Probando cuadernos: Salidas de las celdas⁸

```
[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

```
from transformations import add_one

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

Lógica embebida ¹¹

```
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
```

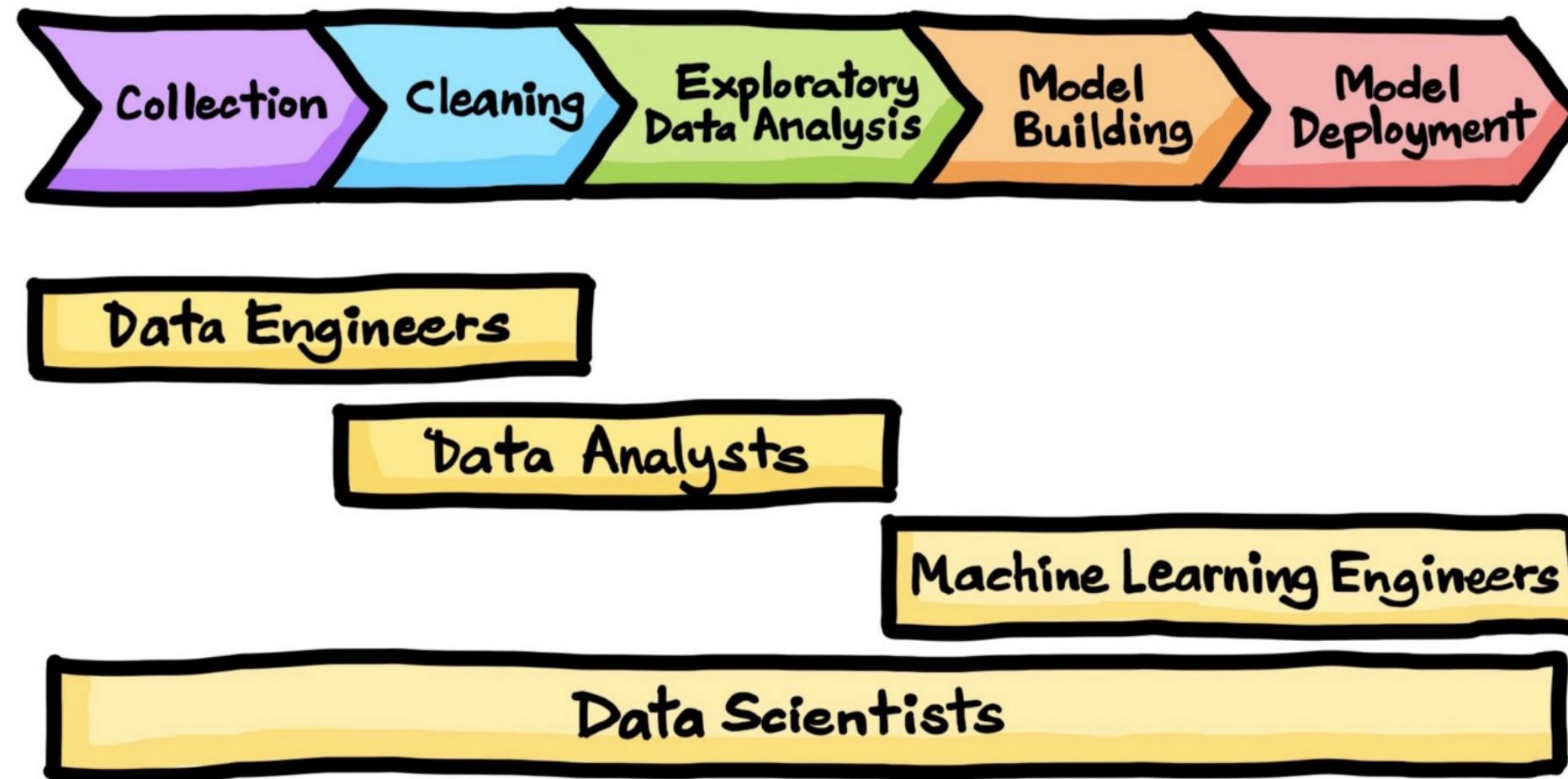
¹⁰ Paquete: [pytest](#)

¹¹ Paquete: [testbook](#)

Parte III: Construyendo *pipelines* de datos ¹²

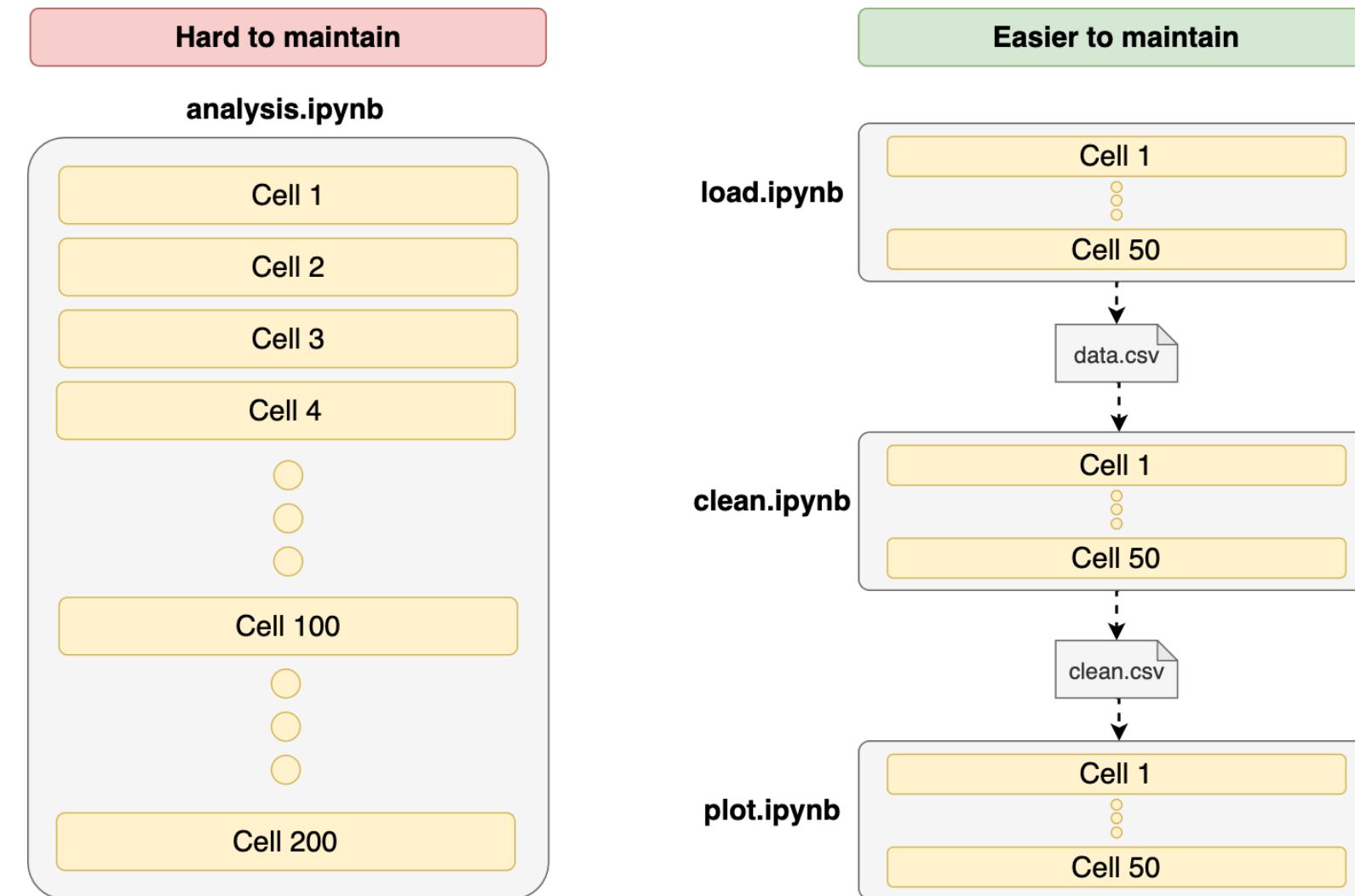
¹² Lectura recomendada: ploomber.io/blog/clean-pipelines

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)

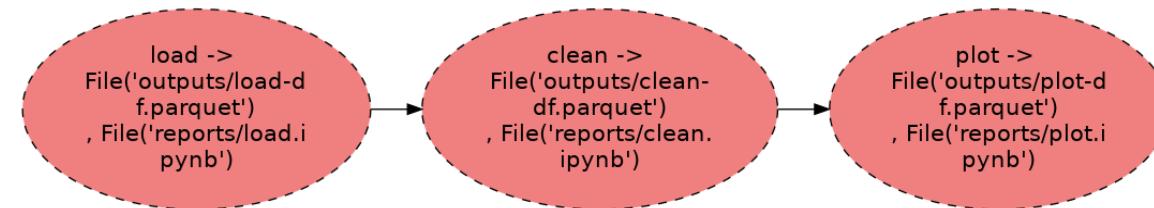
Construyendo *pipelines* de datos

```
# 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 pipelines¹⁵:



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.)