Cómo vamos nosotros a seguir buenas prácticas

Mathieu Kessler

Departamento de Matemática Aplicada y Estadística
Universidad Politécnica de Cartagena

Cumpliendo con una estructura clara de carpetas

En los preliminares, vimos que la estructura adecuada de la carpeta `ids` donde se situarán todos nuestros archivos es:

```
ids
    |__ data
    |__ doc
    |__ figures
    |__ results
    |__ src
```

donde

- `data`: todos los ficheros de datos para cargar
- `doc`: documentos de texto, documentación de interés
- `figures`: las gráficas generadas
- `results`: archivos generados por scripts
- `src`: los ficheros `.py` y los blocs de notas Jupyter `.ipynb`

Cumpliendo con una estructura clara de carpetas

Simplificamos un poco la estructura adecuada de la carpeta `ids` y nos quedamos con tres subcarpetas:

```
ids
    |__ data
    |__ doc
    |__ figures
    |__ results
    |__ src
```

donde

- `data`: todos los ficheros de datos para cargar
- ~~`doc`: documentos de texto, documentación de interés~~
- `figures`: las gráficas generadas
- ~~`results`: archivos generados por scripts~~
- `src`: los ficheros `.py` y los blocs de notas Jupyter `.ipynb`

Cumpliendo con una estructura clara de carpetas

Simplificamos un poco la estructura adecuada de la carpeta `ids` y nos quedamos con tres subcarpetas:

```
ids
    |__ data
    |__ figures
    |__ src
```

donde

- `data`: todos los ficheros de datos para cargar
- `figures`: las gráficas generadas
- `src`: los ficheros `.py` y los blocs de notas Jupyter `.ipynb`

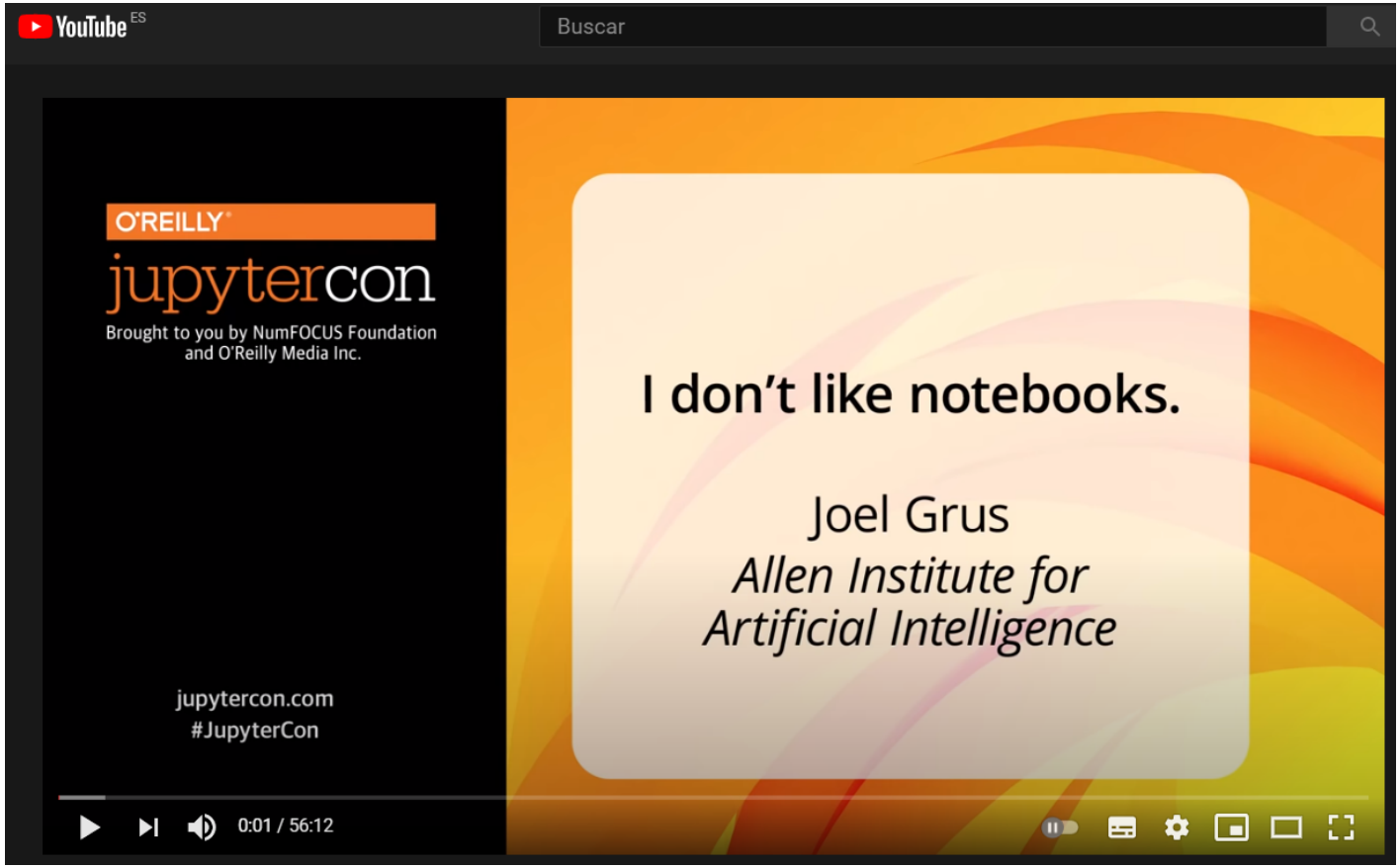
Usar blocs de notas puede facilitar la reproducibilidad

Los blocs de notas Jupyter

- A priori, permiten compartir documentos enriquecidos de explicación y descripción de los pasos del análisis.
- Cualquier debe poder re-ejecutar todas las celdas en orden y reconstruir el análisis.

Sin embargo, no todo el mundo está de acuerdo en que son una buena herramienta para un análisis reproducible.

Usar blocs de notas puede facilitar la reproducibilidad



Enlace Youtube

Usar blocs de notas puede facilitar la reproducibilidad

Enlace Youtube

Usar blocs de notas puede facilitar la reproducibilidad

- En este post, descargaron y analizaron 10000000 de Jupyter notebooks desde Github.
- Netflix ha creado su propio notebook llamado Polynote: https://polynote.org/, especializado para Scala.

Personalmente

- Utilizo blocs de notas para las clases y para la exploración preliminar de un problema.
- Para un análisis más sistemático, prefiero a continuación pasar a ficheros `.py`.

También cuidaremos nuestro estilo de programación

Para ello, la referencia es PEP 8.

Qué es un PEP?

Un PEP es un "Python Enhancement Proposal", un documento contribuido por desarrolladores reconocidos que recogen guias, convenciones, direcciones para la evolución de Python. Se pueden consultar en el PEP 0 – Index of Python Enhancement Proposals.

EL documento PEP 8 es "Style Guide For Python Code".

Guía de estilo

Procuraremos seguir las pautas aceptadas para mejor la legibilidad de nuestros scripts.

Seguiremos el documento PEP 8.

> *Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread.*
> *Hadley Wickham.*

Nombres de objetos

Se recomiendan diferentes convenciones según el tipo de objeto

PEP 8 distingue las convenciones para nombres de objetos según si se trata de nombres de módulos, clases, tipos de variable, variables, funciones etc...

- Los nombres de **variables y funciones** sólo deben componerse de **letras minúsculas, números, y guión bajo (_)**
- Deben ser consisos y tener sentido (lo más difícil!)

```
# Good
day_one
day_1
# Bad
first_day_of_the_month
DayOne
dayone
djm1
```

Espacios

- Ponemos espacio antes y después de todos los operadores (`=`, `+`, `-`, `<-`, etc.)
- Una excepción: no podemos comas antes ni después de "=" si es para el argumento de una función.
- Ponemos espacio después de las comas, pero nunca antes.

```
# Good
day_one
day_1
# Bad
first_day_of_the_month
DayOne
dayone
djm1
```

Espacios

- Ponemos espacios antes de “(“, excepto cuando corresponde a una función.

```
# Good
average = np.mean(feet / 12 + inches, axis=1)
# Bad
average=np.mean(feet/12+inches,axis=1)
average = np.mean(feet / 12 + inches, axis = 1)
```

Indentaciones y líneas

El número máximo de caracteres por línea debe ser 79!

- Usaremos 4 espacios para marcar un bloque indentado. (el defecto de VS Code)
- PE8 admite también otras opciones pero pondremos cada argumento de una función en una linea con indentación:

```
# Good
average = np.mean(feet / 12 + inches, axis=1)
# Bad
average=np.mean(feet/12+inches,axis=1)
average = np.mean(feet / 12 + inches, axis = 1)
```

PEP 8 contiene más detalles sobre indentación: enlace. Merece la pena echarle un vistazo.

Herramientas para mejorar nuestro código

"Formatter and linter"

Existen distintas herramientas para mejorar nuestro código:

- Los formateadores corrigen el código para implementar la guía de estilo de Python, descrita en PEP 8. Los más populares: black, autopep8.
- Los "linters" hacen un primer análisis de código, por ejemplo detectan variables que se definen pero no se vuelven a usar, o paquetes que se importan pero no se llegan a usar. Un linter popular en Python: Flake8.

En Visual Studio Code

- Permite aplicar formateadores como black, autopep8 a documentos, incluso de manera automática cada vez que se guarden.
- Incluye un linter desarrollado por Microsoft llamado "Pylance".