

ACÀMICA

¡Bienvenidos/as a Data Science!



Agenda

Proyecto 1: Análisis Exploratorio de Datos

Repaso: Hasta ahora

Explicación y Hands-On: Funciones

Break

Explicación: Transformación de Datos

Hands-on: Transformación de Datos con Pandas

Cierre



Proyecto 1: Análisis Exploratorio de Datos (EDA)



Análisis Exploratorio de Datos (EDA)

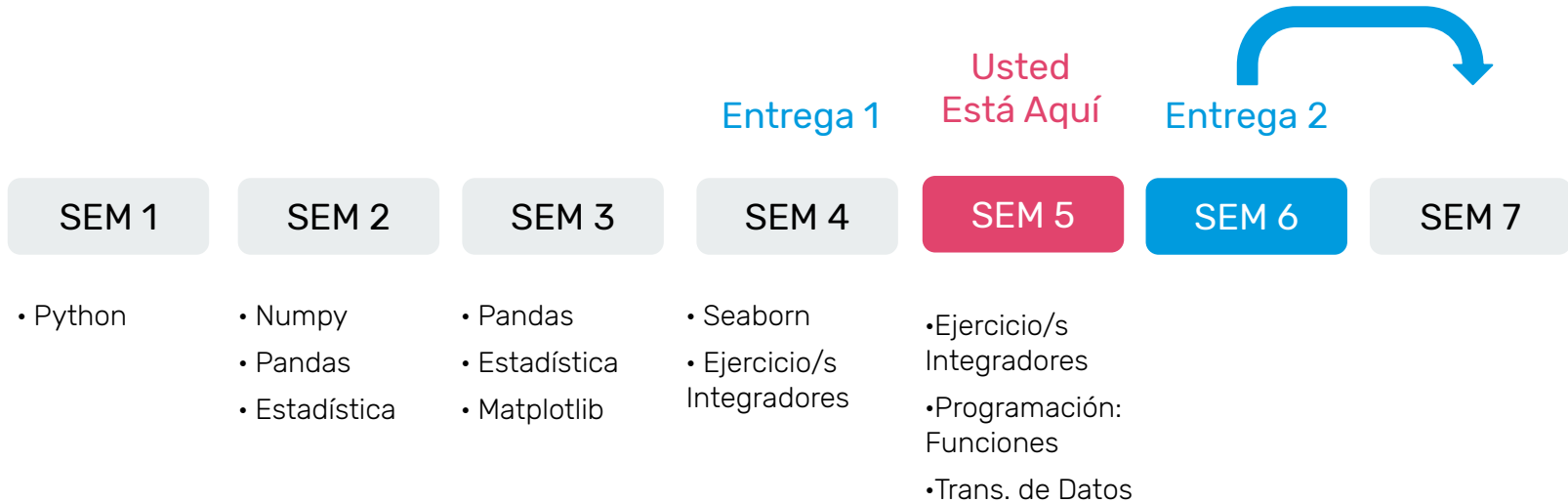
fase	ADQUISICIÓN Y EXPLORACIÓN		MODELADO				DEPLOY
	Exploración de datos	Feature Engineering	Regresión	Optimización de parámetros	Procesam. del lenguaje natural	Sistema de recomendación	Publicación de modelos
entrega	SEM 1	SEM 5	SEM 8	SEM 11	SEM 13	SEM 18	SEM 22
	SEM 2	SEM 6	SEM 9	SEM 12	SEM 14	SEM 19	SEM 23
	SEM 3	SEM 7	SEM 10		SEM 15	SEM 20	SEM 24
	SEM 4				SEM 16	SEM 21	
tiempo					SEM 17		



Proyecto EDA: Hoja de ruta



Proyecto EDA: Hoja de ruta



Proyecto EDA: Hoja de ruta



Proyecto EDA: Hoja de ruta

Entrega 1

SEM 1 - 4

- Python
- Numpy
- Pandas
- Visualización de datos: Matplotlib y Seaborn
- Estadística

Usted
Está Aquí

SEM 5

- Ejercicio/s Integradores
- Programación: Funciones
- Transformación de Datos

SEM 6

- Programación: Clases
- Transformación de Datos con Pandas

Entrega 2

SEM 7

- Transformación de Datos con Scikit-Learn
- Outliers

SEM 8

- ¡Arrancamos con Machine Learning!



Repaso



Hasta ahora

- ✓ Vimos un poco de programación con Python en un entorno particular, Jupyter
- ✓ Aprendimos a trabajar de forma eficiente con números y la computadora usando Numpy
- ✓ Aprendimos cómo abrir y operar con un Dataset usando Pandas
- ✓ Creamos (lindos) gráficos con Matplotlib y Seaborn
- ✓ Además, repasamos algunos conceptos de estadística: variables aleatorias, distribuciones, correlación, etc.

Vamos a seguir profundizando en herramientas (programación y librerías) y en estadística a lo largo de las clases.

Programación: Funciones



Funciones


Una función es un bloque de código que sólo *corre* cuando es *llamado*.

```
[ ]: def unaFuncion(numero):  
    if numero%2 == 0:  
        print('Es par')  
    else:  
        print('Es impar')
```

Funciones

Una función es un bloque de código que sólo *corre* cuando es *llamado*.

Le dice a Python que
vamos a crear una función



```
[ ]: def unaFuncion(numero):  
    if numero%2 == 0:  
        print('Es par')  
    else:  
        print('Es impar')
```

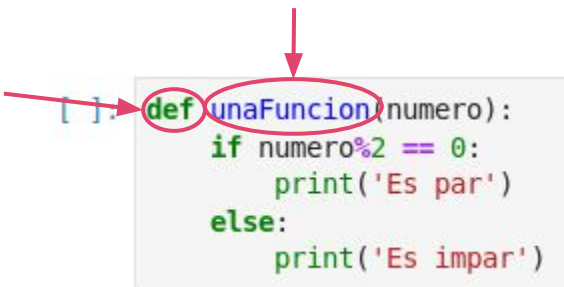
Funciones

Una función es un bloque de código que sólo *corre* cuando es *llamado*.

Le dice a Python que vamos a crear una función

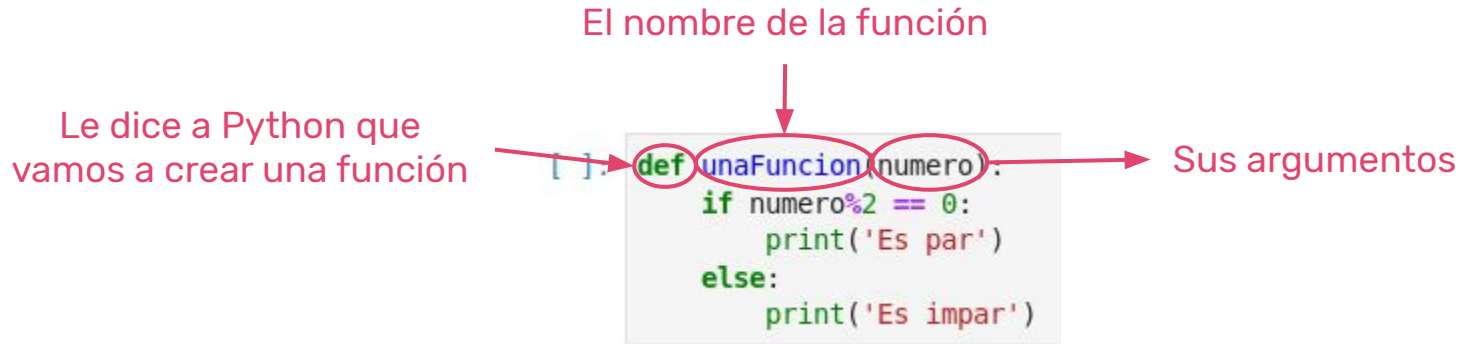
El nombre de la función

```
[ ]: def unaFuncion(numero):  
    if numero%2 == 0:  
        print('Es par')  
    else:  
        print('Es impar')
```



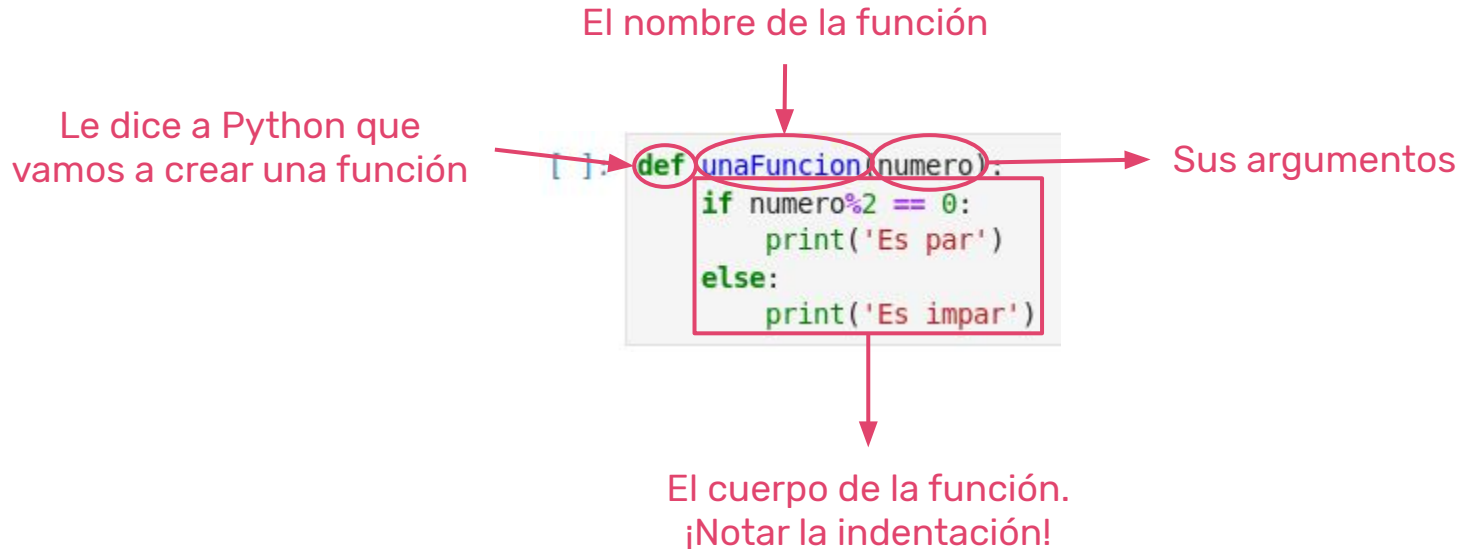
Funciones

Una función es un bloque de código que sólo *corre* cuando es *llamado*.



Funciones

Una función es un bloque de código que sólo *corre* cuando es *llamado*.



Funciones

Las funciones pueden tener argumentos *por default*.

```
[7]: def division(dividendo, divisor = 2):  
      print(dividendo/divisor)
```

Funciones

Las funciones pueden tener argumentos *por default*.

```
[7]: def division(dividendo, divisor = 2):  
      print(dividendo/divisor)
```

Salvo que le digamos lo contrario, el valor default de divisor es 2.

Funciones

Las funciones pueden tener argumentos *por default*.

```
[7]: def division(dividendo, divisor = 2):  
      print(dividendo/divisor)
```

```
[8]: division(9)  
      division(9, 3)  
      division(dividendo = 9, divisor = 3)  
      division(9, divisor = 3)
```

```
4.5  
3.0  
3.0  
3.0
```

Salvo que le digamos lo contrario, el valor default de divisor es 2.

Funciones

Las funciones pueden “devolver” resultados.

```
[9]: def division(dividendo, divisor = 2):  
      variable_auxiliar = dividendo/divisor  
      return variable_auxiliar
```

Funciones

Las funciones pueden “devolver” resultados.

```
[9]: def division(dividendo, divisor = 2):  
      variable_auxiliar = dividendo/divisor  
      return variable_auxiliar
```

Con **return** le decimos qué queremos que devuelva. Puede ser más de un objeto.

Funciones

Las funciones pueden “devolver” resultados.

```
[9]: def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
[10]: resultado_division = division(9,3)  
[11]: print(resultado_division)  
3.0
```

Con **return** le decimos qué queremos que devuelva. Puede ser más de un objeto.

Funciones

Las funciones pueden “devolver” resultados.

Asignamos el resultado
de llamar a la función a
una nueva variable

```
[9]: def division(dividendo, divisor = 2):  
      variable_auxiliar = dividendo/divisor  
      return variable_auxiliar  
[10]: resultado_division = division(9,3)  
[11]: print(resultado_division)  
3.0
```

Con **return** le decimos
qué queremos que
devuelva. Puede ser
más de un objeto.

¿Cuál es la diferencia entre estos bloques de código?

1

```
def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

25.0

```
-----  
NameError  
<ipython-input-15-28655831cd39> in <module>  
      3     return variable_auxiliar  
      4 print(division(50))  
----> 5 print(divisor)
```

NameError: name 'divisor' is not defined

2

```
divisor = 5  
def division(dividendo):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

10.0
5

3

```
divisor = 5  
def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

25.0
5

¿Cuál es la diferencia entre estos bloques de código?

1

```
def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

25.0

```
-----  
NameError  
<ipython-input-15-28655831cd39> in <module>  
      3     return variable_auxiliar  
      4 print(division(50))  
----> 5 print(divisor)
```

NameError: name 'divisor' is not defined

2

```
divisor = 5  
def division(dividendo):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

10.0
5

3

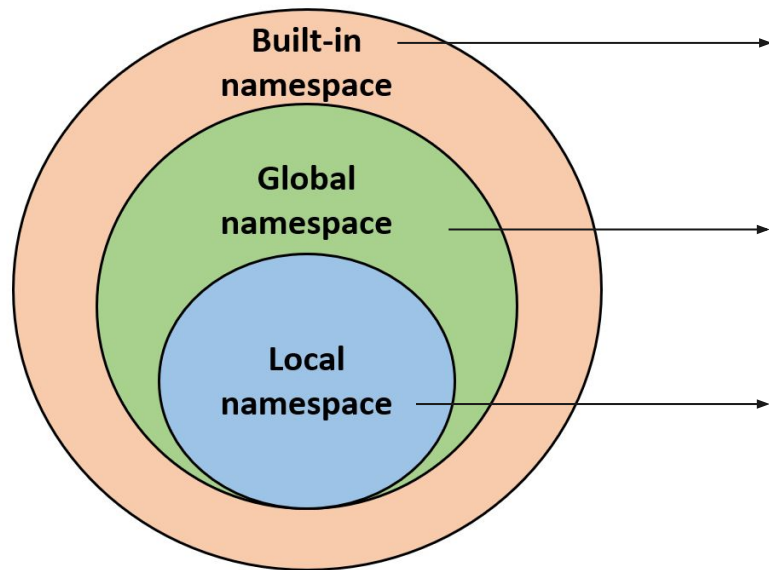
```
divisor = 5  
def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

25.0
5

The Zen of Python

Namespaces are one honking great idea -- let's do more of those!

Namespaces (informalmente)



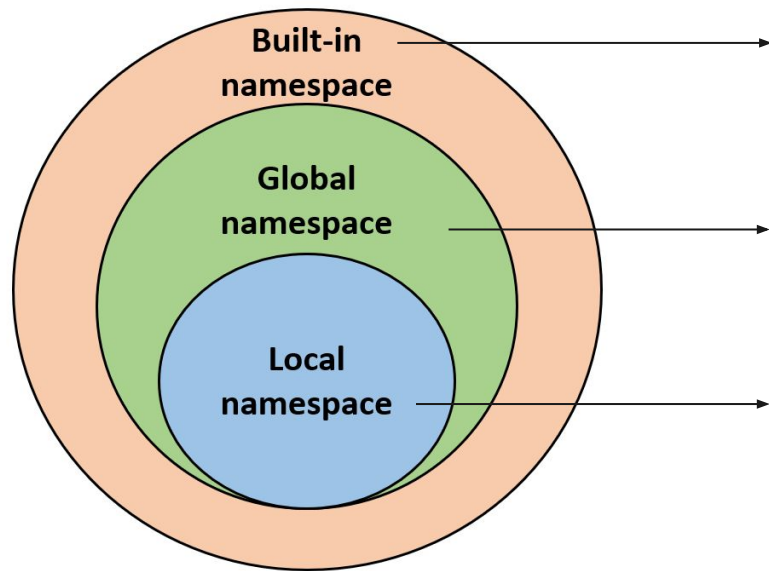
Type of Namespaces

Todas los “nombres” que quedan definidos cuando arrancamos un kernel de Python. Ejemplo: `print()`.

Cuando importamos módulos o trabajamos en un notebook, estamos trabajando acá.

Se crea cuando llamamos a una función y se destruye cuando termina el llamado.

Namespaces (informalmente)



Type of Namespaces

Todas los “nombres” que quedan definidos cuando arrancamos un kernel de Python. Ejemplo: `print()`.

Cuando importamos módulos o trabajamos en un notebook, estamos trabajando acá.

Se crea cuando llamamos a una función y se destruye cuando termina el llamado.

En general, es fácil ir de afuera hacia adentro, pero no de adentro hacia afuera.

¿Cuál es la diferencia entre estos bloques de código?

1

```
def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

25.0

```
-----  
NameError  
<ipython-input-15-28655831cd39> in <module>  
      3     return variable_auxiliar  
      4 print(division(50))  
----> 5 print(divisor)
```

NameError: name 'divisor' is not defined

Acá **divisor** es una variable local dentro de la función, deja de existir cuando termina el llamado a la función.

2

```
divisor = 5  
def division(dividendo):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

10.0

5

Acá **divisor** es una variable global. En el llamado de la función, como no encuentra una variable local llamada **divisor**, busca una variable global llamada **divisor**.

3

```
divisor = 5  
def division(dividendo, divisor = 2):  
    variable_auxiliar = dividendo/divisor  
    return variable_auxiliar  
print(division(50))  
print(divisor)
```

25.0

5

Acá existe **divisor** como variable global y **divisor** como variable local. El llamado a la función usa siempre primero la variable local.

Funciones Lambda

Una función Lambda es una forma conveniente de crear una función **en una sola línea**. También se las conoce como funciones anónimas, ya que no tienen nombre, sino que se asignan a una variable.

```
[18]: lambda_division = lambda x,y: x/y  
lambda_division(80,10)
```

```
[18]: 8.0
```

Algunas características:

1. Pueden tener cualquier cantidad de argumentos, pero solo una expresión
2. No necesitan un *return*
3. Muy cómodas para crear funciones rápido
4. Se combinan muy bien con funciones como `map()`, `filter()`, `apply()`, `applymap()`, etc.

Hands-on training



DS_Clase_10_Funciones.ipynb



A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup is placed on a matching white saucer. In the background, a white napkin and a silver spoon are visible, though they are out of focus. The overall lighting is soft and even, highlighting the textures of the coffee and the smooth surface of the cup.

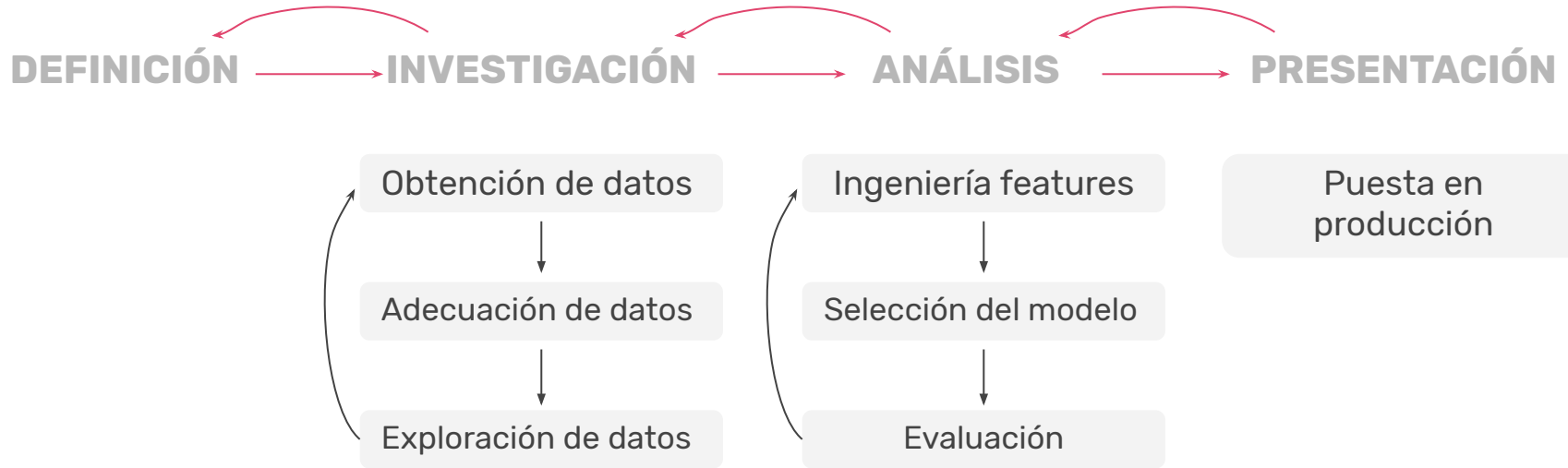
¡BREAK!



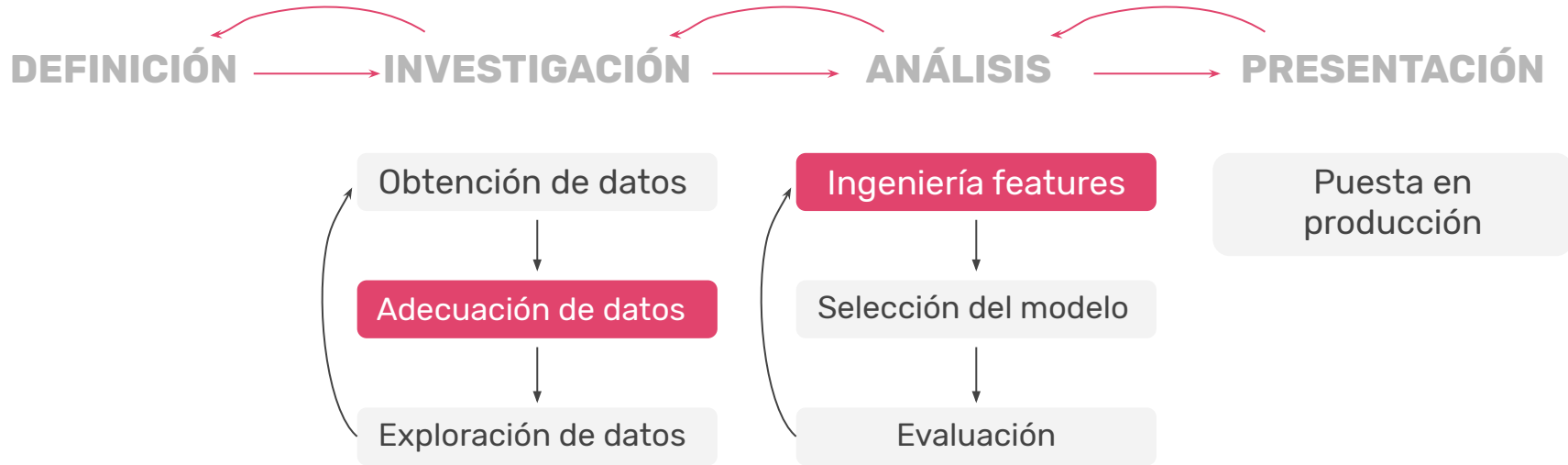
¿Qué es Data Science?



Es un proceso iterativo

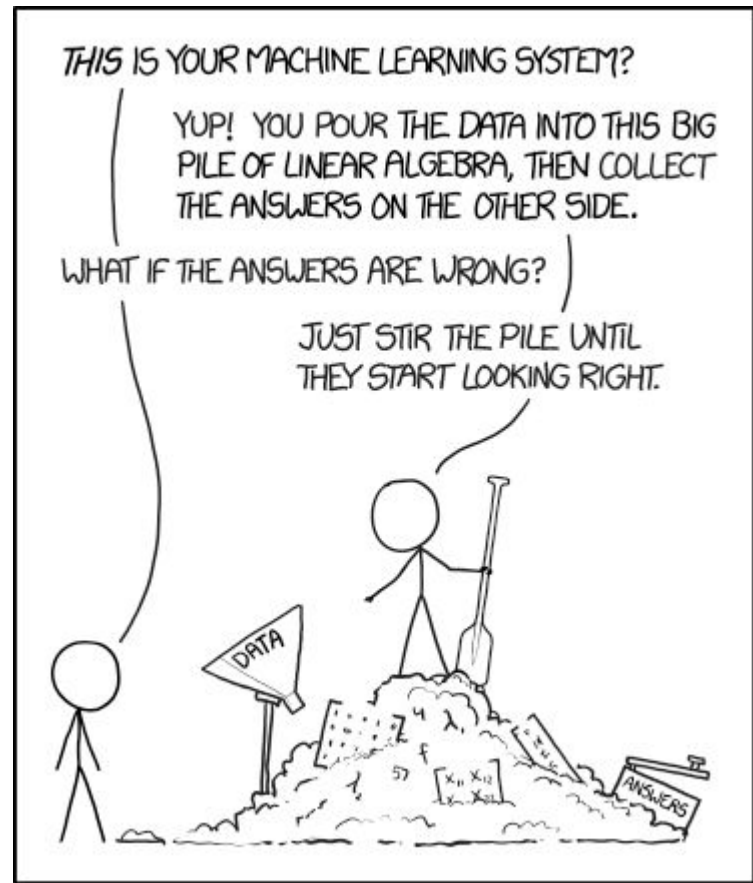


Es un proceso iterativo



"At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used."

—Prof. Pedro Domingos



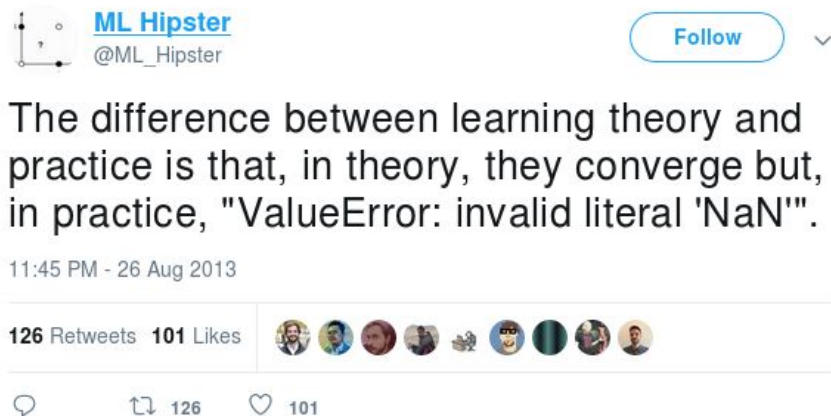
<https://xkcd.com/1838/>

Transformación de Datos



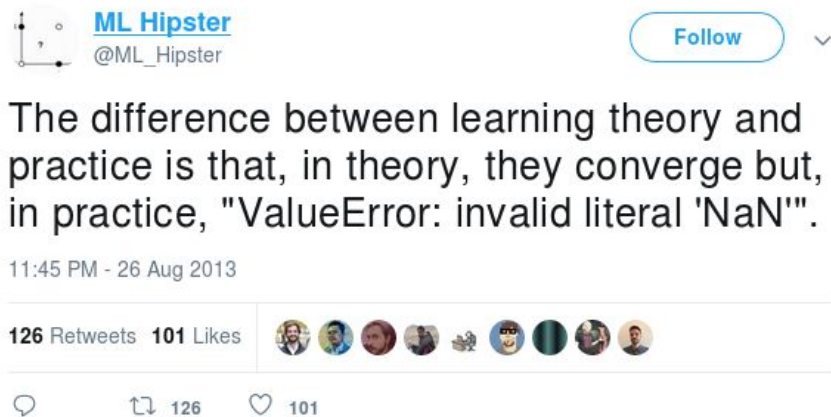
Ya vimos algo de Transformación de Datos: Imputación de valores faltantes.

Ya vimos algo de Transformación de Datos: Imputación de valores faltantes.



https://twitter.com/ML_Hipster/status/372248401951215616

Ya vimos algo de Transformación de Datos: Imputación de valores faltantes.



https://twitter.com/ML_Hipster/status/372248401951215616

**¡La computadora, en el fondo,
sólo entiende de números!**

Pero no todos los features vienen en “números”.

Por ejemplo, en el dataset de críticas de vinos tenemos:

- País, Provincia y Región de Origen
- Variedad
- Bodega
- Etc.

Supongamos que queremos clasificar los vinos según algunos atributos.

¿Cuáles creen que serían útiles?

¿Cuáles creen que faltan?

Supongamos que queremos clasificar los vinos según algunos atributos.

¿Cuáles creen que serían útiles?

¿Cuáles creen que faltan?

¿Y si queremos predecir el precio? ¿O si es bueno (puntaje alto) o malo (puntaje bajo)?

La pregunta que querramos responder nos va a indicar cómo tenemos que trabajar con nuestro dataset.

Pero, en general, hay algunas cosas que **no pueden faltar:**

- Conversión de features.
- Tratamiento de datos faltantes.
- Selección y combinación de variables.
- Y más.

Tipos de datos



Variables numéricas

Son aquellas variables que se miden o se cuentan (discretas o continuas).

- Hay una relación de orden entre ellas
- Se pueden sumar (en algunas circunstancias)

Ejemplos:

- Edad, Altura y Peso de una persona
- Puntaje, precio de un vino
- Valor de un pasaje
- Etc.

Tratamiento

En general, ya vienen en un formato “cómodo” para trabajar, pero a veces queremos agruparlas según grupos o rangos.

Ejemplo: agrupar edades en rangos (bebés, niños, adolescentes, adultos, ancianos)

→ **Discretización y Binning**

Variables ordinales

Sus posibles valores son categorías, pero hay una relación de orden.

¡Notar que no se pueden sumar!

Ejemplos:

- Tamaño de una prenda de ropa: XS, S, M, L, XL
- Tipo de Nafta por octanaje: 95, 98, más de 98.
- Rangos etarios: bebé, niño/a, adolescente, adulto/a, anciano/a

Tratamiento

Podemos hacer una asignación a números enteros manteniendo el orden:

$S \rightarrow 0$

$M \rightarrow 1$

$L \rightarrow 2$

Pero, ¡cuidado!, recordar que no se pueden sumar.

Variables nominales

Sus posibles valores pertenecen a una de varias categorías.

- Las categorías no siguen una relación de orden
- Ninguna es mayor que otra

Ejemplos:

- Nacionalidad
- Tipo de vino
- Color de una prenda de ropa
- Género: femenino, masculino, no binario, etc.

Tratamiento

- Llevar a variables dummies/One-Hot Encoding.
- Hay que tener cuidado porque puede hacer que nuestro dataset crezca mucho.

Resumiendo

Conversión de variables: Los modelos sólo entienden de números.
¿Si los atributos no son números?

Tipos de variables a tratar:

- Numéricas: edad, altura, puntaje.
- Ordinales: tamaño de una prenda de ropa.
- Categóricas/nominales: nacionalidad, color de una prenda de ropa.
- Y más...

Tratamiento

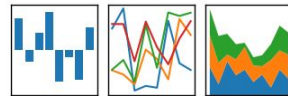
- Discretización/binning
- "Labelización"¹
- Variables dummies/One-Hot encoding
- Y más...

¹ Tal vez inventamos una palabra

Transformación de datos con Pandas

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



¿Qué nos cuentan los videos?

- **Discretización y binning:**
 - ¿Qué funciones podemos usar?
 - ¿Cuáles son sus argumentos?
 - ¿En qué situaciones será conveniente?
- **¿Qué hace la función map()?**
 - ¿Qué toma como argumento?
 - ¿Sobre qué objeto opera?
- **¿Qué es una *variable dummy*?**
 - ¿Cómo funciona get_dummies() de Pandas?

Hands-on training

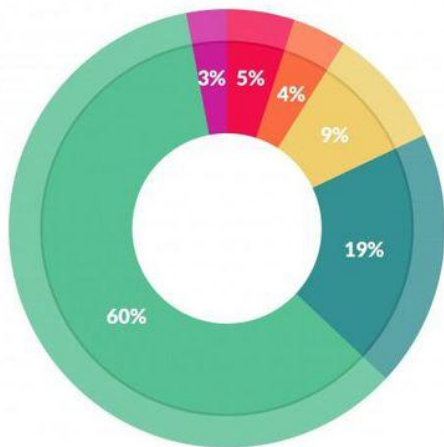


DS_Clase_10_TDD_Pandas.ipynb



Sabías que...

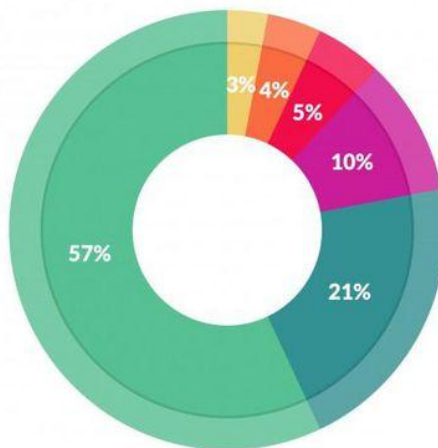




What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

pero...



What's the least enjoyable part of data science?

- Building training sets: 10%
- Cleaning and organizing data: 57%
- Collecting data sets: 21%
- Mining data for patterns: 3%
- Refining algorithms: 4%
- Other: 5%

Proyecto #1: Adquisición y Exploración de datos





En la clase 15 haremos una puesta en común, en la cual pediremos que presenten sus trabajos.





No se preocupen.

**¡No hace falta que los
tengan terminados!**

(aunque lo recomendamos altamente)



Lo que les pediremos es:

- Que muestren lo que tengan hecho hasta ahora
- Que cuenten su proceso de trabajo
- Que cuenten lo que más les costó y cómo lo superaron (o no)
- Que pidan ayuda a la clase si lo necesitan
- Que cuenten cómo piensan encarar los próximos pasos (¡pueden pedir ayuda acá también!)





Evaluaciones: IMPORTANTE

Siguiendo los pasos en la plataforma Acámica, deberán entregar el proyecto, que será evaluado por un evaluador/a profesional.

Los evaluadores/as les darán una devolución sobre su trabajo a través de la plataforma.

Podrán preguntarles por Slack si tienen dudas sobre una corrección que recibieron, **pero NO podrán pedirles ayuda sobre guía y mentoreo técnico para concluir sus proyectos (para eso estamos los/as mentores/as 😊)**.

Recursos



Recursos

Funciones y Namespaces:

- <https://www.geeksforgeeks.org/functions-in-python/>
- <https://www.geeksforgeeks.org/namespaces-and-scope-in-python/>
- Algo de namespaces y clases, que veremos la clase siguiente: <https://docs.python.org/3/tutorial/classes.html>

Transformación de datos con Pandas

<https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114>



Para la próxima

1. Ver los videos de la plataforma “Transformación de Datos”
2. Completar los notebooks de hoy
3. ¡Hay poca tarea! ¡Aprovechen a terminar la Entrega 01!