

# Introducción a la ciencia de datos



Ajuntament de  
Barcelona



Barcelona  
Activa

# Índice

<b>1 INTRODUCCIÓN A LA CIENCIA DE DATOS Y AL ENTORNO DE TRABAJO DE JUPYTER NOTEBOOK.....</b>	<b>3</b>
1.1 ¿QUÉ ES LA CIENCIA DE DATOS Y QUÉ APLICACIONES TIENE?.....	3
1.2 CONCEPTOS BÁSICOS DE LA CIENCIA DE DATOS.....	4
1.3 TRABAJAR CON JUPYTER NOTEBOOK .....	6
1.4 PROFUNDIZAR EN JUPYTER NOTEBOOK.....	9
1.5 IDEAS CLAVE: INTRODUCCIÓN A LA CIENCIA DE DATOS Y AL ENTORNO DE TRABAJO DE JUPYTER NOTEBOOK .....	11
<b>2 EXPLORACIÓN INICIAL EN LOS DATOS .....</b>	<b>11</b>
2.1 EXPLORACIÓN BÁSICA DE LOS DATOS .....	12
2.2 EJEMPLO DE EXPLICACIÓN DE DATOS .....	14
2.3 MODELOS PREDICTIVOS.....	16
2.4 MODELOS DE REGRESIÓN .....	18
2.5 EJEMPLO DE UN MODELO DE REGRESIÓN .....	19
2.6 MODELOS DE CLASIFICACIÓN.....	22
2.7 EJEMPLO DE UN MODELO DE CLASIFICACIÓN .....	23
2.8 IDEAS CLAVE: EXPLORACIÓN INICIAL A LOS DATOS.....	27
<b>3 VISUALIZACIÓN DE DATOS CON MATPLOTLIB .....</b>	<b>27</b>
3.1 GRÁFICAS LINEALES Y DE BARRAS .....	27
3.2 SCATTERPLOTS .....	33
3.3 EJEMPLO PARA GENERAR LOS GRÁFICOS APRENDIDOS .....	39
3.4 IDEAS CLAVE: VISUALIZACIÓN DE DATOS CON MATPLOTLIB .....	41
<b>4 ANÁLISIS DE DATOS CON PANDA.....</b>	<b>42</b>
4.1 CREAR, LEER Y ESCRIBIR UNA TABLA DE DATOS.....	42
4.2 INDEXAR, SELECCIONAR Y FILTRAR.....	46
4.3 ORDENAR Y AGRUPAR DATOS NO DISPONIBLES .....	51
4.4 REPASO DEL TEMARIO CON EJERCICIO PRÁCTICO .....	56
4.5 IDEAS CLAVE: ANÁLISIS DE DATOS CON PANDA.....	59

# 1 INTRODUCCIÓN A LA CIENCIA DE DATOS Y AL ENTORNO DE TRABAJO DE JUPYTER NOTEBOOK

En este primer módulo nos acercaremos a qué es la ciencia de datos, sus fundamentos, las herramientas de trabajo más habituales, qué aplicaciones tiene y qué campos profesionales y de investigación la usan. También abordaremos qué papel juega el lenguaje de programación Python.

Posteriormente, veremos qué es y cómo funciona el entorno de programación Jupyter Notebook y para trabajar, primero, con las funciones más sencillas; posteriormente, haremos un paso adelante con otros recursos más avanzados. A partir de aquí, ya podremos desarrollar y escribir código con Python y disponer de una orientación clara a la ciencia de datos.

Es importante prestar mucha atención durante este primer módulo, ya que conseguiremos las habilidades básicas para hacer frente a todo el curso y seguir aprendiendo. Todo el contenido es importante, desde el inicio hasta el final. ¡La ciencia de datos nos espera!

## 1.1 ¿QUÉ ES LA CIENCIA DE DATOS Y QUÉ APLICACIONES TIENE?

¡Hola!

Quizás te preguntarás qué es esto de la ciencia de datos y por qué este concepto está, prácticamente, en todas partes. En este video, procuraremos responder a esta gran pregunta. ¡Vamos!

La ciencia de datos implica el estudio de los datos en sus diferentes formas para extraer conocimiento o información. Esta ciencia usa las matemáticas, la estadística y varias disciplinas informáticas con el objetivo de encontrar patrones a la generación de datos que tanto los humanos como la naturaleza generan continuamente. Estos patrones son la clave para que los datos tengan un valor aplicable.

La ciencia de datos se usa en el mundo empresarial, en la gestión de los recursos públicos, en la industria y también en la investigación. A medida que crece la generación de datos, sobre todo por el aumento de la actividad humana y la mayor capacidad de medida y detección de elementos susceptibles de convertirse en datos, también crece la necesidad de personal científico y técnico en esta ciencia. El personal especializado en este campo tiene que tener habilidades analíticas de aprendizaje automático y en minería de datos, así como dominar lenguajes de programación para diseñar algoritmos. Además, el científico o científica de datos tendrá que saber interpretar los resultados y transmitirlos al personal no técnico, para activar los cambios o las medidas que el valor de los datos indique. Sí, la ciencia de datos despierta mucha demanda. Si te gusta y te esfuerzas, puede ser una gran oportunidad desde el punto de vista profesional. La ciencia de datos crecerá y tú puedes hacerlo con ella.

Descubrimos ahora, por ejemplo, cómo puede ayudar la ciencia de datos a una empresa. Lo más importante en una organización es saber anticiparse a los cambios y tomar decisiones acertadas. Esta toma de decisiones puede ser guiada por la intuición o por el conocimiento generado gracias a la gestión de los datos. Si sabemos qué patrones de compra sigue la clientela de nuestro sector, podremos, por ejemplo, diseñar nuevos productos que encajen con sus necesidades. También podremos minimizar los volúmenes de mercancías almacenadas en *stock*, si sabemos cuándo se realizan estas compras y, de este modo, gestionaremos mejor nuestra tesorería. La gestión de la información en forma de datos reducirá el riesgo de cometer errores en la toma de decisiones.

¿Y cómo podemos hacer que la gestión de esta información, que es cada vez mayor, pueda generar valor y conocimiento aplicable de una manera más rápida? Aquí entra en juego el aprendizaje automático, del que seguramente habrás oído a hablar. El aprendizaje automático es la capacidad de una máquina de mejorar sus pronósticos, localizando patrones con un mayor grado de certeza, a medida que va gestionando más datos. Esta capacidad de las máquinas de ser cada vez más esmeradas en sus resultados tiene una gran aplicación en el reconocimiento de imágenes y voz, en la conducción autónoma, etc.

Dentro de este contexto, la inteligencia artificial es la habilidad de una máquina de poder evolucionar y mejorar en una serie de funciones y tareas sin que hubiera estado específicamente programada para ejecutarlas del mismo modo que lo hace. El origen de este nuevo campo tiene las raíces en la ciencia de datos, pero va mucho más allá y no está al alcance de este curso, a pesar de que es importante que seamos conscientes de su existencia y qué papel puede jugar en un futuro próximo.

¡Sigamos!

## 1.2 CONCEPTOS BÁSICOS DE LA CIENCIA DE DATOS

Cualquier estudio o ciencia tiene un proceso a seguir y la gestión de los datos no es una excepción. A continuación se presentan los pasos que hay que seguir para realizar una buena gestión de los datos.

- **Objetivo.** Lo primero es establecer un objetivo. Es decir, ¿por qué queremos estos datos y qué finalidad tendrá el conocimiento adquirido?
- **Obtención de los datos.** Necesitamos acceder y recopilar los datos con los que queremos trabajar. Estos datos seguramente llegarán de manera desordenada.
- **Preparar los datos.** Este paso se basa en ordenar los datos y depurar las interferencias antes de empezar a diseñar modelos para encontrar patrones.
- **Exploración de los datos.** Fase centrada en encontrar patrones, correlaciones y desviaciones que puedan convertir los datos en un conocimiento valioso y aplicable.
- **Construcción de modelos.** Esta etapa se basa en generar predicciones en función del modelo escogido. En este caso, podremos aplicar el conocimiento automático para ir mejorando el modelo con la acumulación de más datos.

- **Presentación de resultados y análisis.** Este punto consistirá en hacer públicos los resultados a otras personas, para llevar a cabo las nuevas acciones o los cambios que los modelos predictivos nos indiquen.

Hay que decir que la mayoría a veces este proceso no es lineal, sino cíclico. La generación de datos es continua y hay que revisar todos los pasos continuamente para mejorar la eficacia y la eficiencia de las predicciones (que comportará tomar mejores decisiones).

La ciencia de datos ofrece, básicamente, dos tipos de modelos predictivos: el de regresión y el de clasificación. Veámoslos:

El primero (modelo de regresión) tiene por objetivo la predicción de un número, que depende de la relación que pueda existir respecto a una o más variables. Este modelo predictivo se basa en los coeficientes de correlación entre las variables independientes y la dependiente, de la que queremos saber el valor. Los datos que puede necesitar este modelo cumplen los siguientes requisitos:

- Las variables se tienen que poder medir de manera continua. El tiempo, las ventas o el peso corresponden a este tipo de medidas.
- Las observaciones entre diferentes variables no tienen que tener interferencias entre ellas.
- Los datos no tendrían que tener valores atípicos. Un valor atípico es aquel que queda muy alejado del resto y podría ser debido, probablemente, a un error de medida. En ciertos casos podría no haber ningún error y que el valor del dato fuera solo una excepcionalidad.
- La varianza no tiene que cambiar a lo largo de la línea predictiva. Se puede ver la definición de varianza en: <https://es.wikipedia.org/wiki/Varianza>.
- Los errores a lo largo de la línea de predicción siguen una distribución normal. En el enlace siguiente se puede profundizar en el concepto de distribución normal: [https://es.wikipedia.org/wiki/Distribuci%C3%B3n\\_normal](https://es.wikipedia.org/wiki/Distribuci%C3%B3n_normal).

El modelo de clasificación tiene por objetivo predecir si una opción se llevará a cabo o no. Esta opción puede tener solo dos posibilidades diferentes, lo que se denomina *clasificación binaria*, o más de dos, que, en este caso, se denomina *clasificación multinomial*. Un modelo binario, por ejemplo, por tener las opciones de *verdad o falso*, *sí o no*, etc. El modelo multinomial, en cambio, tiene más: por ejemplo, podríamos incluir las opciones *Grupo A, B o C*, o *Grados de satisfacción desde Muy satisfecho, Poco Satisfecho, Nada satisfecho*, etc.

Gracias al modelo de clasificación, podremos saber, por ejemplo, si un accidente se producirá o no (dos posibilidades) o si un producto determinado será comprado por personas jóvenes, adultas o de edad avanzada (más de dos posibilidades).

Para generar un modelo capaz de hacer esta predicción, necesitaremos encontrar correlaciones con otras variables independientes. Por eso, tendremos que tener en cuenta estos elementos:

- Las variables independientes tienen que ser válidas.
- Tenemos que evitar los datos continuos, como la temperatura, el tiempo, etc.

- Tenemos que procurar no usar variables que estén estrechamente relacionadas entre ellas.

Para entender los próximos capítulos, es importante que nos familiaricemos con los términos siguientes:

- **Base de datos:** espacio digital destinado a almacenar información al que se puede acceder, siempre y cuando se tengan las credenciales y los permisos necesarios para leer e importar los datos que se encuentran.
- **Overfitting:** es el efecto de proveer demasiada información sobre un modelo del que ya se conoce el resultado deseado. Este efecto hace que el modelo quede demasiado definido respecto a una entrada de datos y que no sea capaz de generalizar resultados en otras situaciones.
- **Correlación:** mide cómo están relacionados directamente los cambios de una variable respecto a otra.
- **Mediana:** en una colección ordenada de datos es el valor que se encuentra justo en la posición central.
- **Normalización:** consiste en ajustar los valores medidos en escalas diferentes sobre una misma escala.
- **Valor atípico:** es aquel que está lejos respecto al resto del grupo. El valor atípico se debe a causas excepcionales o, a veces, sencillamente a un error.
- **Minería de datos:** proceso que consiste en extraer datos de una fuente determinada para ser posteriormente examinados. Incluye desde la limpieza, organización y depuración de datos hasta la búsqueda de patrones y relaciones significativas.
- **Clustering:** consiste en recopilar y agrupar un conjunto de puntos suficientemente similares o próximos.
- **Web scraping:** proceso de extracción de datos desde páginas web.

## 1.3 TRABAJAR CON JUPYTER NOTEBOOK

Jupyter es un proyecto sin ánimo de lucro, gratuito y de fuente abierta, destinado a dar apoyo interactivo al desarrollo de la ciencia de datos y a la computación, haciendo uso de cualquier lenguaje de programación. Nuestro entorno de programación será la aplicación llamada Jupyter Notebook con la que trabajaremos haciendo uso de Python como lenguaje de programación.

La instalación de Jupyter es bastante sencilla. Hay que entrar en el enlace <https://jupyter.org/install> y seguir los pasos que describiremos a continuación.

A pesar de que se puede trabajar con cualquier lenguaje de programación, es necesario tener instalado Python en el ordenador. Hay dos maneras de instalar Jupyter. La primera es a través de la plataforma Anaconda y la segunda haciendo uso del método Pip. Nosotros lo haremos con la segunda opción:

Abre directamente la consola (“Terminal” para los sistemas Linux/Mac y “Command Prompt” para Windows) e introduce las sentencias siguientes:

```
python3 -m pip install --upgrade pip  
python3 -m pip install jupyter
```

Para hacer funcionar la nueva aplicación instalada, solo debes escribir:

```
jupyter notebook
```

Si quieras ampliar información, visita la página oficial, donde encontrarás la instalación explicada paso a paso:

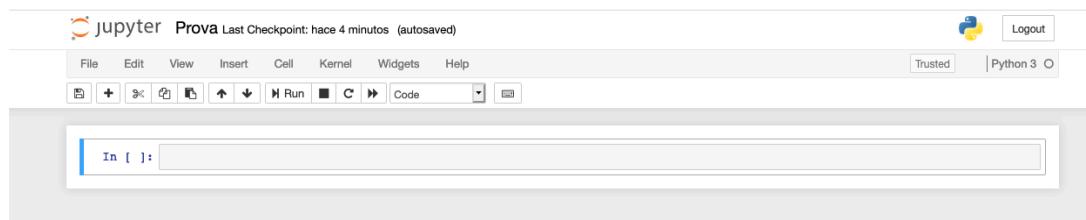
[jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)

Una vez se ha iniciado la aplicación, en la pantalla de la terminal aparecerá la información siguiente:

```
$jupyter notebook  
[I 08:58:24.417 NotebookApp] Serving notebooks from local directory: /Users/catherine  
[I 08:58:24.417 NotebookApp] 0 active kernels  
[I 08:58:24.417 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/  
[I 08:58:24.417 NotebookApp] Use Control-C to stop this server and shut down all kernels  
(twice to skip confirmation).
```

Esto quiere decir que el servidor de Jupyter se ha cargado correctamente y se ha destinado una URL por defecto, que es la que abre el navegador. La página de inicio mostrará el escritorio de la aplicación desde donde se podrá iniciar un nuevo proyecto.

Antes de empezar un proyecto nuevo, crearemos una nueva carpeta para guardar el trabajo del curso: ves arriba a la derecha, donde dice “New”, y selecciona “Folder”. Ahora, iniciaremos un proyecto nuevo dentro de esta carpeta que acabamos de crear. Hay que ir al botón desplegable “New” y hacer clic en Python3. Verás cómo se abre una nueva ventana en el navegador donde aparece el proyecto nuevo.

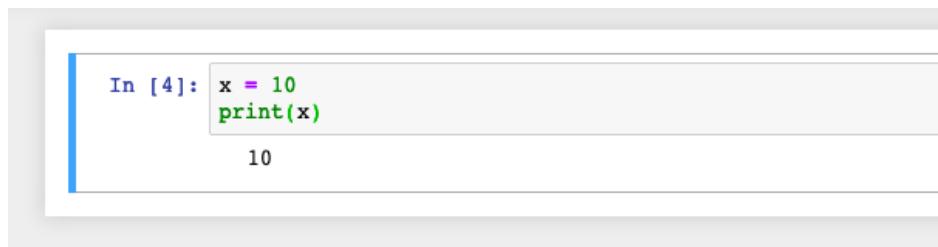


Para cambiarle el nombre, clica sobre el nombre que aparece por defecto (justo a la derecha del logotipo Jupyter) y, a continuación, se abrirá una ventana emergente para escribir el nombre que deseas.

Verás que la pantalla del proyecto tiene tres secciones básicas: la primera fila (donde se encuentran las opciones “File”, “Edit”, “View”, etc.) es el menú con las opciones para

gestionar el proyecto. La segunda fila (donde vemos los iconos de guardar, copiar, etc.) es la barra de herramientas, que nos permite acceder a las operaciones más básicas y más habituales. La tercera fila es la que se denomina *celda*. Aquí podremos escribir código o texto, según el tipo de celda que escojamos.

Por defecto, el proyecto se inicia con una celda de tipo código que nos permitirá escribir nuestro código y también ejecutarlo.



```
In [4]: x = 10
print(x)
```

10

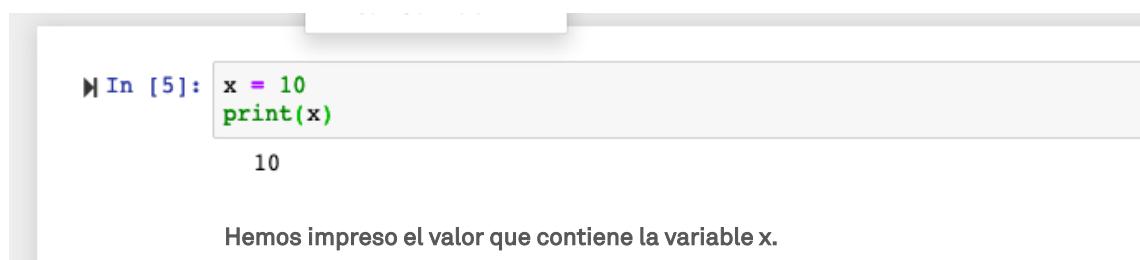
El código que hemos escrito es:

```
x = 10
print(x)
```

Una vez hacemos clic en “Run”, opción que se encuentra en la barra de herramientas, se imprime debajo de la celda el resultado que devuelve el código.

Nota: si, al ejecutar el código, ves que no se imprime nada, haz clic en “Kernel” (opción localizada en la barra del menú) y, a continuación, haz clic en la opción “Restart”.

El otro tipo de celda más habitual es la de texto. Para cambiar el tipo de celda desde el menú, seleccionamos “Cell”, después “Cell type” y, finalmente, “Markdown”. Ahora, la celda se ha convertido en un editor de texto y no de código. Se pueden crear más celdas clicando en “Insert” de la barra del menú. La nueva celda se puede colocar arriba o debajo de la actual seleccionando la opción *above* (arriba) o *below* (debajo). Una vez has escrito un texto, puedes ejecutar la celda y verás cómo se imprime el contenido.



```
In [5]: x = 10
print(x)
```

10

Hemos impreso el valor que contiene la variable x.

En caso de que Jupyter no funcione o no se cargue en el navegador, puedes hacer una de las acciones que se detallan a continuación para intentar solucionar el problema.

- Prueba con otro navegador (por ejemplo, si normalmente usas Firefox, utiliza Chrome).
- Intenta desactivar cualquier extensión del navegador o cualquier extensión de Jupyter que hayas instalado.
- Algunos programas de seguridad de internet pueden interferir con Jupyter. Si tienes un programa de seguridad, prueba de desactivarlo temporalmente y mira la configuración para obtener una solución.
- En la barra de navegación (URL), cambia el localhost por 127.0.0.1.

Para más información sobre la instalación de Jupyter y su funcionamiento, recuerda que puedes visitar la página oficial de la plataforma: <https://jupyter-notebook.readthedocs.io/en/latest/index.html>.

## 1.4 PROFUNDIZAR EN JUPYTER NOTEBOOK

¡Hola a todo el mundo!

En este video, profundizaremos en las acciones y herramientas de la aplicación Jupyter.

Una vez tengamos el proyecto creado (recuerda que un proyecto es un *notebook*), podemos añadirle, primero, el título. La primera celda la cambiaremos a modo de encabezamiento de esta manera.

Abrimos el desplegable de la barra de herramientas y clicamos “Heading”. Aquí aparece un mensaje diciendo que, solo añadiendo una almohadilla en una celda normal de texto, es decir de tipo Markdown, ya tenemos bastante. Pues, lo haremos así. Podemos ponerle el título que queramos.

Escribimos este código para empezar a hacer cosas nuevas:

```
lista_resultados = [12.5, 14.7, 21.6, 15.7]
nueva_lista = []
for num in lista_resultados:
    if num > 15.0:
        nueva_lista.append(num)
print(nueva_lista)
```

Este código nos crea y nos imprime por pantalla una nueva lista que solo incluye esos valores superiores a 15.0 de la primera lista.

Clicamos en “Run” y vemos el resultado.

[21.6, 15.7]

Fíjate que, cuando hacemos clic en “Run”, por defecto, se genera una celda nueva, siempre de tipo “código”.

Si durante el desarrollo del proyecto, vemos que es necesario insertar una celda antes de otra que ya tiene contenido, lo podemos hacer a través del menú, clicando “Insert” y “Insert Above”. En esta nueva celda, podemos indicar la acción del código que encontramos a continuación:

*Escribimos código para añadir una lista de valores superiores a 15.0*

¡Importante! Recuerda que, cuando quieras introducir texto, tendrás que cambiar el tipo de celda de código a Markdown.

Si ahora queremos eliminar una celda, la seleccionamos y clicamos sobre el icono de las tijeras que se encuentra en la barra de herramientas. Después, se pueden añadir más celdas y seguir trabajando solo haciendo clic en el botón +, situado en la misma barra.

Un truco que ayudará a localizar código más rápidamente dentro de una celda es numerar las líneas de código. Esta opción se encuentra en la barra del menú, en la sección “View”. Si hacemos clic en “Toggle Line Numbers” veremos cómo aparece esta numeración.

Si, a medida que vayamos desarrollando el proyecto, queremos previsualizar cómo quedaría el trabajo impreso, solo hay que ir a “File” y seleccionar “Print Preview”. Recuerda que mucho del trabajo que se hace en ciencia de datos se tiene que mostrar a otras personas para encontrar apoyo y aplicar los nuevos conocimientos en nuestra empresa, equipo de investigación, etc. Esto quiere decir que la presentación del *notebook* es muy importante.

También podemos convertir nuestro *notebook* en un fichero HTML, para publicarlo en una web. Para hacerlo, en este caso, solo hay que seleccionar “Download as” y a continuación “HTML”.

Si quieres hacer un repaso a la navegación básica de la aplicación, puedes hacer clic en la sección “Help” y seleccionar “User Interface Tour”, que te guiará a través de las acciones de Jupyter Notebook.

¡Eh! Antes de terminar, piensa a guardar el trabajo a menudo. Aunque la aplicación lo haga de manera automática, es importante tener presente que lo puedes hacer de manera manual, solo haciendo clic en el icono del disco.

Ahora ya estamos en condiciones de trabajar en un proyecto de ciencia de datos haciendo uso de Jupyter Notebook.

¡Sigamos!

## 1.5 IDEAS CLAVE: INTRODUCCIÓN A LA CIENCIA DE DATOS Y AL ENTORNO DE TRABAJO DE JUPYTER NOTEBOOK

En este primer módulo, hemos aprendido los conceptos básicos de la ciencia de datos y sus aplicaciones más relevantes. Ahora sabemos que la información codificada en forma de datos se puede depurar y limpiar, y podemos encontrar patrones para generar conocimiento aplicado.

Los campos donde más se usa la ciencia de datos son el mundo empresarial y la investigación en general. El aprendizaje automático es la capacidad de una máquina de mejorar sus pronósticos, localizando patrones con un mayor grado de certeza, a medida que esta máquina va gestionando más datos. Hay dos tipos de modelos:

- Modelo de regresión. Sirve para calcular un valor determinado, es decir, un número.
- Modelo de clasificación. Sirve para responder a una cuestión en solo dos opciones (binomio) o con más de dos opciones (multinomial).

Jupyter Notebook es una plataforma gratuita y de fuente abierta muy popular para desarrollar proyectos de ciencia de datos. Esta aplicación tiene tres secciones: la barra de menú, la barra de herramientas y la sección de las celdas. En cuanto a las celdas, podemos diferenciar, principalmente, de dos tipos: uno corresponde al código y el otro al texto. Cuando necesitemos introducir código, escogeremos el primero y, cuando necesitemos escribir texto, usaremos el segundo. Esto nos permite escribir código y añadir explicaciones escritas y relevantes sobre este, y presentarlo todo en el mismo documento.

Jupyter Notebook permite añadir y eliminar celdas fácilmente. Las celdas nuevas se pueden colocar allí donde más nos convenga. La aplicación también nos permite visualizar nuestro trabajo mediante una previsualización e, incluso, exportar el fichero a HTML, para que se pueda colgar en una web.

## 2 EXPLORACIÓN INICIAL EN LOS DATOS

Después de una introducción al mundo de la ciencia de datos, es el momento de empezar a trabajarla. Lo primero que haremos será estudiar los datos. Normalmente, los datos nos llegan en forma de tabla, con filas y columnas. Veremos cómo importar o cargar una tabla y cómo extraer rápidamente la información general que contiene. Para hacerlo, necesitaremos aplicar unas funciones sencillas con Python. Todo este trabajo será ejecutado en nuestro nuevo entorno de programación Jupyter Notebook. Este primer paso es muy importante y siempre lo tendremos que ejecutar en cualquiera de nuestros proyectos, independientemente de la complejidad de los datos que usamos en cada momento.

También profundizaremos en los dos tipos de modelos predictivos básicos: el de regresión y el de clasificación. Empezaremos, primero, desde un punto de vista teórico o descriptivo y, después, cogeremos experiencia con ejercicios prácticos. Todo lo haremos paso a paso.

## 2.1 EXPLORACIÓN BÁSICA DE LOS DATOS

La ciencia de datos nos permite descodificar la información para adquirir más conocimientos de un sistema o de nuestro entorno. Normalmente, estos datos vienen en forma de tabla, conformada por dos entidades: filas y columnas.

Las filas, generalmente, contienen los registros u observaciones, y las columnas contienen las variables o atributos. Los registros, pues, son los casos de estudio que incorporan una información determinada en cada atributo. Veamos un ejemplo sencillo:

	EDAD	PESO	ALTURA	CIUDAD	SEXO
JOAN	35	80	1,75	BARCELONA	HOMBRE
ALBA	32	65	1,68	TARRAGONA	MUJER
MIQUEL	47	74	1,78	GIRONA	HOMBRE
MERITXELL	44	59	1,60	LLEIDA	MUJER

- Las variables o atributos (columnas) son: edad, peso, altura, ciudad y sexo.
- Las observaciones o registros (filas) son: Joan, Alba, Miquel y Meritxell.

Nota: de ahora en adelante siempre nos referiremos a las filas como observaciones y a las columnas como atributos.

Es importante saber que las tablas pueden venir con dos tipos de datos: los **operacionales** y los **organizativos**. Los primeros contienen información directa de observaciones unitarias y puntuales como, por ejemplo, el importe y el producto de cada compra en un establecimiento, todas y cada una de las compras de billetes de avión de una determinada compañía área, etc. Los segundos, en cambio, recogen datos agrupados o tendencias. Un ejemplo de dato organizativo serían los casos de una enfermedad agrupados en diferentes ciudades: aquí no estudiamos caso por caso una observación individual, sino una agrupación basada en ciertos criterios. Este segundo grupo se usa muchas veces cuando pueden aparecer conflictos relacionados con la protección de datos.

Recuerda que lo primero que necesitamos establecer antes de empezar a trabajar con los datos son los **objetivos**. Tenemos que saber responder a la pregunta: ¿qué información relevante queremos obtener con estos datos? Una vez lo tengamos claro, nos tendremos que poner a trabajar para conseguir la respuesta. Después del planteamiento de los objetivos, los siguientes pasos son la obtención de la información (que tendremos resuelta, cuando dispongamos de una tabla de datos) y, finalmente, la preparación de los datos. Para hacer este último paso, tenemos que ver si tenemos, o no, celdas con valores anormales o sin información (*missing value*).

Nota: es importante que nos familiaricemos con expresiones en inglés como *missing values*, puesto que la industria tiene como referencia este idioma y, a pesar de que trabajes en Cataluña, seguro que las tendrás que usar.

Vamos a ver, primero, un ejemplo de datos anormales:

	EDAD	PESO	ALTURA	CIUDAD	SEXO
JOAN	35	80	1,75	BARCELONA	HOMBRE
ALBA	32	65	1,68	TARRAGONI	MUJER
MIQUEL	47	74	1,78	TARRAGONA	HOMBRE
MERITXELL	44	59	1,06	LLEIDA	MUJER

En una exploración rápida, podríamos sospechar que la altura de Meritxell puede tener un valor anormal (probablemente, debido a un error a la hora de poner el dato). También vemos que el valor de ciudad de Alba es Tarragoni y, por lo tanto, no corresponde con el valor real. Estos errores, tarde o temprano, pueden aparecer y, si no los detectamos, la información que extraemos puede ser de baja calidad. A veces, es prácticamente imposible detectarlos. Por eso, es importante trabajar con una base de datos amplia, que tenga suficientes observaciones para que estos errores no nos alteren excesivamente nuestros modelos predictivos.

En los casos de celdas sin valores, las causas pueden ser muy diversas. Si, por ejemplo, registramos con un termómetro la temperatura cada cierto periodo de tiempo y vemos celdas sin valores, será probablemente debido a un mal funcionamiento del instrumento de medida. A continuación, mostramos un ejemplo de un sistema formado por cinco termómetros, que nos sirve para medir la temperatura ambiente cada 30 minutos en varias zonas de una determinada ciudad.

El objetivo de esta tabla podría ser, por ejemplo, ver si hay ciertas partes de la ciudad que son más calurosas que otras. El estudio nos podría aportar conocimiento importante si podemos relacionar las variaciones de temperatura con la arquitectura del municipio o con el tráfico. Esto nos permitiría proponer cambios urbanísticos, si se considerase oportuno.

	TERM 1	TERM 2	TERM 3	TERM 4	TERM 5
08:00	14,5	13,9	14,9	14,3	14,0
08:30	14,8	14,3	15,5	14,7	14,3
09:00	15,3	14,9	15,9		14,8

09:30	15,9	15,5	16,5		15,4
10:00	16,4	16,0	16,9		16,1
10:30	17,0	16,4	17,3	16,9	16,5
11:00	17,7	17,0	17,8	17,6	16,9

Nota: las columnas son los diferentes termómetros situados a distintos puntos de la ciudad.

Las celdas de color gris no contienen valores dentro de la franja horaria de 9:30 a 10:00. Por alguna razón desconocida, el termómetro no ha registrado la temperatura. En un caso real, tendríamos que analizar si esta carencia de información podría alterar significativamente nuestro conocimiento.

## 2.2 EJEMPLO DE EXPLICACIÓN DE DATOS

¡Hola!

Ahora cargaremos una tabla de datos y extraeremos la información básica. En primer lugar, crearemos un proyecto nuevo. Después, cargaremos el módulo de Python denominado *Pandas*. Este módulo nos permitirá trabajar con bases de datos, en inglés *Data Frames*. Así que introducimos este código en la primera celda:

```
import pandas as pd
```

Hemos denominado nuestro módulo *Pandas as pd*. A partir de ahora, cada vez que nos refiramos a él o lo llamemos con nuestro código, usaremos *pd*.

El siguiente paso es crear una variable para almacenarla en nuestra base de datos. Antes, pero, tengamos en cuenta que tenemos que tener guardada nuestra base de datos en formato CSV. Podemos exportar un documento con formato CSV desde el programa Excel.

Te puedes descargar el fichero (la base de datos) en este enlace ([...../iris.csv](#)). Guárdalo en la misma carpeta donde tenemos el proyecto.

Luego, introducimos este código:

```
data_frame = pd.read_csv('iris.csv')
```

Esta base de datos recoge las longitudes y anchuras del sépalo y del pétalo de una serie de flores.

En realidad, esta variable es un objeto que se crea logrando las funciones del paquete Pandas. Más adelante, tendremos que hacer uso de estas funciones. Recuerda que en Python todo son objetos.

Fíjate que estamos usando funciones. `read_csv()` es una función del módulo `pd`(Pandas) que nos permite cargar ficheros del tipo CSV.

Ejecutamos la variable `data_frame`, para visualizar la base de datos.

También podemos ver solo un determinado número de observaciones. Si aplicamos la función `head()` en la base de datos, escribiendo el código siguiente:

```
data_frame.head()
```

Ahora solo aparecen los cinco primeros registros; pero, si ponemos el número de registros que queremos visualizar como argumento de la función, la tabla mostrará los que le digamos. Por ejemplo, si queremos ver los diez primeros, escribiremos:

```
data_frame.head(10)
```

Incluso podremos visualizar solo una serie de observaciones y solo de los atributos que queramos. Por ejemplo, si escribimos este código:

```
data_frame.loc[[0, 1, 6], ['Sepal.Length']]
```

Aquí veremos los datos de las observaciones 0, 1, 6..., es decir, de la primera, segunda y séptima..., respecto a la longitud del atributo sépalos.

Ahora, si aplicamos la función `describe()` a toda la base de datos con este código:

```
data_frame.describe()
```

vemos una tabla que recoge una serie de datos para cada columna. Vamos a analizar qué quieren decir cada uno de estos resultados.

- Count: es el número de observaciones para cada atributo. Y, claro, todas tienen el mismo número de observaciones.
- Mean: nos indica la media, considerando todos los valores de la columna.
- std: nos muestra la desviación de los resultados.
- min: el valor más pequeño del atributo.
- 25%: expone el valor que es mayor que el 25 % de la serie.
- 50%: el valor que es mayor que el 50 % de la serie.
- 75%: el valor que es mayor que el 75 % de la serie.
- Max: el mayor valor de la columna.

Ahora ya conocemos las funciones básicas que nos permiten cargar y explorar una base de datos. Esta primera exploración es el inicio que nos permitirá ver, después, qué modelo predictivo necesitaremos usar para conseguir nuestro objetivo. Entender los datos es la primera tarea y un paso que nunca nos tenemos que saltar.

¡Sigamos!

## 2.3 MODELOS PREDICTIVOS

Un modelo predictivo es un mecanismo que nos permite calcular, con un grado de certeza determinado, un resultado futuro a través de la relación de unos datos previamente dados. Hay que decir que los modelos son muy diversos y el uso de unos u otros variará en función del tipo de información que queramos conseguir (por ejemplo, dependerá de si queremos obtener un valor determinado o, en cambio, si queremos conocer una situación entre diversas de posibles). La elección de un modelo también variará según el tipo de datos y las relaciones que se establezcan entre ellas. Lo más importante es saber que ningún modelo es perfecto.

Para determinar el grado de certeza de un modelo, lo tendremos que entrenar. Esto quiere decir que parte de las observaciones de nuestra base de datos las utilizaremos para prever si el modelo genera resultados correctos o no. Una vez hemos entrenado varios modelos y hemos visto cuál funciona mejor, podremos escoger uno y desarrollar el trabajo.

Para escoger un buen modelo, primero tenemos que entender los datos: de qué fuente vienen, a qué se refieren, qué relación tienen con nuestro objetivo, etc. La calidad de los datos es el primer punto a tener en cuenta. Si, para llegar a un objetivo, escogemos los datos incorrectos, la información que gestionemos nos llevará a conclusiones erróneas que provocarán que tomemos decisiones incorrectas.

Los modelos se pueden clasificar en dos grandes grupos: los de regresión y los de clasificación, a pesar de que hay muchos que pueden servir para ambos casos. Las predicciones de regresión procura predecir un valor numérico discreto o determinado, mientras que los de clasificación buscan predecir un estado de entre varios escenarios posibles. A continuación, mostramos una tabla con varios modelos y a qué grupo pertenecen:

Modelo	Grupo
<i>Random forest</i>	Clasificación
<i>Lineal regression</i>	Regresión
<i>KNN</i>	Regresión y clasificación
<i>Boosted Tree</i>	Regresión y clasificación
<i>Gradient Boosted Machines</i>	Regresión y clasificación

Para que los datos generen patrones entre ellos, tienen que estar relacionados. Dicho de otro modo, los datos que contienen una determinada variable (atributo) están relacionadas, con más o menos afinidad, con el dato que contiene otra variable.

La correlación es la herramienta estadística que mide cómo es de intensa una relación entre un atributo de valor numérico y otro. Los valores de correlación entre dos variables numéricas pueden asumir valores desde 0 a 1 o a -1. Si la correlación es positiva, esto querrá decir que, si el valor de un dato aumenta, también lo hará el otro. También será positiva si el valor de uno baja y el del otro también. En cambio, si la correlación es negativa, significa que, si el valor de uno aumenta, el del otro bajará, y al revés.

El uso de la prueba de independencia Chi-Square permite al personal investigador evaluar si la relación observada entre las variables nominales (no numéricas) en una muestra particular también se puede encontrar en la población. Aun así, esta puede no ser adecuada, si la medida de la muestra no es bastante extensa.

A continuación, analizamos algunos ejemplos de industrias que usan modelos predictivos para mejorar su competitividad:

- La industria aeroespacial necesita modelos predictivos que predigan la duración de los componentes para aumentar así el rendimientos de sus aviones y reducir los costes de mantenimiento.
- Los seguros usan, como herramienta principal, los modelos predictivos, para diseñar productos de riesgo para diferentes tipos de clientela.
- Los bancos calculan mediante modelos predictivos, el endeudamiento límite que puede lograr una persona que encaje dentro de un determinado grupo de la población.
- La medicina usa modelos predictivos para relacionar el sonido respiratorio con el riesgo de sufrir un ataque de asma.
- Un departamento de *marketing* siempre necesita saber si los nuevos productos que está diseñando el área de I+D serán aceptados por el mercado. El riesgo de diseñar un producto nuevo se puede reducir haciendo un estudio previo sobre el comportamiento de la clientela y de productos relacionados.

Los pasos para generar y optimizar un modelo predictivo son los siguientes:

- Una vez hemos estudiado el tipo de datos con los que estamos trabajando y qué objetivo queremos lograr, escogemos uno o más modelos, basándonos en nuestra experiencia.
- Lectura de los parámetros que muestran la certeza de nuestra predicción: optimización del modelo mediante la configuración de ciertos parámetros y comparación con el resto de los modelos escogidos.
- Entender los resultados. Es muy importante saber dominar los resultados con una mirada clara a la situación en la que nos encontramos. Los datos dan valor cuantitativo que necesita una valoración cualitativa, para que se conviertan en conocimiento aplicable.
- Validación del modelo e integración en aplicaciones, páginas web, dispositivos, etc.

Ahora ya tienes los conocimientos necesarios para empezar a aplicar modelos predictivos.

## 2.4 MODELOS DE REGRESIÓN

Actualmente vivimos rodeados de una gran cantidad de datos, computadoras potentes e inteligencia artificial. Esto solo es el principio. La ciencia de datos y el aprendizaje automático impulsan el reconocimiento de imágenes, el desarrollo de vehículos autónomos, decisiones en los sectores financiero y energético, avances en medicina, aumento de las redes sociales y mucho más. Los modelos de regresión juegan un papel muy importante en este campo. Predecir el comportamiento de un sistema nos permitirá tomar decisiones más acertadas o, incluso, diseñar nuevos productos que tengan éxito en los mercados.

Un modelo predictivo de regresión se usa, principalmente, para predecir la respuesta o resultado de una variable, según los cambios que logran unas variables que están relacionadas con esta. Imaginemos que queremos estimar el crecimiento de las ventas de una empresa en función de las condiciones económicas actuales. Tenemos los datos recientes de la empresa que indican que el crecimiento de las ventas es alrededor de dos veces y media el crecimiento de la economía. Con esta perspectiva, podemos predecir las ventas futuras de la empresa, a partir de información actual y anterior.

La regresión lineal es la técnica de regresión más sencilla y la que ofrece la interpretación de resultados más sencilla. Vamos a descubrir sus fundamentos matemáticos.

Cuando implementamos una regresión lineal de alguna variable dependiente y del conjunto de variables independientes  $\mathbf{x} = (x_1, \dots, x_r)$ , donde  $r$  es el número de predictores, asumimos una relación lineal entre  $y$  y  $\mathbf{x}$ :  $y = \beta_0 + \beta_1 x_1 + \dots + \beta_r x_r + \varepsilon$ . Esta ecuación es la de regresión.  $\beta_0, \beta_1, \dots, \beta_r$  son los coeficientes de regresión y  $\varepsilon$  es el error aleatorio.

Y se correspondería al atributo del que queremos prever el resultado. Los valores de  $\mathbf{x}$  corresponden con los atributos de los que se supone una cierta relación con  $y$ .

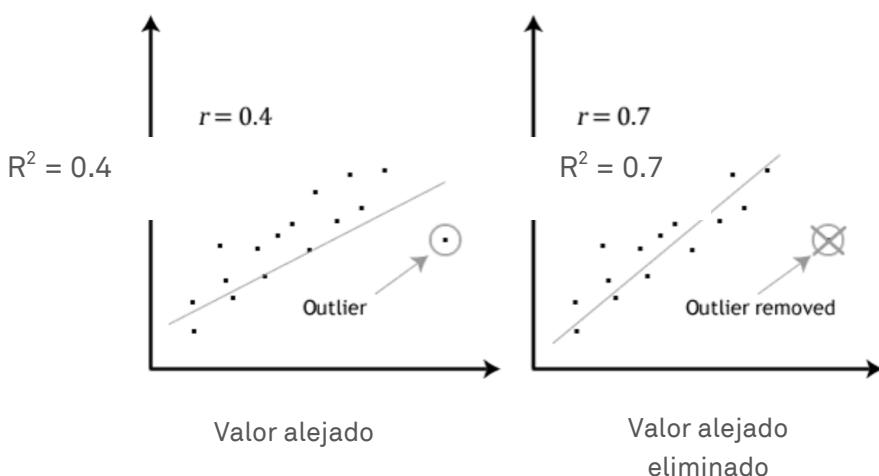
La variación de las respuestas reales  $y_i$ ,  $i = 1, \dots, n$ , se produce, en parte, a causa de la dependencia del atributo  $x_i$ . Aun así, también hay una diferencia inherente a la salida. El coeficiente de determinación, conocido como  $R^2$ , indica qué cantidad de variación en  $y$  se puede explicar para la dependencia de  $\mathbf{x}$  mediante el modelo de regresión. El mayor  $R^2$  indica un mejor ajuste y significa que el modelo puede explicar mejor la variación de la salida con diferentes entradas. El valor  $R^2 = 1$  se adapta perfectamente, ya que los valores de las respuestas previstas y reales se ajustan y coinciden perfectamente.

La regresión lineal más simple es aquella en la que la variable  $y$  solo depende de otra variable  $\mathbf{x}$ . En los casos prácticos, en ciencia de datos, este caso no se suele dar. Los casos reales son más complejos y normalmente tienen más de una variable  $\mathbf{x}$  relacionada con un atributo  $y$ . En estos casos reales, estaremos hablando, pues, de regresión lineal múltiple.

Si solo hay dos variables independientes, la función de regresión estimada es  $f(x_1, x_2) = b_0 + b_1 x_1 + b_2 x_2$ . El objetivo de la regresión es determinar los valores de los coeficientes  $b_0, b_1$  y  $b_2$ , de modo que los resultados estén lo más cerca posible de las respuestas reales.

Cuando generamos los modelos, tenemos que ver si estos pueden tener un coeficiente de correlación demasiado bajo o demasiado alto (muy cerca de 1). En el primer caso, estaremos

ante un modelo que genera unos resultados demasiado alejados de los reales. Estaremos ante una situación de *underfitting*. En el segundo caso, estaremos ante un sistema de *overfitting*. Esto querrá decir que el modelo encaja perfectamente con los datos dados, pero corre el riesgo de prever muy mal cuando se aplica a otros datos con los que no ha sido entrenado. Como que el error es la distancia cuadrada entre el punto de datos y la línea de regresión, las grandes distancias tienen errores desproporcionadamente grandes, y hacen que el análisis de regresión se convierta en una solución con un coeficiente de correlación bajo. Como tal, esos valores que estén significativamente alejados de la recta, tendrían que ser eliminados del conjunto de datos. Estos valores son aquellos que se podrían descubrir durante la exploración y depuración de datos.



En las dos gráficas que se muestran justo arriba, se ve como una vez hemos eliminado el valor alejado, nuestro modelo predictivo ha mejorado considerablemente. Para facilitarnos la interpretación de los resultados, en ciencia de datos es muy útil contar con gráficas que muestren los datos de manera visual.

## 2.5 EJEMPLO DE UN MODELO DE REGRESIÓN

¡Hola!

En este video, usaremos nuestro primer modelo. Concretamente, crearemos un modelo de regresión. Este nos servirá para ver cómo podemos predecir la anchura de los pétalos, mirando la relación que hay entre las anchuras y las longitudes de los pétalos.

Bien, primero creamos un proyecto nuevo. Seguro que ya lo sabes hacer, pero, si no, es muy fácil. Clicamos en “New” y seleccionamos “Python 3”. Después, importamos el módulo *Pandas* y cargamos la base de datos con la que queremos trabajar. Usaremos la llamada ‘iris.csv’.

```
import pandas as pd
data_frame = pd.read_csv('iris.csv')
```

Una vez tenemos la base de datos cargada, necesitamos importar otros módulos que nos permitan seleccionar parte de los datos, para poder entrenar el modelo.

```
from sklearn.model_selection import train_test_split
```

Después, importaremos concretamente el modelo que queremos usar a partir de una regresión lineal.

```
from sklearn.linear_model import LinearRegression
```

Después de importar estas librerías, lo que haremos será seleccionar solo las columnas “Petal.Width” y “Petal.Length”, y las guardaremos en otra variable. Esta variable estará guardando otra base de datos.

```
df = data_frame[['Petal.Length', 'Petal.Width']]
```

Ahora, nos toca seleccionar de manera aleatoria una serie de observaciones para entrenar el modelo. Concretamente, estamos cogiendo el 80 % de los registros y dejaremos el 20 % restante para testar nuestro modelo.

```
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)
```

Antes de continuar, guardaremos una copia del *train\_set* en una nueva variable.

```
df_copy = train_set.copy()
```

Una vez hemos hecho esto, aplicaremos la función *corr()* para ver cómo están relacionados los valores de las dos columnas de la parte de entrenamiento. Recuerda que ahora ya no estamos trabajando con toda la base de datos inicial. Primero, hemos seleccionado las columnas que queríamos estudiar y, después, hemos cogido el 80 % para entrenar el modelo.

```
df_copy.corr()
```

Un valor de 0,9625 es bastante alto. La relación es muy estrecha. Recuerda que R mide cómo están de relacionadas entre sí dos variables. Cuanto más próximo sea este valor a 1 o -1, más estrecha será la relación. Esto querrá decir que un cambio en una variable afectará directamente a la otra.

```
from sklearn.metrics import r2_score
```

Ahora sería interesante visualizar los valores de las dos columnas en una gráfica. Por eso, importamos el módulo siguiente:

```
import matplotlib.pyplot as plt
```

Asignamos, ahora, qué columna corresponderá al eje  $X$  y cuál al eje  $Y$ , y ya lo tenemos.

```
df_copy.plot.scatter(x='Petal.Length', y='Petal.Width')
```

Se ve claramente como las dos variables están estrechamente ligadas. Cuando una crece, también lo hace la otra, tal como nos indicaba el resultado de la función `corr()`.

Bien, ahora viene el paso decisivo: aplicar el modelo escogido.

```
train_set_x = df_copy..drop(["Petal.Width"], axis=1)
```

```
train_set_label = df_copy["Petal.Width"]
```

Con este código, lo que estamos haciendo es decirle al modelo cuál será la variable independiente y cuál será la variable de la que queremos predecir los resultados en función de la primera.

```
lin_reg = LinearRegression()
```

```
lin_reg.fit(train_set_x, train_set_label)
```

Creamos una instancia del objeto `LinearRegression()` y usamos la función `fit()` que contiene, para aplicar el modelo. Además, este objeto `lin_reg` también incorpora `coefs` e `intercept`.

```
print("Coefficients: ", lin_reg.coef_)
```

```
print("Intercept: ", lin_reg.intercept_)
```

Los resultados que obtenemos son:

*Coefficients: [0.41323829]*

*Intercept: -0.3566680410565528*

Con estos valores, ya podemos escribir la fórmula matemática que define la regresión lineal:

$$\text{Petal.width} = 0.4132 * \text{Petal.Length} - 0.3566$$

Pero, en vez de hacerlo a mano, el objeto `lin_reg` nos da la función `predict()`, en la que, si ponemos el valor de la longitud del pétalo como argumento, esta nos devolverá el valor de la anchura que predice.

Este es un ejemplo sencillo, pero bastante didáctico para entender el concepto. La vida real tiene problemas más complejos para resolver, pero este es un buen inicio.

¡Sigamos!

## 2.6 MODELOS DE CLASIFICACIÓN

La clasificación es una función de gestión de datos que asigna elementos de una colección a categorías, clases o también a los llamados segmentos. El objetivo de la clasificación es predecir con certeza la clase de objetivo para un elemento dado de la colección. Por ejemplo, se podría utilizar un modelo de clasificación para identificar a qué categoría de riesgo (bajo, alto o mediano) se pueden asignar las personas solicitantes de préstamos. En este caso, las personas equivaldrían a los elementos de la colección (en este caso, una población) y los riesgos serían las categorías.

Un modelo de clasificación empieza con un conjunto de datos de los que se conocen las asignaciones de las categorías. Por ejemplo, se podría desarrollar un modelo de clasificación que prevé el riesgo de crédito basado en datos observados para muchas personas solicitantes de préstamos ya existentes, que tienen asignado un riesgo alto, bajo o mediano. Además de la calificación de crédito histórico, los datos podrían hacer un seguimiento del historial de ocupación, propiedad o alquiler de viviendas, años de residencia, número y tipo de inversiones, etc. La calificación crediticia sería el objetivo, los otros atributos serían predictores y los datos de cada cliente o clienta constituirían una observación.

La clasificación más simple es la clasificación binaria, en la que el atributo solo tiene dos valores posibles; por ejemplo, calificación crediticia alta o calificación crediticia baja.

En el proceso de creación de modelos, un algoritmo de clasificación encuentra relaciones entre los valores de los atributos y la categoría o segmento donde queremos colocar una observación. Diferentes algoritmos de clasificación utilizan técnicas diversas para encontrar estas relaciones. Las relaciones, pues, se codifican en un determinado modelo que sirve después para predecir qué elementos de una colección caerán en una categoría o en otra. Los modelos de clasificación se prueban mediante la comparación de los valores predichos con los valores objetivo conocidos. Los datos históricos de un proyecto de clasificación se dividen generalmente en dos conjuntos de datos: uno para construir el modelo; el otro para probar el modelo. Suponemos que queremos predecir cuáles de nuestros clientes y clientas podrían

aumentar el gasto, si se les proporciona una tarjeta de afinidad. Podemos crear un modelo mediante datos demográficos sobre clientela que ha utilizado anteriormente una tarjeta de afinidad. Como queremos predecir una respuesta positiva o negativa (saber si aumentará o no el gasto), este caso es un claro ejemplo de un modelo de clasificación binario.

Los datos de prueba tienen que ser compatibles con los datos que se utilizan para crear el modelo y se tienen que preparar del mismo modo que se prepararon los datos de creación. Normalmente, los datos de creación y los datos de prueba provienen del mismo conjunto de datos históricos. Un porcentaje de los registros se utiliza para crear el modelo; los registros restantes se utilizan para probarlo.

Las métricas de prueba se utilizan para evaluar la precisión del modelo de predicción de los valores conocidos. Si el modelo funciona bien y cumple los requisitos comerciales, se puede aplicar a datos nuevos, para predecir el futuro.

A continuación, vamos a describir los parámetros que miden la calidad del modelo empleado:

- La precisión (CVA) hace referencia al porcentaje de predicciones correctas realizadas por el modelo, respecto a las clasificaciones reales de los datos de prueba.
- Confusion matrix es una matriz que muestra el número de predicciones correctas e incorrectas realizadas por el modelo, respecto a las clasificaciones reales de los datos de prueba. La matriz es n-por-n, donde n es el número de clases o segmentos.
- La elevación o lift mide el grado en el que las predicciones de un modelo de clasificación son mejores que las predicciones generadas aleatoriamente. La elevación solo se aplica a la clasificación binaria y requiere la designación de una clase positiva. (Consultad “Clases positivas y negativas”). Si el modelo en sí no tiene un destino binario, podemos calcular la designación, designando una clase como positiva y combinando todas las otras clases como una clase negativa.

Este parámetro se utiliza habitualmente para medir el rendimiento de los modelos de respuesta en aplicaciones de *marketing*. El objetivo de un modelo de respuesta es identificar segmentos de la población con concentraciones potencialmente altas de encuestados positivos en una campaña de *marketing*. La elevación revela qué parte de la población se tiene que solicitar para obtener el porcentaje más alto de posibles personas que respondan.

ROC es una representación gráfica de la sensibilidad ante la especificidad para un sistema de clasificación binario, según varía el umbral de discriminación.

## 2.7 EJEMPLO DE UN MODELO DE CLASIFICACIÓN

¡Hola!

En este video, usaremos un modelo de clasificación. Concretamente el KNN, para poder predecir qué categoría de estrellas (entre 1 y 5) tendrá un comentario, dependiendo de su número de palabras y del valor del sentimiento (valores entre -4 y 4). El valor del sentimiento hace referencia al grado de intensidad que se añade arbitrariamente para dar más peso o menos a los comentarios. ¿Todo preparado? Pues, ¡empecemos!

Como vemos, ya tenemos el proyecto creado. Podemos descargarnos la base de datos con la que trabajaremos a través del enlace que encontramos debajo del video. Las primeras líneas de código son:

```
import pandas as pd
df = pd.read_csv("reviews_sentiment.csv", sep=',')
```

A continuación, podemos ver un primer resumen estadístico. ¿Recuerdas cuál era la función que podemos usar? Sí, es esta:

```
df.describe()
```

Aquí vemos, pues, que la categoría mínima de los comentarios es 1 y la máxima es 5. Y que el número de palabras de un comentario es 1 y el máximo es 103.

Acerquémonos a estos datos de una manera más visual. Escribamos este código:

```
import matplotlib.pyplot as plt
df.hist()
plt.show()
```

Pasemos, ahora, a preparar el modelo. Primero, importemos estos dos módulos:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

Una vez hecho este paso, preparamos los datos de tal manera que tengamos *Star Rating* como variable resultante y *wordcount* *sentimentvalue* como variables dependientes.

```
X = df[['wordcount', 'sentimentValue']].values
y = df['Star Rating'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Antes de aplicar el modelo, necesitamos importarlo desde el módulo donde está presente:

```
from sklearn.neighbors import KNeighborsClassifier
```

Y ahora sí que lo tenemos preparado. Escribimos el código:

```
n_neighbors = 7
knnModel = KNeighborsClassifier(n_neighbors)
knnModel.fit(X_train, y_train)
```

Para ver si el modelo es bueno o no, tendremos que calcular *Accuracy*. Lo hacemos de la siguiente manera:

```
print('Accuracy del modelo K-NN por el train set es: {:.2f}'
      .format(knnModel.score(X_train, y_train)))
```

El resultado que obtenemos es:

*Accuracy del modelo K-NN por el train set es: 0.90*

Lo que nos está indicando es que la precisión por el paquete de entrenamiento o *train set* es del 90 %.

Fíjate que estamos usando un modelo de clasificación porque queremos predecir una categoría (1, 2, 3, 4 o 5) y no un número de valor cuantitativo, como en el ejercicio anterior.

Para trabajar sobre la precisión del modelo, necesitamos importar las funciones siguientes:

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

Y, después, ponerlas a trabajar sobre nuestro modelo:

```
pred = knnModel.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

Como vemos, el resultado “f1-score” es del 87 % y, por lo tanto, podemos considerar que la precisión es bastante buena.

Ahora, nos queda ver qué categoría asigna nuestro modelo, considerando un número de palabras (*wordcount*) y el valor de sentimiento (*sentimentvalue*) determinados. Introducimos este código:

```
clf = KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X, y)
```

Introducimos el *wordcount* y el *sentimentvalue* en la función *predict()*:

```
print(clf.predict([[5, 1.0]]))
```

Y el modelo nos dice que, en este caso, la categoría más probable sería 5.

Podemos usar la función *predict\_proba()*, para ver qué probabilidad tendría cada categoría para un determinado comentario, sabiendo el número de palabras y el valor de sentimiento. Lo haremos así:

```
print(clf.predict_proba([[20, 0.0]]))
```

El resultado que obtenemos es:

```
[[0.00381998 0.02520212 0.97097789 0. 0. ]]
```

Esto quiere decir que la probabilidad de cada categoría para un comentario de 20 palabras con un valor de sentimiento de 0.0 es la siguiente:

Categoría 1 – 0,3%

Categoría 2 – 2,52 %

Categoría 3 – 97,09 %

Categoría 4 – 0 %

Categoría 5 – 0 %

Ya hemos visto un modelo de clasificación para predecir qué categoría tendría un comentario, teniendo en cuenta su número de palabras y el sentimiento de valor.

¡Sigamos!

## 2.8 IDEAS CLAVE: EXPLORACIÓN INICIAL A LOS DATOS

Lo más importante antes de empezar a usar ningún modelo es entender los datos que tratamos, detectar los errores más comunes, identificar las celdas sin datos y los valores alejados. Es un trabajo esencial encontrarlos y limpiarlos. De lo contrario, estos afectarían negativamente la calidad de los modelos. Para hacer una exploración inicial de los datos, hemos aprendido a hacer uso de funciones básicas como son *head()* y *describe()*.

Tenemos dos tipos básicos de modelo predictivo: el de regresión y el de clasificación. El primero recopila aquellos modelos que procuran predecir valores numéricos cuantitativos, mientras que los modelos que forman parte del segundo tipo se usan cuando el resultado es un valor cualitativo.

Todos los modelos necesitan que una parte de la tabla se use para el entrenamiento. La otra parte restante se puede usar para medir si las predicciones de los modelos son bastante buenas o no. En los modelos de regresión lineal, el parámetro de calidad es el coeficiente de regresión, mientras que, para un modelo de clasificación, es *accuracy*.

## 3 VISUALIZACIÓN DE DATOS CON MATPLOTLIB

Para visualizar los datos que tratamos es útil aprender a generar gráficos. Los gráficos son la herramienta más poderosa para transmitir al público no técnico nuestro trabajo de análisis de los datos: muestran visualmente mucha información que de otra manera podría pasar desapercibida.

En este módulo, aprenderemos a construir gráficos lineales, de barra y de dispersión. Dependiendo del tipo de datos con las que estemos trabajando y las cuestiones que queramos resolver, usaremos uno o el otro. El instrumento que emplearemos será Matplotlib.

### 3.1 GRÁFICAS LINEALES Y DE BARRAS

Ha llegado el momento de aprender a hacer gráficos con los datos. La visualización gráfica es, seguramente, una de las herramientas más potentes para entender rápidamente qué información nos pueden transmitir los datos. En este tutorial, aprenderemos a construir gráficos lineales y de barras haciendo uso de la librería Matplotlib de Python.

Empezaremos iniciando un nuevo proyecto con Jupyter Notebook. Una vez tengamos el proyecto cargado, lo primero que tendremos que hacer es importar la librería `matplotlib.pyplot`.

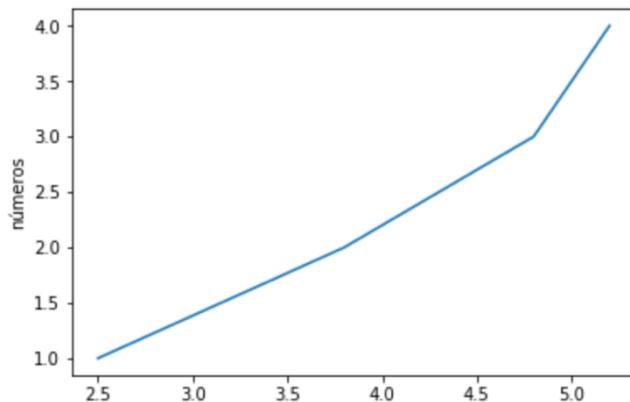
```
import matplotlib.pyplot as plt
```

Nota: la palabra `plt` justo detrás de `as` es el nombre que asignamos a Pyplot, a la hora de usarlo en nuestro código.

A continuación, haremos un gráfico lineal muy sencillo. Escribimos el código que se muestra a continuación en una nueva celda y lo ejecutamos:

```
plt.plot([2.5,3.8,4.8,5.2],[1, 2, 3, 4])
plt.ylabel('números')
plt.show()
```

Si todo va bien, este es el gráfico que aparecerá.



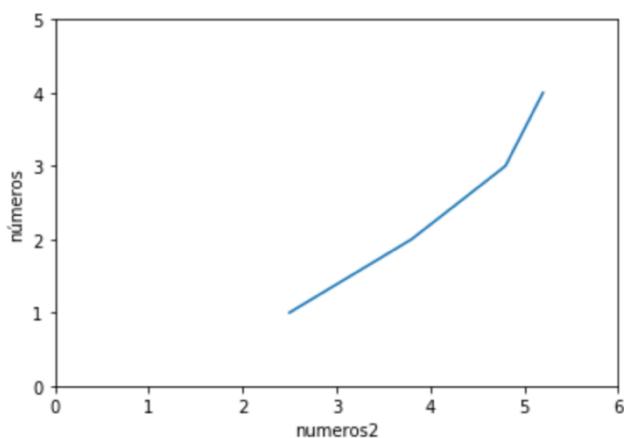
Repasamos las líneas de código que hemos usado para generar el gráfico.

La primera línea, `plt.plot([2.5,3.8,4.8,5.2],[1, 2, 3, 4])`, contiene dos listas: la primera contiene los valores del eje *x* y la segunda los valores del eje *y*. La segunda línea, `plt.ylabel('números')`, indica el título del eje *y*. La tercera línea, `plt.show()`, es la orden que hace imprimir el gráfico. Si queremos, podemos añadir otra línea antes de `plt.show()` con la sentencia `plt.xlabel('...')`, para dar nombre al eje *x*.

Fijémonos, pero, que la intersección entre el eje *y* y *x* no muestra el punto (0,0). Si queremos mostrar que la intersección coincide con el punto (0,0), tenemos que usar la función `axis()`, que también nos permite fijar el valor máximo del eje *y* y el del eje *x*.

```
plt.axis([0,6.0,0,5.0])
```

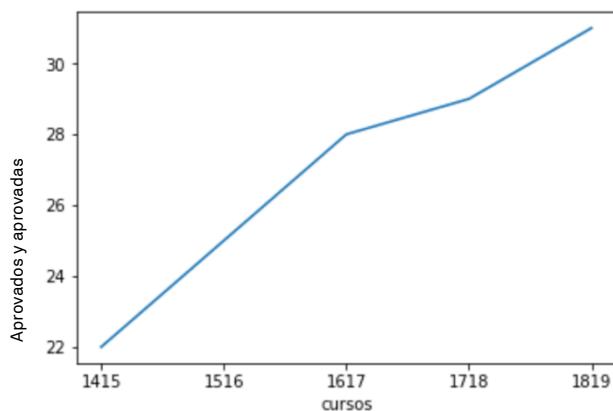
Si ejecutamos el código, este será el gráfico que obtendremos:



Normalmente, los valores de las listas que forman los puntos del eje *x* y *y* ya vendrán dados por una lista que estará guardada en una variable. Supongamos que tenemos el número de alumnos aprobados y aprobadas de una asignatura en relación con diferentes cursos. Abrimos otra celda y escribimos este código:

```
cursos = ['1415', '1516', '1617', '1718', '1819']
aprovats = [22,25,28,29,31]
plt.plot(cursos,aprovados)
plt.xlabel('cursos')
plt.ylabel('aprovados y aprobadas')
plt.show()
```

Este es el gráfico que obtendremos:

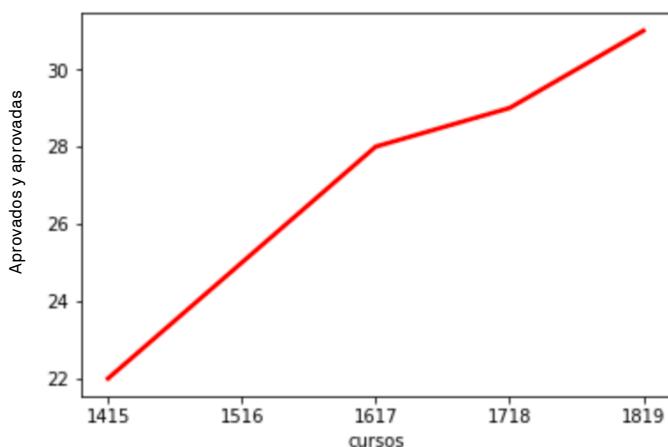


La información que nos muestra el gráfico es que el número de aprobados y aprobadas aumenta año tras año.

Vamos, ahora, a modificar el diseño de los gráficos haciendo uso de `setp()`. Modificamos el código, introduciendo una nueva sentencia con la función `setp()`:

```
cursos = ['1415', '1516', '1617', '1718', '1819']
aprovados = [22,25,28,29,31]
datos = plt.plot(cursos,aprovados)
plt.xlabel('cursos')
plt.ylabel('aprovados y aprobadas')
plt.setp(datos, color='red', linewidth=2.5)
plt.show()
```

El gráfico quedará modificado de este modo:



La función `setp()` toma como argumentos el gráfico formado por los datos del eje *x* y *y*, el color con el que queremos pintar la línea y el grueso de la misma. Podemos modificar estos parámetros y el gráfico se ajustará a las nuevas instrucciones.

Nota: el parámetro `linewidth` hace referencia al grueso de la línea.

Dejamos, ahora, los gráficos lineales y pasamos a construir gráficos de barras. Si cogemos el ejemplo de los aprobados, solo habrá que cambiar la función `plot()` por la función `bar()`.

El código quedaría así:

```
cursos = ['1415', '1516', '1617', '1718', '1819']
```

```
aprovados = [22,25,28,29,31]
```

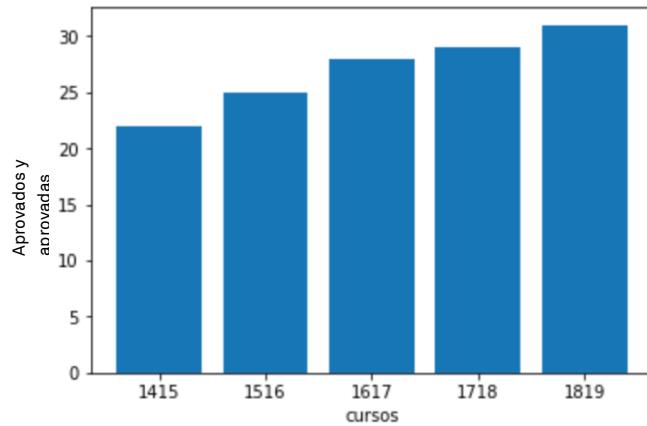
```
plt.bar(cursos,aprovados)
```

```
plt.xlabel('cursos')
```

```
plt.ylabel('aprovados y aprobadas')
```

```
plt.show()
```

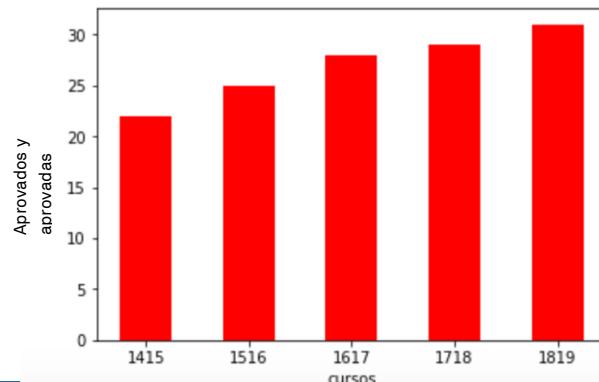
El gráfico que obtendremos es el siguiente:



La función `bar()` permite introducir más parámetros, para personalizar el diseño del gráfico. Por ejemplo, podemos modificar la función, escogiendo el color de las barras y su anchura. Lo haremos así:

```
plt.bar(cursos,aprovados, color='red', width=0.5)
```

El nuevo gráfico será este:



Nota: el parámetro *width* hace referencia a la anchura de las barras.

También tenemos la opción de añadir un color alrededor de la barra y darle, incluso, un grueso. Lo haremos así:

```
plt.bar(cursos,aprobados, color='red', width=0.5, edgecolor='green', linewidth='2.0')
```

Nota: *edgecolor* hace referencia al color del entorno a la barra y *linewidth* al grueso de esta línea.

Si, en vez de trabajar con un gráfico de barras vertical, queremos que sea horizontal, tendremos que cambiar de función. En este caso, la función será *barh()*. Seguimos con los datos de aprobados, según el curso y, escribimos este código:

```
cursos = ['1415', '1516', '1617', '1718', '1819']

aprovados = [22,25,28,29,31]

plt.barh(cursos,aprovados, color='red', height=0.5, edgecolor='green', linewidth='2.0')

plt.xlabel('aprovados y aprobadas')

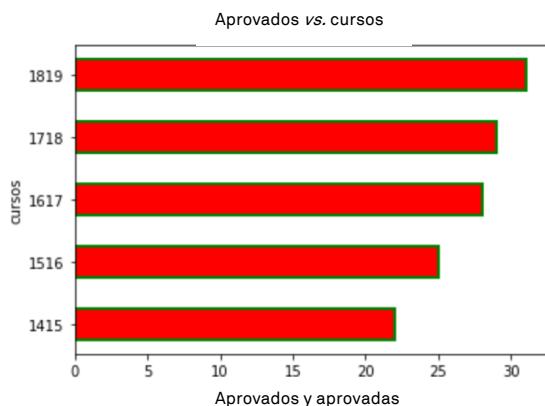
plt.ylabel('cursos')

plt.show()
```

Fijémonos que los parámetros de las dos funciones: *bar()* y *barh()* son prácticamente los mismos, sacado del parámetro *width* de *bar()*, que, en la función *barh()*, pasa a ser *height*. También hemos intercambiado los valores de *xlabel* y *ylabel*, para ser coherentes con la nueva orientación. Por último, hay que decir que, cualquier gráfico hecho con el módulo *plt*, puede tener un título solo gritando la función *plt.title()* y, poniendo como argumento, el mismo título. Por ejemplo, podemos escribir:

```
plt.title('Aprobados vs cursos')
```

Haciendo el cambio de orientación y añadiendo el título, el nuevo gráfico puede quedar así:



## 3.2 SCATTERPLOTS

Un gráfico de dispersión (*scatter plot*, en inglés) muestra los puntos que relacionan los valores del eje *x* con los valores del eje *y*. Si, por ejemplo, cogemos los datos de alumnos aprobados y aprobadas respecto a los cursos, ¿cómo podemos generar un gráfico de dispersión? Si cambiamos la función *plot()* por *scatter()*, ya lo tendremos. Escribimos este código:

```
cursos = ['1415', '1516', '1617', '1718', '1819']

aprovados = [22,25,28,29,31]

dades = plt.scatter(cursos,aprovados)

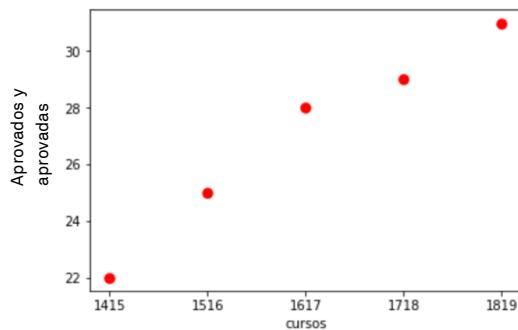
plt.xlabel('cursos')

plt.ylabel('aprovados y aprobadas')

plt.setp(dades, color='red', linewidth=2.5)

plt.show()
```

La función *plt.scatter* recibe los parámetros que forman los valor del eje *x* (cursos) y *y* (personas aprobadas). El gráfico que obtenemos, cuando ejecutamos el código, es:



Si suponemos que tenemos más de un grupo haciendo la misma asignatura, por ejemplo, tres grupos, podemos visualizar el número de personas aprobadas por año de estos tres grupos en el mismo gráfico. Lo haremos así:

```
cursos = ['1415', '1516', '1617', '1718', '1819']

aprovados1 = [22,25,28,29,31]

aprovados2 = [15,20,37,24,30]

aprovados3 = [20,12,22,30,25]

plt.scatter(cursos,aprovados1, color='yellow')

plt.scatter(cursos,aprovados2, color='red')

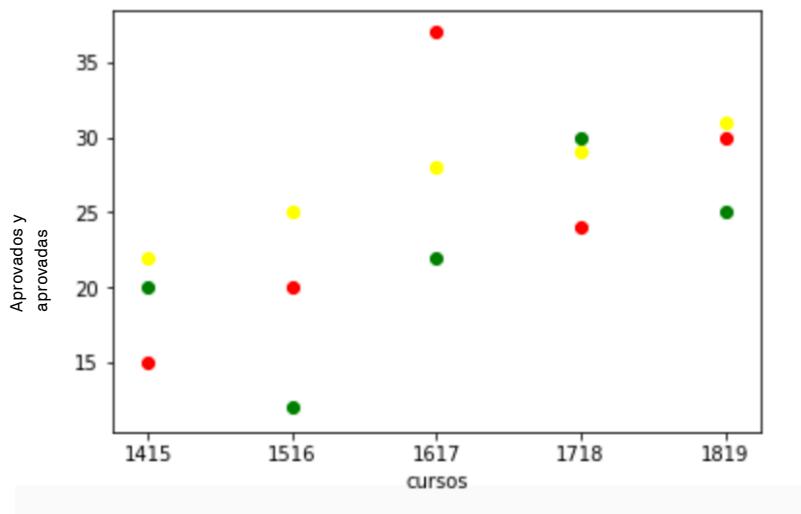
plt.scatter(cursos,aprovados3, color='green')

plt.xlabel('cursos')

plt.ylabel('aprovados y aprobadas')

plt.show()
```

Los y las alumnas aprobadas del grupo 1 (aprovados1) se representarán de color amarillo, los del grupo 2 de color rojo y los del grupo 3 de color verde.



Vamos, ahora, a trabajar con una tabla de datos más grande. Haz clic [aquí](#) para descargarte el fichero en formato CSV. Esta tabla es un ejemplo de una empresa de seguros. Después, escribe este código:

```
file_path = ('insurance.csv')

insurance_data = pd.read_csv(file_path)

print(insurance_data.head(10))
```

Verás las diez primeras observaciones de la tabla. Vamos a ver la relación entre los datos *bmi* y *charges*:

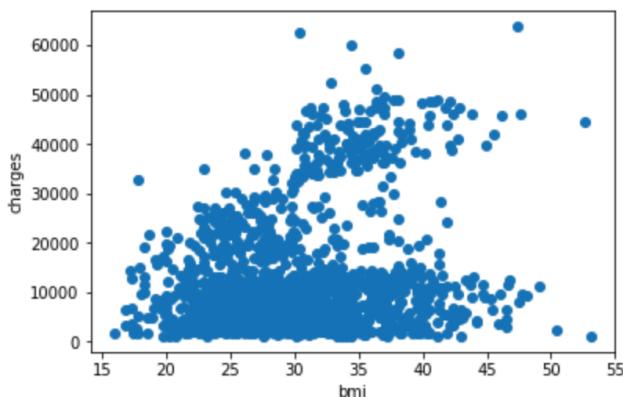
```
plt.scatter(insurance_data['bmi'], insurance_data['charges'])

plt.xlabel('bmi')

plt.ylabel('charges')

plt.show()
```

Este es el gráfico que obtenemos:



Nota: *bmi* es el valor de *Body Mass Index* y *charges* es el importe de riesgo del seguro.

Fíjate que, en el código, la lista de valores del eje *x* y el eje *y* están dentro de *insurance\_data['bmi']*, *insurance\_data['charges']*.

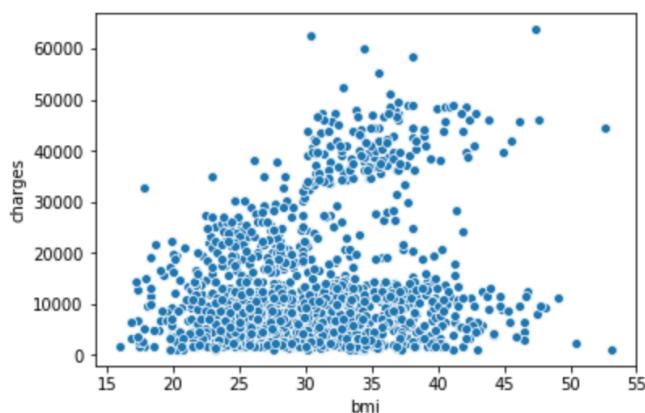
A continuación, utilizaremos otra librería basada en *Matplotlib* que se llama *Seaborn*. Para cargar esta librería y poderla usar, tenemos que importarla.

```
import seaborn as sns
```

Podemos generar un gráfico de dispersión con los mismos datos que en el caso anterior:

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'])
```

El gráfico que obtenemos es el siguiente:

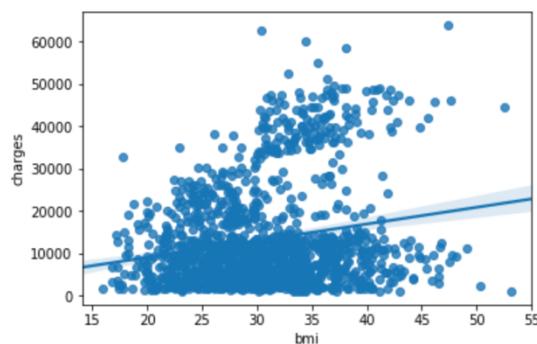


Fíjate que los puntos tienen más definición y que no hemos tenido que declarar el nombre de los ejes. La misma función `scatterplot()` ya los ha generado.

Vamos a ver, ahora, si hay una correlación entre los valores *bmi* y el precio de la póliza. Para hacer esto, tenemos que usar otra función llamada `regplot()`:

```
sns.regplot(x=insurance_data['bmi'], y=insurance_data['charges'])
```

El gráfico que obtenemos es el siguiente:



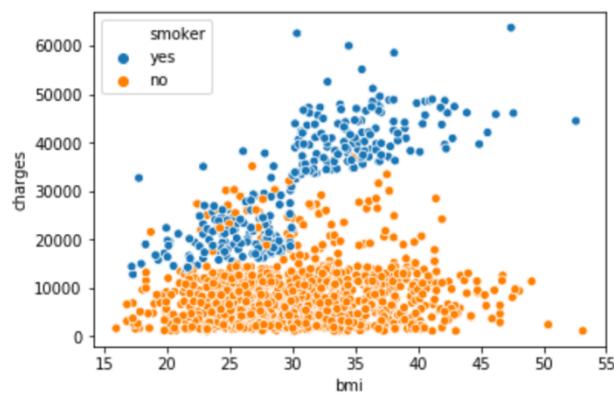
Según el gráfico, efectivamente hay una correlación positiva. A medida que el valor *bmi* de una persona aumenta, también crece el precio.

Si te fijas en los datos de la tabla, verás que hay una columna llamada *smoke*, que

indica si aquella persona es fumadora o no. Ahora haremos que los puntos del gráfico de arriba tengan un color u otro, dependiendo de si la persona es fumadora o no. Añadimos un nuevo parámetro a la función `scatterplot()`:

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue=insurance_data['smoker'])
```

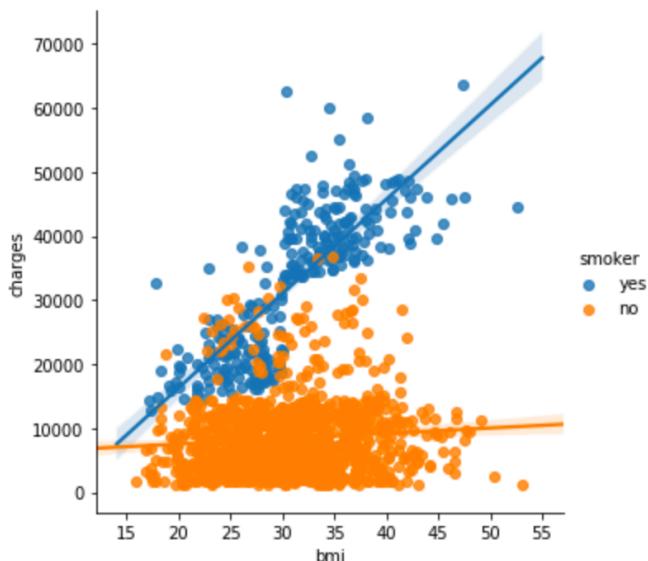
El gráfico que obtenemos es el siguiente:



Aquí vemos que, claramente, si la persona es fumadora, el precio de su seguro será más alto. Y, si queremos ver si la relación entre bmi y charges es diferente, en función de si la persona es fumadora o no, tenemos que ejecutar este código.

```
sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
```

El gráfico generado por la función `lmplot()` es el siguiente:



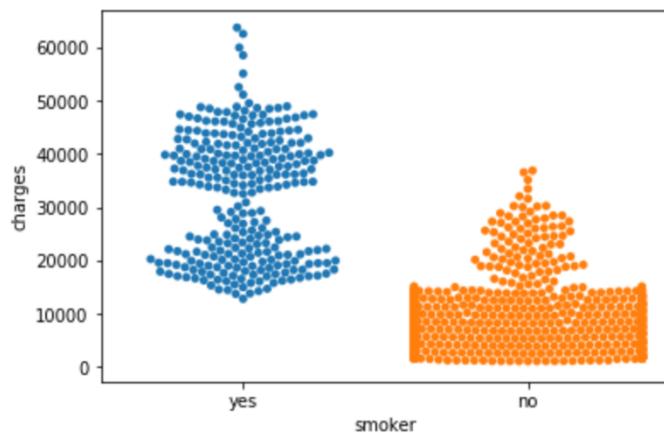
Efectivamente, vemos que, para una persona fumadora, el precio de su póliza aumenta considerablemente cuando tiene un *bmi* más alto, mientras que las personas no fumadoras tienen un aumento mucho más pequeño.

Hasta ahora, hemos hecho gráficos de dispersión entre dos variables numéricas, pero también podemos hacer que una variable sea categórica (no numérica). Por ejemplo, podemos ver la dispersión de precios de las pólizas, en función de si la persona es fumadora o no.

En este caso, usaremos la función *swarmplot()*. Escribiremos el código siguiente:

```
sns.swarmplot(x=insurance_data['smoker'], y=insurance_data['charges'])
```

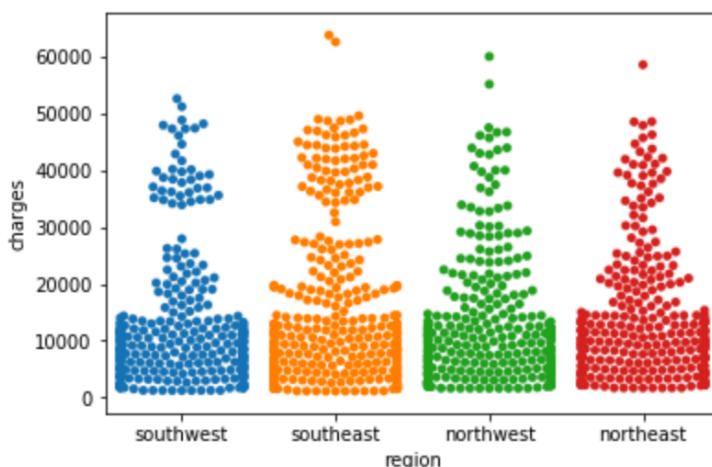
El gráfico que genera la función es este:



Este gráfico nos enseña que la gran mayoría de personas no fumadoras tienen un precio inferior al de las personas fumadoras.

Si ahora queremos ver como son los precios, dependiendo de la región, tenemos que escribir el código siguiente:

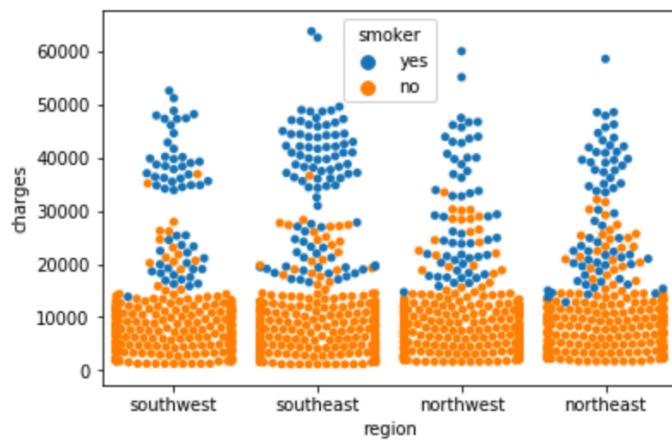
```
sns.swarmplot(x=insurance_data['region'], y=insurance_data['charges'])
```



Con el gráfico generado, vemos la dispersión de precios agrupados en las cuatro regiones que tiene la tabla. En principio, la mayoría de la población no presenta diferentes precios, según su región. Y ¿qué sucede, si añadimos al análisis la variable *smoke*? Para saberlo, tendremos que añadir a la función *swarmplot* el parámetro *hue* de este modo:

```
sns.swarmplot(x=insurance_data['region'],y=insurance_data['charges'],
hue=insurance_data['smoker'])
```

Y visualizamos este gráfico:



Aquí, otra vez, comprobamos que el factor más determinante para ver como varía el precio de un seguro es saber si la persona es fumadora o no.

¡Sigamos!

### 3.3 EJEMPLO PARA GENERAR LOS GRÁFICOS APRENDIDOS

¡Hola!

Repasamos lo que hemos hecho hasta ahora y descubrimos alguna herramienta nueva.

Como puedes ver, tenemos un proyecto nuevo con los módulos que necesitaremos. Tú ya sabes cómo crear un proyecto y también cómo importar los módulos, así que ya podemos empezar.

Primero, cargamos la base de datos que tiene por nombre *insurance.csv* con la que hemos trabajado. Esta base de datos contiene datos de una muestra de personas sobre su edad, sexo, bmi (*Body Mass Index*), si tiene hijos o no, si es fumadora o no, la región donde vive y el precio de su póliza de seguro.

```
file_path = ('insurance.csv')  
  
insurance_data = pd.read_csv(file_path)
```

Obtendremos un resumen de la información más importante que contiene, haciendo uso de la función *describe()*. Añadimos este código a la celda:

```
insurance_data.describe()
```

Y ya podemos ejecutarlo.

Con los datos que nos devuelve la función *describe()*, podemos ver que la edad media es de 39,20 años, siendo la edad máxima 64 años y la mínima 18. En cuanto al dato *bmi*, la media es de 30,66, siendo el valor más grande 39,86 y el más pequeño 13,40. Las personas de esta tabla tienen una media de 1,09 hijos o hijas, siendo 5 y 0 el valor máximo y mínimo respectivamente. Por último, podemos ver que el precio medio es 2,270, el valor máximo es 63.770 y el precio mínimo 1.121.

Este primer análisis ya nos da mucha información sobre los datos del estudio.

Veamos un histograma que nos enseñe cuántas observaciones tienen 0, 1, 2, 3, 4 o 5 hijos o hijas. Lo escribiremos así:

```
insurance_data['children'].hist(bins = 50, width = 0.25)
```

Vemos que mucha gente no tiene hijos ni hijas, y muy poca tiene 4 o 5.

Visualizamos, ahora, los puntos de dispersión que relacionan los *bmi* y los precios, y vemos la diferencia en función del número de hijos o hijas. Este es el código.

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue =  
insurance_data['children'])
```

El gráfico es bastante visual, con diferentes colores, puesto que hemos impuesto que los puntos alcancen un color diferente, en función del número de hijos o hijas que tengan. Aquí, la dependencia queda marcada en el código que hay dentro de la función *scatterplot()* con el parámetro *hue*. En cualquier caso, la verdad es que la información que podemos extraer no es

de calidad, porque no se puede ver ninguna tendencia. Vamos, pues, a hacer uso de la función *lmplot()*, para generar una recta de regresión para cada grupo, en función del número de hijos o hijas.

El código que tenemos que escribir es:

```
sns.lmplot(x="bmi", y="charges", hue="children", data=insurance_data)
```

Con esta gráfica, lo que vemos claramente es que, para todos los casos (excepto para aquellos en los que el número de hijos o hijas es 5), los precios aumentan a medida que aumenta el *bmi*. Es curioso ver cómo, para el caso de 5 hijos o hijas, el precio disminuye con un *bmi* mayor. Posiblemente, la realidad será diferente, o no, pero ciertamente el caso requiere un estudio más profundo. En cualquier caso, la zona difusa que hay en torno a cada línea indica suficiente margen de error como para que la correlación sea positiva.

## 3.4 IDEAS CLAVE: VISUALIZACIÓN DE DATOS CON MATPLOTLIB

En este apartado, hemos aprendido a extraer información de los datos mediante representaciones gráficas, que nos ayudan a visualizar tendencias y patrones con solo un vistazo.

Concretamente, hemos trabajado con gráficos de línea, de barra y de dispersión.

**Los gráficos de línea** muestran una línea continua que enlaza todos los puntos que relacionan los valores del eje *x* y el valor del eje *y* correspondientes.

**Los gráficos de barras** son aquellos que utilizan barras, que representan la altura que tiene un punto respecto del eje *x*.

**Los gráficos de dispersión** muestran los puntos y las coordenadas formadas por el valor en el eje *x* y el valor en el eje *y* correspondiente.

Para construir los diferentes tipos de gráficos, hemos trabajado con funciones de los módulos *Matplotlib.pyplot* y *Seaborn*.

El módulo Pyplot proporciona funciones muy útiles para generar los diferentes gráficos. Para trabajar con gráficos de línea, usaremos *plot()*, de barra *bar()* y de dispersión *scatter()*. Estas funciones incorporan diferentes parámetros para configurar los gráficos. Por ejemplo, *xlabel()* da nombre al eje *x*, *ylabel()* al eje *y*, *title()* sirve para posar un título, *color()* proporciona el color, *linewidth()* la anchura de la línea, etc.

## 4 ANÁLISIS DE DATOS CON PANDA

Cada proyecto, sea grande o pequeño, incorporará una base de datos con la que tendremos que trabajar. Independientemente del volumen, esta base de datos, probablemente, requerirá una serie de acciones para descubrir la información relevante y útil que contiene.

En este módulo aprenderemos a trabajar y procesar bases de datos. Sabremos cómo crear una desde cero y, después, leer su contenido y escribirle de nuevo. A continuación, haremos frente al primer bloque de operaciones básicas de procesamiento de base de datos, que estará formado por las acciones de seleccionar, filtrar e indexar. A continuación, aprenderemos a ordenar y agrupar datos y a gestionar datos no disponibles.

Finalmente, pondremos a prueba los nuevos conocimientos con un ejercicio práctico. Ciertamente, la habilidad al saber gestionar bases de datos se adquiere, primero, con una buena base teórica y, después, necesariamente, con ejercicios prácticos. ¡Empecemos!

### 4.1 CREAR, LEER Y ESCRIBIR UNA TABLA DE DATOS

Empezaremos aprendiendo a crear una base de datos desde cero. Lo primero que tenemos que hacer, una vez ya tenemos el proyecto iniciado con Jupyter Notebook, es importar la librería Pandas. Lo haremos del mismo modo que en los módulos anteriores:

```
import pandas as pd
```

Recuerda que una base de datos está formada por filas y columnas. Construiremos una que, inicialmente, solo tiene una fila y dos columnas. La primera columna indica que hay 30 manzanas y la segunda que hay 20 fresas. Este es el código:

```
frutas = pd.DataFrame([[30,20]], columns=['Manzanas', 'Fresas'])
```

```
frutas
```

La tabla que vemos una vez ejecutamos el código es:

	Manzanas	Fresas
0	30	20

Lo que hace el código que hemos escrito es generar un objeto (una base de datos que denominamos frutas) a partir de su objeto superior DataFrame, logrando una serie de parámetros que se especifican dentro del paréntesis. El primer parámetro indica los valores

de las filas y el segundo el nombre que tendrá cada columna. Tanto el valor de las filas como el nombre de las columnas se introducen haciendo uso de listas.

Avanzamos un poco más y añadimos, ahora, más filas a esta base de datos. Escribimos este código:

```
frutas_compradores=pd.DataFrame([[30,20],[28,14]],indice=['Miquel','Julia'],columns=['Manzanas', 'Fresas'])
```

*frutas\_compradores*

Si nos fijamos en el primer parámetro, allá donde definimos el valor de las filas, lo que hemos hecho es añadirle otra lista (otra fila). Después, hemos definido otro parámetro denominado *indice* que sirve para dar nombre a las filas.

Manzanas Fresas		
Miquel	30	20
Julia	28	14

Hasta aquí hemos construido una base de datos a partir de valores agrupados en filas. También podemos crear una introduciéndole los datos agrupados en columnas. Hagámoslo:

```
frutas_compradores_2 = pd.DataFrame({'Manzanas':[30,28],'Fresas':[20,14]},indice=['Miquel', 'Julia'])
```

*frutas\_compradores\_2*

Hemos creado una nueva variable donde guardamos la base de datos que se ha introducido con los valores agrupados en columnas. Esencialmente, el código que hemos escrito tiene que generar la misma base de datos que en el caso anterior.

Manzanas Fresas		
Miquel	30	20
Julia	28	14

Efectivamente, vemos que la nueva base de datos presenta los mismos valores que la anterior. Si estudiamos el código, vemos que las columnas se han introducido haciendo uso

de diccionarios, que tienen, como clave, el nombre de la columna y, como valores, los datos que forman esta columna. Fíjémonos que, para cada columna, los datos se introducen mediante una lista.

A pesar de que las dos bases de datos presentan la misma información, no son realmente la misma base de datos. De hecho, son objetos diferentes porque cada uno está guardado en un espacio de memoria diferente (en dos variables diferentes). Comprobémoslo:

```
print(frutas_compradores_2 is frutas_compradores)
```

El resultado que nos devuelve Python es *False*, lo que confirma nuestra sospecha. Para hacer que las dos bases de datos sean realmente el mismo objeto, lo que tenemos que hacer es que apunten al mismo espacio de memoria. Así que necesitamos escribir esta línea de código:

```
frutas_compradores = frutas_compradores_2
```

Comprobemos, ahora, si son el mismo objeto:

```
print(frutas_compradores_2 is frutas_compradores)
```

Confirmado, ahora son lo mismo. Python nos devuelve *True* y, por lo tanto, podemos afirmar que sí que lo son. Esto quiere decir que, si hacemos un cambio en cualquiera de las dos bases de datos (*frutas\_compradores* o *frutas\_compradores\_2*), este cambio también se aplicará a la otra base de datos porque se han convertido en el mismo objeto. Comprobémoslo:

En primer lugar, asignamos un valor nuevo a la primera posición de la columna que hemos denominado *Manzanas*, de la primera base de datos creada:

```
frutas_compradores['Manzanas'][0] = 15
```

Para asignar este valor nuevo a la posición deseada, tenemos que llamar la base de datos que queremos cambiar, asignarle la columna (*Manzanas*) y fijar la posición (0).

Comprobamos que el cambio se ha materializado correctamente:

```
print(frutas_compradores['Manzanas'][0])
```

Python nos devuelve 15, lo que quiere decir que el cambio se ha hecho. Y, ahora, solo hay que ver si el cambio también se puede observar si usamos el nombre de la variable *frutas\_compradores\_2*:

```
print(frutas_compradores_2['Manzanas'][0])
```

Sí, también obtenemos 15.

Si el cambio lo hacemos empezando por la segunda base de datos creada, este también se podrá ver cuando hagamos uso del nombre de la variable que hace referencia a la primera base de datos creada. Escribimos este bloque de código:

```
frutas_compradores_2['Manzanas'][0]=2
print(frutas_compradores_2['Manzanas'][0])
print(frutas_compradores['Manzanas'][0])
```

Correcto. En los dos casos, vemos que el valor de la primera posición de la columna Manzanas ahora es 2.

Si queremos ver cómo queda el objeto final y queremos comprobar que, haciendo uso de cualquier nombre de variable, nos imprime la misma tabla, solo tenemos que denominar el nombre de cada variable:

```
frutas_compradores
```

Manzanas Fresas		
<b>Miquel</b>	2	20
<b>Julia</b>	28	14

```
frutas_compradores_2
```

Manzanas Fresas		
<b>Miquel</b>	2	20
<b>Julia</b>	28	14

Efectivamente, vemos que el cambio se ha materializado en la primera posición de la columna *Manzanas*.

Nota: recuerda que la primera posición en una lista se identifica con un 0 y no con un 1.

## 4.2 INDEXAR, SELECCIONAR Y FILTRAR

Seguimos avanzando y profundizando en las bases de datos.

A continuación, crea un nuevo proyecto con Jupyter Notebook y descárgate esta base de datos desde este [enlace](#). Guárdala en la misma carpeta donde tienes el proyecto nuevo.

Primero, cargamos la librería *Pandas* y le asignamos el nombre *pd*:

```
import pandas as pd
```

Importamos la base de datos y la guardamos en una variable denominada *data\_base*. Después, vemos las cinco primeras observaciones:

```
data_base = pd.read_csv('insurance.csv')
```

```
data_base.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Entonces, nos aparecen todas las columnas con sus cinco primeras observaciones, pero no sabemos cómo es de grande esta base de datos. Si queremos imprimir solo un determinado número de filas, pero queremos, además, ver el número total de filas, lo tendremos que hacer así:

```
pd.set_option("display.max_rows", 5)
```

```
data_base
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.90	0	yes	southwest	16884.9240
1	18	male	33.77	1	no	southeast	1725.5523
...	...	...	...	...	...	...	...
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

1338 rows × 7 columns

Aquí, nos aparecen los títulos de las columnas y cuatro filas con sus datos. Debajo de la tabla, vemos que está formada por 1.338 filas (observaciones) y 7 columnas (atributos o variables).

Si queremos específicamente la información que contiene una variable determinada, solo tendremos que introducir este código:

```
data_base['region']
```

Python nos mostrará esta información:

```
0      southwest
1      southeast
...
1336    southwest
1337    northwest
Name: region, Length: 1338, dtype: object
```

Ahora, vemos solo información correspondiente a la variable *region*.

Si quieras ver más observaciones, tendrás que cambiar el parámetro de la función *set\_option()*. En vez de cinco, puedes poner diez. Si haces el cambio, tendrás que volver a ejecutar los códigos, para ver más filas en la base de datos.

Nota: tú puedes escoger otra variable, como puede ser *smoker*, *sex*, *age*, etc.

Supongamos, ahora, que quieras saber el valor más grande que tiene la columna *bmi*. Esto lo puedes saber, llamando la función *describe()*. ¿Te acuerdas? O bien seleccionando solo la columna *bmi* y haciendo uso de la función *max()*:

```
data_base['bmi'].max()
```

En ambos casos, el resultado es 53,13.

Y ¿cómo lo podemos hacer para seleccionar solo los 30 primeros valores de una columna? Pues así:

```
bmi_30 = data_base['bmi'][0:30]
```

```
bmi_30
```

Ahora, miramos a ver cuál es el valor máximo de los 30 primeros valores de la columna *bmi*:

```
bmi_30.max()
```

El valor obtenido es 42,13. El valor 53,13, pues, no está dentro de los 30 primeros valores.

Vamos a indexar, ahora, no solo valores de una columna, sino filas y columnas conjuntamente. Si queremos, por ejemplo, trabajar con las diez primeras filas y las dos primeras columnas, ¿cómo lo hacemos? Necesitamos trabajar con el operador *iloc* y lo haremos así:

```
data_base_2 = data_base.iloc[0:10,0:2]
```

El que hemos hecho, aquí, es decir que queremos las diez primeras filas (0:10) y las dos primeras columnas (0:2) de *data\_base*. Hemos guardado esta tabla reducida en un nuevo espacio de memoria, a través de un nombre nuevo que es *data\_base\_2*. Esto es un objeto nuevo, una nueva base de datos. Si llamamos esta nueva base de datos, obtenemos:

```
data_base_2
```

Podemos, también, en vez de seleccionar un intervalo de filas o columnas, seleccionar

	age	sex
0	19	female
1	18	male
2	28	male
3	33	male
4	32	male
5	31	female
6	46	female
7	37	female
8	37	male
9	60	female

específicamente unas determinadas filas o columnas:

```
data_base_3 = data_base.iloc[[0,5,6],[0,4]]
```

```
data_base_3
```

Lo que hemos hecho aquí es obtener las filas 0,5 y 6 y las columnas 0 y 4. Después, las hemos guardado como otra base de datos reducida en otro espacio de memoria.

Para las columnas, podemos usar sus nombres propios para seleccionarlas. En este caso, tenemos que usar el operador *loc*. Observemos este ejemplo:

```
data_base_4 = data_base.loc[0:8,['children','age']]
```

```
data_base_4
```

Lo que hemos hecho es seleccionar las ocho primeras filas de las columnas *children* y *age*.

	children	age
0	0	19
1	1	18
2	3	28
3	0	33
4	0	32
5	0	31
6	1	46
7	3	37
8	2	37

Los datos que se muestran son los siguientes:

Damos un paso adelante y vamos a crear tablas reducidas a partir de la tabla madre, no haciendo uso de los índices de filas o columnas, sino de criterios condicionales. Imagina que solo queremos aquellas observaciones que se encuentren en la región *southwest*. Este es el código que necesitamos:

```
data_base_southwest = data_base.loc[data_base.region == 'southwest']
```

Y queremos sacar, por pantalla, solo las siete primeras filas y las dos primeras columnas:

```
data_base_southwest.iloc[0:7,0:2]
```

La tabla que se ve por pantalla es esta:

	age	sex
0	19	female
12	23	male
15	19	male
18	56	male
19	30	male
21	30	female
29	31	male

Fíjate que la numeración de filas hace referencia a las filas 0, 12, 15, 18, 19, 21 y 29, que son las siete primeras filas que pertenecen en la región *southwest* de la tabla madre *data\_base*.

Si queremos ver aquellas observaciones de la base de datos principal que tengan un *bmi* entre 35 y 45, tendremos que declarar dos condicionales:

```
data_base.loc[(data_base.bmi >= 35) & (data_base.bmi <= 45)].describe()
```

Hemos aplicado la función `describe()`, para ver qué valores máximos y mínimos tiene la columna `bmi` de esta tabla reducida:

	age	bmi	children	charges
<b>count</b>	296.000000	296.000000	296.000000	296.000000
<b>mean</b>	41.679054	38.301976	1.027027	16913.681515
<b>std</b>	14.550498	2.427277	1.149479	15367.757351
<b>min</b>	18.000000	35.090000	0.000000	1141.445100
<b>25%</b>	30.000000	36.297500	0.000000	5745.351188
<b>50%</b>	43.500000	37.707500	1.000000	10979.853800
<b>75%</b>	54.000000	39.840000	2.000000	26781.395215
<b>max</b>	64.000000	44.880000	5.000000	58571.074480

Efectivamente, vemos que, en esta tabla reducida, solo tenemos aquellas filas que tienen un valor de `bmi` entre 35 y 45.

Por último, vamos a crear una columna nueva. Sus valores dependerán del valor de `bmi` de cada fila. El código es el siguiente:

```
approved = []
for bmi in data_base['bmi']:
    if bmi < 45:
        approved.append(True)
    else:
        approved.append(False)
data_base['approved'] = approved
data_base.head()
```

Lo que estamos haciendo es dar un valor a cada fila de la columna `approved` de `True` o `False`, según si el valor de `bmi` de la misma fila es superior o inferior a 45. Veamos los cinco primeros registros con la nueva columna:

	age	sex	bmi	children	smoker	region	charges	approved
0	19	female	27.900	0	yes	southwest	16884.92400	True
1	18	male	33.770	1	no	southeast	1725.55230	True
2	28	male	33.000	3	no	southeast	4449.46200	True
3	33	male	22.705	0	no	northwest	21984.47061	True
4	32	male	28.880	0	no	northwest	3866.85520	True

## 4.3 ORDENAR Y AGRUPAR DATOS NO DISPONIBLES

En este artículo, empezaremos descubriendo una nueva función que nos permitirá generar una serie de datos agrupados en torno a los valores de una columna. El primer paso, como es habitual, es cargar la librería y crear un nuevo objeto con la base de datos que hemos usado en la artículo anterior:

```
import pandas as pd

data_frame = pd.read_csv('insurance.csv')
```

Vamos a ver cómo se distribuyen los valores de *region*:

```
data_frame.groupby('region').size()
```

```
region
northeast    324
northwest    325
southeast    364
southwest    325
dtype: int64
```

Aquí vemos cuántas observaciones tienen cada una de las diferentes regiones de la tabla madre.

Si queremos ver cuál es el valor máximo de *bmi* para cada región, hacemos:

```
data_frame.groupby('region')['bmi'].max()
```

```
region
northeast    48.07
northwest    42.94
southeast    53.13
southwest    47.60
Name: bmi, dtype: float64
```

Y, si queremos saber la media de *bmi* para cada región, tecleamos:

```
data_frame.groupby('region')['bmi'].mean()
```

```
region
northeast    29.173503
northwest    29.199785
southeast    33.355989
southwest    30.596615
Name: bmi, dtype: float64
```

Aquí, vemos que la media de valores de *bmi* más elevada la encontramos en la región *southeast* y la más baja en la *northeast*.

Tenemos la opción de generar la lista de estas medias de manera ordenada. Lo podemos hacer con la función *sorted()*. El código sería este:

```
sorted(data_frame.groupby('region')['bmi'].mean())
```

La lista que obtendríamos sería:

```
[29.17350308641976, 29.199784615384626, 30.59661538461538, 33.35598901098903]
```

Ahora, haremos una agrupación más compleja. Mostraremos las medias de los precios de las pólizas agrupadas, en un primer nivel, por la región y, en un segundo nivel, por la variable *smoker*. Este es el código:

```
data_frame.groupby(['region', 'smoker'])['charges'].mean()
```

region	smoker	
northeast	no	9165.531672
	yes	29673.536473
northwest	no	8556.463715
	yes	30192.003182
southeast	no	8032.216309
	yes	34844.996824
southwest	no	8019.284513
	yes	32269.063494

Name: charges, dtype: float64

Vemos claramente que, independientemente de la región, la media de precios siempre es más alta, si la persona es fumadora, que cuando no lo es.

Si ahora mantenemos la agrupación de la región, pero cambiamos la variable *smoker* por *sex* en el segundo nivel de la agrupación, necesitamos solo hacer este pequeño cambio:

```
data_frame.groupby(['region', 'sex'])['charges'].mean()
```

La tabla que obtenemos es:

```

region      sex
northeast   female    12953.203151
              male     13854.005374
northwest   female    12479.870397
              male     12354.119575
southeast   female    13499.669243
              male     15879.617173
southwest   female    11274.411264
              male     13412.883576
Name: charges, dtype: float64

```

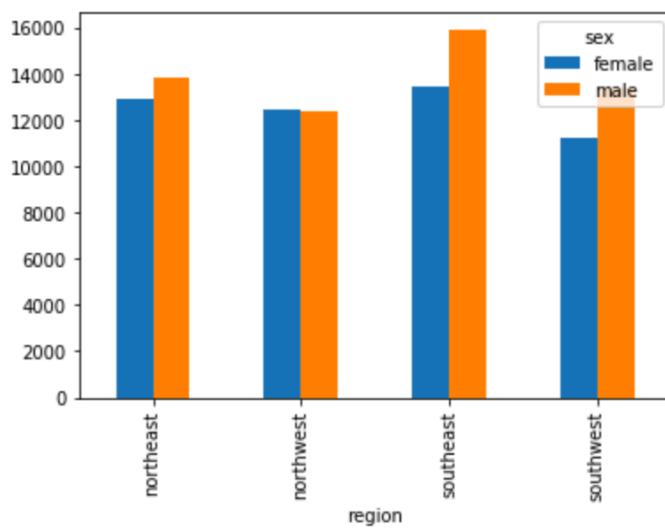
Vemos que, en todas las regiones, excepto en *southwest*, la media de precios es más alta para los hombres que para las mujeres.

Si asignamos esta agrupación a una nueva variable y aplicamos la función *plot.bar()*, obtendremos una gráfica de barras coherente con la agrupación:

```

group = data_frame.groupby(['region', 'sex'])['charges'].mean()
group.unstack(fill_value=0).plot.bar()

```



El gráfico muestra, de una manera mucho más visual, lo que la tabla de arriba nos decía.

Si, dada la base de datos *data\_frame*, queremos mostrarla de manera ordenada según los valores de una variable determinada, solo tenemos que usar la función *sort\_values()*. Lo haremos así:

```
data_frame.sort_values(['age','bmi'])
```

Si queremos mostrar solo los primeros 12 registros de este orden, tendremos que aplicar la función *head()* y especificar el número de registros que queremos mostrar:

```
data_frame.sort_values(['age','bmi']).head(12)
```

La tabla que obtenemos, pues, será esta:

El primer criterio para ordenar las filas corresponde al valor de la edad (*age*) y el segundo corresponde a *bmi*. El código, primero, ordena por edades y, en el caso de que haya filas con la misma edad, el código aplica un segundo criterio, el *bmi*. Por defecto, el orden se aplica siempre de manera ascendente, es decir, de valor más pequeño a más grande. Si esto lo queremos invertir, tendremos que añadir otro parámetro de este modo:

```
data_frame.sort_values(['age','bmi'], ascending=[False, False]).head(12)
```

	age	sex	bmi	children	smoker	region	charges
534	64	male	40.480	0	no	southeast	13831.11520
768	64	female	39.700	0	no	southwest	14319.03100
199	64	female	39.330	0	no	northeast	14901.51670
418	64	male	39.160	1	no	southeast	14418.28040
603	64	female	39.050	3	no	southeast	16085.12750
635	64	male	38.190	0	no	northeast	14410.93210
752	64	male	37.905	0	no	northwest	14210.53595
1241	64	male	36.960	2	yes	southeast	49577.66240
801	64	female	35.970	0	no	southeast	14313.84630
335	64	male	34.500	0	no	southwest	13822.80300
420	64	male	33.880	0	yes	southeast	46889.26120
328	64	female	33.800	1	yes	southwest	47928.03000

Ahora, la tabla que obtenemos es la siguiente:

El parámetro *ascendiendo*, por defecto, alcanza el valor *True* y es por eso que las tablas siempre muestran un orden ascendente. Si especificamos que el *ascending* tiene que tener un valor *False*, obtendremos una tabla con orden descendente, como esta última que hemos obtenido.

Hasta ahora, hemos supuesto que todas las celdas de una tabla tenían valores. La realidad, en cambio, siempre es diferente y esta nos presentará, a menudo, tablas que tienen celdas sin valores. Estos valores se denominan *Nas*.

Una de las funciones más utilizadas del paquete Pandas, para ver si tenemos *Nas* o no, es la función *isna()*. Esta función devuelve, como resultado, *False*, cuando el valor existe y devuelve *True*, cuando el valor es *Na*. Por ejemplo, si queremos saber si entre los registros 230 y 238 de la columna *smoker* hay *Nas* o no, necesitamos escribir este código:

```
data_frame['smoker'][230:239].isna()
```

	age	sex	bmi	children	smoker	region	charges
172	18	male	15.960	0	no	northeast	1694.79640
250	18	male	17.290	2	yes	northeast	12829.45510
359	18	female	20.790	0	no	southeast	1607.51010
1212	18	male	21.470	0	no	northeast	1702.45530
1033	18	male	21.565	0	yes	northeast	13747.87235
1282	18	female	21.660	0	yes	northeast	14283.45940
1080	18	male	21.780	2	no	southeast	11884.04858
295	18	male	22.990	0	no	northeast	1704.56810
1041	18	male	23.085	0	no	northeast	1704.70015
940	18	male	23.210	0	no	southeast	1121.87390
1023	18	male	23.320	1	no	southeast	1711.02680
121	18	male	23.750	0	no	northeast	1705.62450

La lista de valores que obtenemos es:

```
230    False
231    False
232    False
233    False
234    False
235    False
236    False
237    False
238    False
Name: smoker, dtype: bool
```

Observando los resultados, podemos concluir que, en este intervalo de valores, no tenemos ningún *Na*, ya que todos los valores que devuelve la función *isna()* son *False*.

Vamos, ahora, a copiar la base de datos *data\_frame* en una nueva variable:

```
data_frame2 = data_frame[:]
```

Importante: *data\_frame2* y *data\_frame* no son el mismo objeto. No hemos apuntado realmente *data\_frame* en la nueva variable, puesto que hemos usado una copia con el operador `[:]`.

```
data_frame2['smoker'][230] = None
```

```
data_frame2['smoker'][230:239].isna()
```

La tabla que obtenemos ahora sí que tiene un *Na*, según indica la función *isna()*:

```
230    True
231    False
232    False
233    False
234    False
235    False
236    False
237    False
238    False
Name: smoker, dtype: bool
```

La función *notna()* funciona justo al contrario. Allá donde hay un *Na*, devuelve un *False* y, donde no hay, devuelve un *True*. ¿Lo vemos?

```
data_frame2['smoker'][230:239].notna()
```

```

230    False
231    True
232    True
233    True
234    True
235    True
236    True
237    True
238    True
Name: smoker, dtype: bool

```

¡Sigamos!

## 4.4 REPASO DEL TEMARIO CON EJERCICIO PRÁCTICO

¡Hola a todo el mundo!

Repasemos lo que hemos hecho hasta ahora con unos ejercicios.

Abre un proyecto nuevo con Jupyter Notebook e introduce este código que aparece en pantalla. Primero, carga la librería *Pandas*, después, crea una base de datos que guardamos en una variable que podemos denominar, por ejemplo, *notas*.

Bien,

```

In [1]: 1 import pandas as pd
In [2]: 1 notas = pd.DataFrame([[6.5, 6.8, 5.8, 7.9, 8.8, 'Barcelona'], [6.8, 7., 6.5, 8.9, 9.2, 'Girona'],
                           2 [7.8, 6.2, 6.5, 4.1, 8.0, 'Tarragona'], [6.9, 7.2, 4.7, 8.8, 6.2, 'Lleida']],
                           3 columns=['Historia', 'Matemáticas', 'Física', 'Inglés', 'Biología', 'Ciudad'],
                           4 index = ['Joan', 'Laura', 'Enric', 'Georgina'])
Out[2]:      Historia Matemáticas Física Inglés Biología Ciudad
Joan        6.5          6.8   5.8    7.9    8.8  Barcelona
Laura       6.8          7.0   6.5    8.9    9.2   Girona
Enric       7.8          6.2   6.5    4.1    8.0  Tarragona
Georgina    6.9          7.2   4.7    8.8    6.2    Lleida

```

vemos que se trata de una tabla que recoge las notas de cuatro estudiantes y su ciudad de procedencia. Para empezar, revisemos cómo podemos extraer la nota que ha sacado Laura en Física y la ciudad de Joan. ¿Sabes cómo hacerlo?

Solo hay que mencionar el nombre de la base de datos, seleccionar la columna del atributo, en este caso *física* o *ciudad* y, después, indicar la posición de la observación, es decir, la posición del estudiante del que queremos extraer la información:

---

In [3]:	1	<b>notas['Física'][1]</b>
	2	<b>notas['Ciudad'][0]</b>

---

¿Te has fijado que estamos trabajando con diccionarios? Sí, el nombre de la columna es la clave que da acceso a su lista de valores y, solo indicando la posición, obtenemos el dato que queremos.

Vamos, ahora, a hacer un filtro. Si queremos ver las observaciones (estudiantes) que han aprobado física, ¿cómo lo haríamos? Pues denominamos la base de datos y, dentro de los corchetes, introducimos la condición. Es fácil, ¿verdad?

In [4]: 1 notas [notas.Física > 5]

Si mostramos la tabla, veremos que Georgina es quien no ha aprobado física. ¿Y si ha sido un error y realmente sí que lo ha aprobado? Si su nota es de 7.0, ¿cómo la podemos cambiar? Pues bien, solo hay que acceder a su nota y cambiarla.

Out[4]:

	Historia	Matemáticas	Física	Inglés	Biología	Ciudad
<b>Joan</b>	6.5	6.8	5.8	7.9	8.8	Barcelona
<b>Laura</b>	6.8	7.0	6.5	8.9	9.2	Girona
<b>Enric</b>	7.8	6.2	6.5	4.1	8.0	Tarragona

▶ In [6]: 1 notas[ 'Física' ][3] = 5.0  
2 notas

Si mostramos la tabla de nuevo, veremos que el cambio se ha aplicado con éxito. Ahora nadie ha suspendido Física. ¡Y esto son buenas noticias!

Out[6]:

	Historia	Matemáticas	Física	Inglés	Biología	Ciudad
<b>Joan</b>	6.5	6.8	5.8	7.9	8.8	Barcelona
<b>Laura</b>	6.8	7.0	6.5	8.9	9.2	Girona
<b>Enric</b>	7.8	6.2	6.5	4.1	8.0	Tarragona
<b>Georgina</b>	6.9	7.2	5.0	8.8	6.2	Lleida

A continuación, extraemos una tabla reducida que esté formada por las notas de Biología de solo las chicas. La clave está en usar el operador *loc[]*.

```
In [6]: 1 notas.loc[['Laura', 'Georgina'], ['Biología']]
```

Biología	
Laura	9.2
Georgina	6.2

Por último, vamos a hacer algo un poco más difícil. Añadimos una columna que muestre las notas de cada estudiante y su desviación. ¡Construimos el código! Primero, tenemos que crear dos listas. Después, crear un ciclo *for* que nos permita recorrer la tabla fila por fila. Por eso usaremos el operador *iloc[]*. El operador formará una lista, a la que podremos aplicar la función *max()* para calcular la media, y el operador *std()*, para calcular la desviación. ¡Importante! Cada vez que apliquemos las funciones, el resultado se tiene que cargar a las listas correspondientes: la media a su lista y la desviación a la suya. Es importante dar nombres coherentes a las variables, para hacer que nuestro código sea fácil de entender, sin tener que añadirle comentarios.

```
In [12]: 1 mediana = []
2 desviación = []
3 for i in range(4):
4     mediana = notas.iloc[i, 0:5].mean()
5     desv = notas.iloc[i, 0:5].std()
6     mediana.append(mediana)
7     desviación.append(desv)
8 notas['Mediana'] = mediana
9 notas['Máxima'] = desviación
10
```

	Historia	Matemáticas	Física	Inglés	Biología	Ciudad	Medianas	Máxima
Joan	6.5	6.8	5.8	7.9	8.8	Barcelona	7.16	1.188697
Laura	6.8	7.0	6.5	8.9	9.2	Girona	7.68	1.267675
Enric	7.8	6.2	5.0	4.1	8.0	Tarragona	6.22	1.706458
Georgina	6.9	7.2	4.7	8.8	6.2	Lleida	6.76	1.494323

Es básico practicar para avanzar. Anímate y procura jugar un poco más con esta base de datos. Aquí, hemos visto varias cosas, pero seguro que puedes encontrar más acciones para hacer.

¡Suerte!

## 4.5 IDEAS CLAVE: ANÁLISIS DE DATOS CON PANDA

En este módulo hemos aprendido a trabajar con bases de datos y hemos visto como pueden ser de potentes, si sabemos cómo aplicar varias acciones destinadas al análisis de datos.

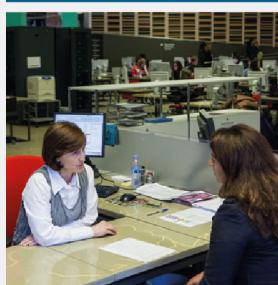
Hemos empezado a **construir una desde cero, haciendo uso de la clase DataFrame**. Después, hemos visto cómo leer y modificar bases de datos, accediendo a sus posiciones.

Aplicando los operadores `iloc[]` y `loc[]`, podemos **crear bases de datos más pequeñas**, partiendo de una base de datos principal.

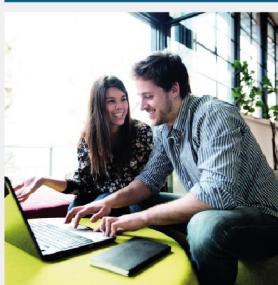
También hemos aprendido a **seleccionar filas, dependiendo de una determinada condición**, o más de una, lo que nos permite filtrar la base de datos en relación con los diferentes valores que pueden tener los atributos o variables.

Podemos **agrupar bases de datos** en torno a los valores de una o más variables. Una vez hayamos agrupado estos valores, es posible aplicar diferentes funciones como, por ejemplo, la media, el número de casos o las desviaciones de estas agrupaciones.

# Descubre todo lo que Barcelona Activa puede hacer por ti



Acompañamiento durante todo el proceso de búsqueda de empleo  
[barcelonactiva.cat/treball](http://barcelonactiva.cat/treball)



Apoyo en la puesta en marcha de tu idea de negocio  
[barcelonactiva.cat/emprendedoria](http://barcelonactiva.cat/emprendedoria)



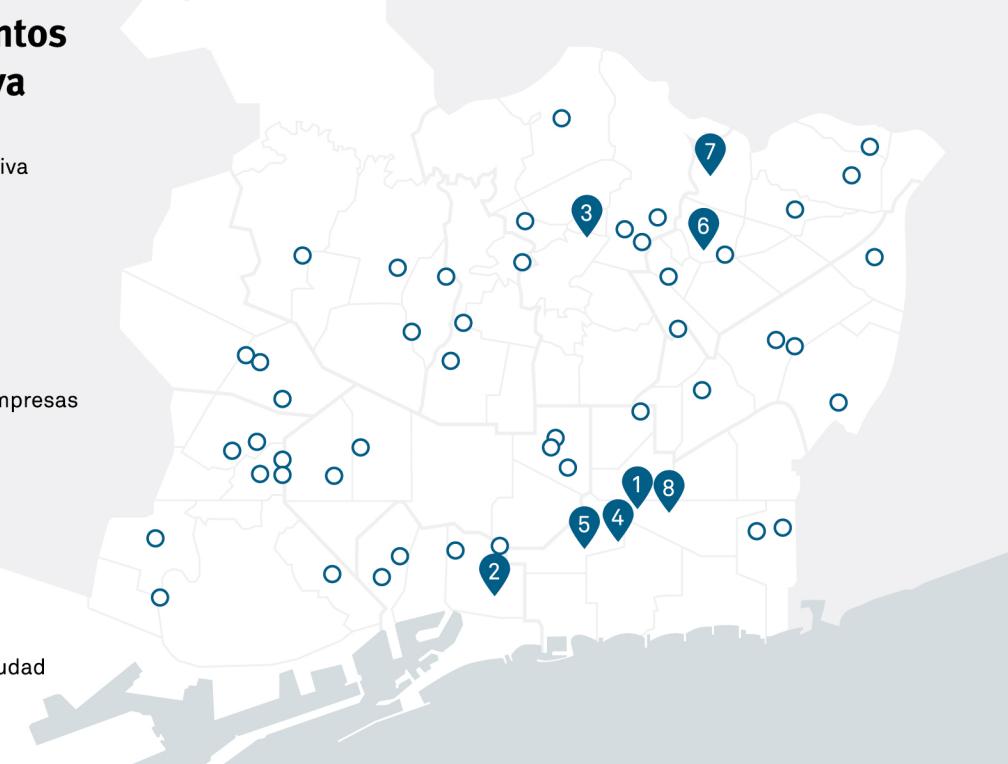
Servicios a las empresas e iniciativas socioempresariales  
[barcelonactiva.cat/empresas](http://barcelonactiva.cat/empresas)



Formación tecnológica y gratuita para la ciudadanía  
[barcelonactiva.cat/cibernarium](http://barcelonactiva.cat/cibernarium)

## Red de equipamientos de Barcelona Activa

- 1 Sede Central Barcelona Activa  
Porta 22  
Centro para la Iniciativa Emprendedora Glòries  
Incubadora Glòries
  - 2 Convent de Sant Agustí
  - 3 Ca n'Andalet
  - 4 Oficina de Atención a las Empresas Cibernàrium  
Incubadora MediaTIC
  - 5 Incubadora Almogàvers
  - 6 Parque Tecnológico
  - 7 Nou Barris Activa
  - 8 innoBA
- Puntos de atención en la ciudad



© Barcelona Activa  
Última actualización 2019

Cofinanciado por:



Síguenos en las redes sociales:

- [barcelonactiva.cat/cibernarium](http://barcelonactiva.cat/cibernarium)
- [barcelonactiva](https://www.facebook.com/barcelonactiva)
- [barcelonactiva](https://twitter.com/barcelonactiva)
- [company/barcelona-activa](https://www.linkedin.com/company/barcelona-activa)