

# Introducció a la ciència de dades



Ajuntament de  
Barcelona



Barcelona  
Activa

# Índex

<b>1 INTRODUCCIÓ A LA CIÈNCIA DE DADES I A L'ENTORN DE TREBALL DE JUPYTER NOTEBOOK</b>	<b>3</b>
1.1 QUÈ ÉS LA CIÈNCIA DE DADES I QUINES APLICACIONS TÉ?	3
1.2 CONCEPTES BÀSICS DE LA CIÈNCIA DE DADES	4
1.3 TREBALLAR AMB JUPYTER NOTEBOOK	6
1.4 APROFUNDIR EN JUPYTER NOTEBOOK	9
1.5 IDEES CLAU: INTRODUCCIÓ A LA CIÈNCIA DE DADES I A L'ENTORN DE TREBALL DE JUPYTER NOTEBOOK	10
<b>2 EXPLORACIÓ INICIAL A LES DADES</b>	<b>11</b>
2.1 EXPLORACIÓ BÀSICA DE LES DADES	11
2.2 EXEMPLE D'EXPLICACIÓ DE DADES	13
2.3 MODELS PREDICTIUS	15
2.4 MODELS DE REGRESSIÓ	17
2.5 EXEMPLE D'UN MODEL DE REGRESSIÓ	19
2.6 MODELS DE CLASSIFICACIÓ	21
2.7 EXEMPLE D'EXPLICACIÓ DE DADES	23
2.8 IDEES CLAU: EXPLORACIÓ INICIAL A LES DADES	26
<b>3 VISUALITZACIÓ DE DADES AMB MATPLOTLIB</b>	<b>26</b>
3.1 GRÀFIQUES LINEALS I DE BARRES	26
3.2 SCATTERPLOTS	32
3.3 EXEMPLE PER GENERAR ELS GRÀFICS APRESOS	38
3.4 IDEES CLAU: VISUALITZACIÓ DE DADES AMB MATPLOTLIB	40
<b>4 ANÀLISI DE DADES AMB PANDA</b>	<b>40</b>
4.1 CREAR, LLEGIR I ESCRIURE UNA TAULA DE DADES	41
4.2 INDEXAR, SELECCIONAR I FILTRAR	44
4.3 ORDENAR I AGRUPAR DADES NO DISPONIBLES	49
4.4 REPÀS DEL TEMARI AMB EXERCICI PRÀCTIC	55
4.5 IDEES CLAU: ANÀLISIS DE DADES AMB PANDA	58

# 1 INTRODUCCIÓ A LA CIÈNCIA DE DADES I A L'ENTORN DE TREBALL DE JUPYTER NOTEBOOK

En aquest primer mòdul ens aproparem a què és la ciència de dades, els seus fonaments, les eines de treball més habituals, quines aplicacions té i quins camps professionals i de recerca la fan servir. També abordarem quin paper hi juga el llenguatge de programació Python.

Posteriorment, veurem què és i com funciona l'entorn de programació Jupyter Notebook i per treballar, primer, amb les funcions més senzilles; posteriorment, farem un pas endavant amb altres recursos més avançats. A partir d'aquí, ja podrem desenvolupar i escriure codi amb Python i disposar d'una orientació clara a la ciència de dades.

És important parar molta atenció durant aquest primer mòdul, ja que assolirem les habilitats bàsiques per fer front a tot el curs i seguir aprenent. Tot el contingut és important, des de l'inici fins al final. La ciència de dades ens espera!

## 1.1 QUÈ ÉS LA CIÈNCIA DE DADES I QUINES APLICACIONS TÉ?

Hola!

Potser et preguntaràs què és això de la ciència de dades i per què aquest concepte és, pràcticament, a tot arreu. En aquest vídeo, procurarem respondre a aquesta gran pregunta. Som-hi!

La ciència de dades implica l'estudi de les dades en les seves diferents formes per tal d'extreure'n coneixement o informació. Aquesta ciència fa servir les matemàtiques, l'estadística i diverses disciplines informàtiques amb l'objectiu de trobar patrons a la generació de dades que tant els humans com la natura generen contínuament. Aquests patrons són la clau per tal que les dades tinguin un valor aplicable.

La ciència de dades es fa servir en el món empresarial, en la gestió dels recursos públics, en la indústria i també en la recerca. A mesura que creix la generació de dades, sobretot per l'augment de l'activitat humana i la major capacitat de mesura i detecció d'elements susceptibles de convertir-se en dades, també creix la necessitat de personal científic i tècnic en aquesta ciència. El personal especialitzat en aquest camp ha de tenir habilitats analítiques d'aprenentatge automàtic i en mineria de dades, així com dominar llenguatges de programació per dissenyar algoritmes. A més, el científic o científica de dades haurà de saber interpretar els resultats i transmetre'ls al personal no tècnic, per tal d'activar els canvis o les mesures que el valor de les dades indiqui. Sí, la ciència de dades desperta molta demanda. Si t'agrada i t'hi esforces, pot ser una gran oportunitat des del punt de vista professional. La ciència de dades creixerà i tu pots fer-ho amb ella.

Descobrim ara, per exemple, com pot ajudar la ciència de dades a una empresa. El més important en una organització és saber anticipar-se als canvis i prendre decisions encertades. Aquesta presa de decisions pot ser guiada per la intuïció o pel coneixement

generat gràcies a la gestió de les dades. Si sabem quins patrons de compra segueix la clientela del nostre sector, podrem, per exemple, dissenyar nous productes que encaixin amb les seves necessitats. També podrem minimitzar els volums de mercaderies emmagatzemades en estoc, si sabem quan es realitzen aquestes compres i, d'aquesta manera, gestionarem millor la nostra tresoreria. La gestió de la informació en forma de dades reduirà el risc de cometre errors en la presa de decisions.

I com podem fer que la gestió d'aquesta informació, que és cada vegada més gran, pugui generar valor i coneixement aplicable d'una manera més ràpida? Aquí entra en joc l'aprenentatge automàtic, del qual segurament n'hauràs sentit a parlar. L'aprenentatge automàtic és la capacitat d'una màquina de millorar els seus pronòstics, localitzant patrons amb un major grau de certesa, a mesura que va gestionant més dades. Aquesta capacitat de les màquines de ser cada vegada més acurades en els seus resultats té una gran aplicació en el reconeixement d'imatges i veu, en la conducció autònoma, etc.

Dins d'aquest context, la intel·ligència artificial és l'habilitat d'una màquina de poder evolucionar i millorar en una sèrie de funcions i tasques sense que hagués estat específicament programada per executar-les de la mateixa manera de com ho arriba a fer. L'origen d'aquest nou camp té les arrels en la ciència de dades, però va molt més enllà i no està a l'abast d'aquest curs, tot i que és important que siguem conscients de la seva existència i quin paper pot jugar en un futur proper.

Seguim!

## 1.2 CONCEPTES BÀSICS DE LA CIÈNCIA DE DADES

Qualsevol estudi o ciència té un procés a seguir i la gestió de les dades no és una excepció. A continuació es presenten els passos que cal seguir per tal de realitzar una bona gestió de les dades.

- **Objectiu.** El primer de tot és establir un objectiu. És a dir, per què volem aquestes dades i quina finalitat tindrà el coneixement adquirit?
- **Obtenció de les dades.** Necessitem accedir i recopilar les dades amb les que volem treballar. Aquestes dades segurament arribaran de manera desordenada.
- **Preparar les dades.** Aquest pas es basa en ordenar les dades i depurar les interferències abans de començar a dissenyar models per trobar patrons.
- **Exploració de les dades.** Fase centrada en trobar patrons, correlacions i desviacions que puguin convertir les dades en un coneixement valuós i aplicable.
- **Construcció de models.** Aquesta etapa es basa en generar prediccions en funció del model escollit. En aquest cas, podrem aplicar el coneixement automàtic per anar millorant el model amb l'acumulació de més dades.
- **Presentació de resultats i ànalisis.** Aquest punt consistirà en fer públics els resultats a altres persones, per tal de dur a terme les noves accions o els canvis que els models predictius ens indiquin.

Cal dir que la majoria de vegades aquest procés no és lineal, sinó cíclic. La generació de dades és continua i cal revisar tots els passos contínuament per tal de millorar l'eficàcia i l'eficiència de les prediccions (que comportarà prendre decisions millors).

La ciència de dades ofereix, bàsicament, dos tipus de models predictius: el de regressió i el de classificació. Vegem-los:

El primer (model de regressió) té per objectiu la predicció d'un número, que depèn de la relació que pugui existir respecte a una o més variables. Aquest model predictiu es basa en els coeficients de correlació entre les variables independents i la dependent, de la qual volem saber-ne el valor. Les dades que pot necessitar aquest model compleixen els següents requisits:

- Les variables s'han de poder mesurar de manera contínua. El temps, les vendes o el pes corresponen a aquest tipus de mesures.
- Les observacions entre diferents variables no han de tenir interferències entre elles.
- Les dades no haurien de tenir valors atípics. Un valor atípic és aquell que queda molt allunyat de la resta i podria ser degut, probablement, a un error de mesura. En certs casos podria no haver cap error i que el valor de la dada fos només una excepcionalitat.
- La variància no ha de canviar al llarg de la línia predictiva. Es pot veure la definició de variància a: <https://ca.wikipedia.org/wiki/Variància>.
- Els errors al llarg de la línia de predicció segueixen una distribució normal. En l'enllaç següent es pot aprofundir en el concepte de distribució normal:  
[https://ca.wikipedia.org/wiki/Distribució\\_normal](https://ca.wikipedia.org/wiki/Distribució_normal).

El model de classificació té per objectiu predir si una opció es durà a terme o no. Aquesta opció pot tenir només dues possibilitats diferents, el que s'anomena *classificació binària*, o més de dues, que, en aquest cas, s'anomena *classificació multinomial*. Un model binari, per exemple, per tenir les opcions de *veritat o fals*, *sí o no*, etc. El model multinomial, en canvi, en té més: per exemple podríem incloure les opcions *Grup A, B o C*, o *Graus de satisfacció des de Molt satisfet, Poc Satisfet, Gens satisfet*, etc.

Gràcies al model de classificació, podrem saber, per exemple, si un accident es produirà o no (dues possibilitats) o si un producte determinat serà comprat per persones joves, adultes o d'edat avançada (més de dues possibilitats).

Per tal de generar un model capaç de fer aquesta predicció, necessitarem trobar correlacions amb altres variables independents. Per això, haurem de tenir en compte aquests elements:

- Les variables independents han de ser vàlides.
- Hem d'evitar les dades contínues, com ara la temperatura, el temps, etc.
- Hem de procurar no fer servir variables que estiguin estretament relacionades entre elles.

Per tal d'entendre els propers capítols, és important que ens familiaritzem amb els termes següents:

- **Base de dades:** espai digital destinat a emmagatzemar informació al qual es pot accedir, sempre i quan es tinguin les credencials i els permisos necessaris per tal de llegir i importar les dades que s'hi troben.
- **Overfitting:** és l'efecte de proveir massa informació sobre un model del qual ja es coneix el resultat desitjat. Aquest efecte fa que el model quedí massa definit respecte a una entrada de dades i que no sigui capaç de generalitzar resultats en altres situacions.
- **Correlació:** mesura com estan relacionats directament els canvis d'una variable respecte a una altra.
- **Mediana:** en una col·lecció ordenada de dades és el valor que es troba just a la posició central.
- **Normalització:** consisteix en ajustar els valors mesurats en escales diferents sobre una mateixa escala.
- **Valor atípic:** és aquell que està lluny respecte a la resta del grup. El valor atípic es deu a causes excepcionals o, a vegades, senzillament a un error.
- **Mineria de dades:** procés que consisteix a extreure dades d'una font determinada per ser posteriorment examinades. Inclou des de la neteja, organització i depuració de dades fins a la cerca de patrons i relacions significatives.
- **Clustering:** consisteix a recopilar i agrupar un conjunt de punts suficientment similars o propers.
- **Web scraping:** procés d'extracció de dades des de pàgines web.

## 1.3 TREBALLAR AMB JUPYTER NOTEBOOK

Jupyter és un projecte sense ànim de lucre, gratuït i de font oberta, destinat a donar suport interactiu al desenvolupament de la ciència de dades i a la computació, fent ús de qualsevol llenguatge de programació. El nostre entorn de programació serà l'aplicació anomenada Jupyter Notebook amb la qual treballarem fent ús de Python com a llenguatge de programació.

La instal·lació de Jupyter és força senzilla. Cal entrar a l'enllaç <https://jupyter.org/install> i seguir els passos que descriurem a continuació.

Tot i que es pot treballar amb qualsevol llenguatge de programació, és necessari tenir instal·lat Python a l'ordinador. Hi ha dues maneres d'instal·lar Jupyter. La primera és a través de la plataforma Anaconda i la segona fent ús del mètode Pip. Nosaltres ho farem amb la segona opció:

Obre directament la consola (“Terminal” per als sistemes Linux/Mac i “Command Prompt” per a Windows) i introduceix-hi les sentències següents:

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

Per tal de fer funcionar la nova aplicació instal·lada, només cal que escriguis:

*jupyter notebook*

Una vegada s'ha iniciat l'aplicació, a la pantalla de la terminal apareixerà la informació següent:

*\$jupyter notebook*

*[I 08:58:24.417 NotebookApp] Serving notebooks from local directory: /Users/catherine*

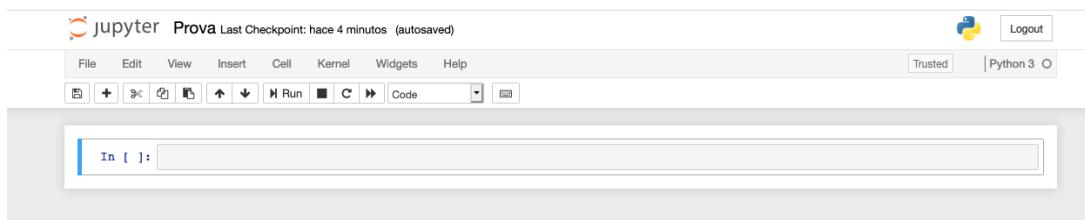
*[I 08:58:24.417 NotebookApp] 0 active kernels*

*[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).*

Això vol dir que el servidor de Jupyter s'ha carregat correctament i s'ha destinat una URL per defecte, que és la que obre el navegador. La pàgina d'inici mostrarà l'escriptori de l'aplicació des d'on es podrà iniciar un nou projecte.

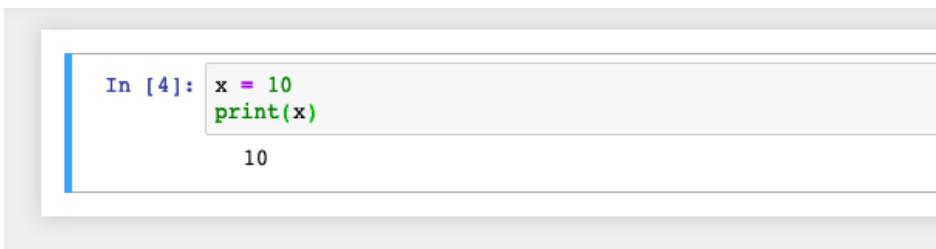
Abans de començar un projecte nou, crearem una nova carpeta per guardar-hi la feina del curs: ves a dalt a la dreta, on diu “New”, i selecciona “Folder”. Ara, iniciarem un projecte nou dins aquesta carpeta que acabem de crear. Cal anar al botó desplegable “New” i fer clic a Python3. Veuràs com s'obre una nova finestra al navegador on apareix el projecte nou.



Per tal de canviar-li el nom, clica sobre del nom que apareix per defecte (just a la dreta del logotip Jupyter) i, tot seguit, s'obrirà una finestra emergent per escriure el nom que desitgis.

Veuràs que la pantalla del projecte té tres seccions bàsiques: la primera fila (on es troben les opcions “File”, “Edit”, “View”, etc.) és el menú amb les opcions per gestionar el projecte. La segona fila (on veiem les icones de guardar, copiar, etc.) és la barra d'eines, que ens permet accedir a les operacions més bàsiques i més habituals. La tercera fila és la que s'anomena *cel·la*. Aquí podrem escriure-hi codi o text, segons el tipus de cel·la que escollim.

Per defecte, el projecte s'inicia amb una cel·la de tipus codi que ens permetrà escriure el nostre codi i també executar-lo.



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

10

El codi que hem escrit és:

$x = 10$   
 $\text{print}(x)$

Una vegada fem clic a “Run”, opció que es troba a la barra d’eines, s’imprimeix a sota de la cel·la el resultat que torna el codi.

Nota: si, en executar el codi, veus que no s’imprimeix res, fes clic a “Kernel” (opció localitzada a la barra del menú) i, tot seguit, fes clic a l’opció “Restart”.

L’altre tipus de cel·la més habitual és la de text. Per canviar el tipus de cel·la des del menú, seleccionem “Cell”, després “Cell type” i, finalment, “Markdown”. Ara, la cel·la s’ha convertit en un editor de text i no pas de codi. Es poden crear més cel·les clicant a “Insert” de la barra del menú. La nova cel·la es pot col·locar a dalt o a sota de l’actual seleccionant l’opció *above* (a dalt) o *below* (a sota). Una vegada has escrit un text, pots executar la cel·la i veuràs com se n’imprimeix el contingut.



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

10

Hem imprès el valor que conté la variable x

En cas que Jupyter no funcioni o no es carregui al navegador, pots fer una de les accions que es detallen a continuació per tal d’intentar solucionar el problema.

- Prova amb un altre navegador (per exemple, si normalment fas servir Firefox, utilitza Chrome).
- Intenta desactivar qualsevol extensió del navegador o qualsevol extensió de Jupyter que hagis instal·lat.
- Alguns programes de seguretat d’internet poden interferir amb Jupyter. Si tens un programa de seguretat, prova de desactivar-lo temporalment i mira la configuració per obtenir una solució.
- A la barra de navegació (URL), canvia el localhost per 127.0.0.1.

Per a més informació sobre la instal·lació de Jupyter i el seu funcionament, recorda que pots visitar la pàgina oficial de la plataforma: <https://jupyter-notebook.readthedocs.io/en/latest/index.html>.

## 1.4 APROFUNDIR EN JUPYTER NOTEBOOK

Hola a tothom!

En aquest vídeo, aprofundirem en les accions i eines de l'aplicació Jupyter.

Una vegada tinguem el projecte creat (recorda que un projecte és un *notebook*), podem afegir-hi, primer, el títol. La primera cel·la la canviarem a mode d'encapçalament d'aquesta manera.

Obrim el desplegable de la barra d'eines i cliquem “Heading”. Aquí apareix un missatge dient que, només afegint un coixinet en una cel·la normal de text, és a dir de tipus Markdown, ja en tenim prou. Doncs, ho farem així. Podem posar-hi el títol que vulguem.

Escrivim aquest codi per començar a fer coses noves:

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

Aquest codi ens crea i ens imprimeix per pantalla una nova llista que només inclou aquells valors superiors a 15.0 de la primera llista.

Cliquem a “Run” i veiem el resultat.

[21.6, 15.7]

Fixa't que, quan fem clic a “Run”, per defecte, es genera una cel·la nova, sempre de tipus “codi”.

Si durant el desenvolupament del projecte, veiem que és necessari inserir una cel·la abans d'una altra que ja té contingut, ho podem fer a través del menú, clicant “Insert” i “Insert Above”. En aquesta nova cel·la, hi podem indicar l'acció del codi que trobem tot seguit:

*Escrivim codi per afegir una llista de valors superiors a 15.0*

Important! Recorda que, quan vulguis introduir text, hauràs de canviar el tipus de cel·la de codi a Markdown.

Si ara volem eliminar una cel·la, la seleccionem i cliquem sobre la icona de les tisores que es troba a la barra d'eines. Després, s'hi poden afegir més cel·les i seguir treballant només fent clic al botó **+**, situat a la mateixa barra.

Un truc que ajudarà a localitzar codi més ràpidament dins d'una cel·la és numerar les línies de codi. Aquesta opció es troba a la barra del menú, a la secció “View”. Si fem clic a “Toggle Line Numbers” veurem com apareix aquesta numeració.

Si, a mesura que anem desenvolupant el projecte, volem previsualitzar com quedaría la feina impresa, només cal anar a “File” i seleccionar “Print Preview”. Recorda que molta de la feina que es fa en ciència de dades s'ha de mostrar a altres persones per tal de trobar suport i aplicar els nous coneixements en la nostra empresa, equip de recerca, etc. Això vol dir que la presentació del *notebook* és molt important.

També podem convertir el nostre *notebook* en un fitxer HTML, per publicar-lo en un web. Per fer-ho, en aquest cas, només cal seleccionar “Download as” i tot seguit “HTML”.

Si vols fer un repàs a la navegació bàsica de l'aplicació, pots fer clic a la secció “Help” i seleccionar “User Interface Tour”, que et guiarà a través de les accions de Jupyter Notebook.

Ep! Abans d'acabar, pensa a guardar la feina sovint. Encara que l'aplicació ho faci de manera automàtica, és important tenir present que ho pots fer de manera manual, només fent clic a la icona del disc.

Ara ja estem en condicions de treballar en un projecte de ciència de dades fent ús de Jupyter Notebook.

Seguim!

## 1.5 IDEES CLAU: INTRODUCCIÓ A LA CIÈNCIA DE DADES I A L'ENTORN DE TREBALL DE JUPYTER NOTEBOOK

En aquest primer mòdul, hem après els conceptes bàsics de la ciència de dades i les seves aplicacions més rellevants. Ara sabem que la informació codificada en forma de dades es pot depurar i netejar, i podem trobar patrons per tal de generar coneixement aplicat.

Els camps on més es fa servir la ciència de dades són el món empresarial i la recerca en general. L'aprenentatge automàtic és la capacitat d'una màquina de millorar els seus pronòstics, localitzant patrons amb un major grau de certesa, a mesura que aquesta màquina va gestionant més dades. Hi ha dos tipus de models:

- Model de regressió. Serveix per calcular un valor determinat, és a dir, un número.
- Model de classificació. Serveix per respondre a un qüestió en només dues opcions (binomi) o amb més de dues opcions (multinomial).

Jupyter Notebook és una plataforma gratuïta i de font oberta molt popular per desenvolupar projectes de ciència de dades. Aquesta aplicació té tres seccions: la barra de menú, la barra d'eines i la secció de les cel·les. Pel que fa a les cel·les, podem diferenciar-ne, principalment,

dos tipus: un correspon al codi i l'altre al text. Quan necessitem picar codi, escollirem el primer i, quan necessitem escriure text, farem servir el segon. Això ens permet escriure codi i afegir explicacions escrites i rellevants sobre aquest, i presentar-ho tot en el mateix document.

Jupyter Notebook permet afegir i eliminar cel·les fàcilment. Les cel·les noves es poden col·locar allà on més ens convingui. L'aplicació també ens permet visualitzar la nostra feina mitjançant una previsualització i, fins i tot, exportar el fitxer a HTML, per tal que es pugui penjar en un web.

## 2 EXPLORACIÓ INICIAL A LES DADES

Després d'una introducció al món de la ciència de dades, és hora de començar a treballar-hi. El primer que farem serà estudiar les dades. Normalment, les dades ens arriben en forma de taula, amb files i columnes. Veurem com importar o carregar una taula i com extreure ràpidament la informació general que conté. Per fer-ho, necessitarem aplicar unes funcions senzilles amb Python. Tota aquesta feina serà executada en el nostre nou entorn de programació Jupyter Notebook. Aquest primer pas és molt important i sempre l'haurem d'executar en qualsevol dels nostres projectes, independentment de la complexitat de les dades que fem servir en cada moment.

També aprofundirem en els dos tipus de models predictius bàsics: el de regressió i el de classificació. Començarem, primer, des d'un punt de vista teòric o descriptiu i, després, agafarem experiència amb exercicis pràctics. Tot ho farem pas a pas.

### 2.1 EXPLORACIÓ BÀSICA DE LES DADES

La ciència de dades ens permet descodificar la informació per tal d'adquirir més coneixements d'un sistema o del nostre entorn. Normalment, aquestes dades venen en forma de taula, conformada per dues entitats: files i columnes.

Les files, generalment, contenen els registres o observacions, i les columnes contenen les variables o atributs. Els registres, doncs, són els casos d'estudi que incorporen una informació determinada en cada atribut. Vegem-ne un exemple senzill:

	EDAT	PES	ALÇADA	CIUTAT	SEXЕ
JOAN	35	80	1,75	BARCELONA	HOME
ALBA	32	65	1,68	TARRAGONA	DONA
MIQUEL	47	74	1,78	GIRONA	HOME
MERITXELL	44	59	1,60	LLEIDA	DONA

- Les variables o atributs (columnes) són: edat, pes, alçada, ciutat i sexe.
- Les observacions o registres (files) són: Joan, Alba, Miquel i Meritxell.

Nota: d'ara en endavant sempre ens referirem a les files com observacions i a les columnes com a atributs.

És important saber que les taules poden venir amb dos tipus de dades: les **operacionals** i les **organitzatives**. Les primeres contenen informació directa d'observacions unitàries i puntuals com, per exemple, l'import i el producte de cada compra en un establiment, totes i cadascuna de les compres de bitllets d'avió d'una determinada companyia àrea, etc. Les segones, en canvi, recullen dades agrupades o tendències. Un exemple de dada organitzativa serien els casos d'una malaltia agrupats en diferents ciutats: aquí no estudiem cas per cas una observació individual, sinó una agrupació basada en certs criteris. Aquest segon grup es fa servir moltes vegades quan poden aparèixer conflictes relacionats amb la protecció de dades.

Recorda que el primer que necessitem establir abans de començar a treballar amb les dades són els **objectius**. Hem de saber respondre a la pregunta: quina informació rellevant volem obtenir amb aquestes dades? Una vegada ho tinguem clar, ens haurem de posar a treballar per aconseguir la resposta. Després del plantejament dels objectius, els següents passos són l'obtenció de la informació (que tindrem resolta, quan disposem d'una taula de dades) i, finalment, la preparació de les dades. Per fer aquest últim pas, hem de veure si tenim, o no, cel·les amb valors anormals o sense informació (*missing value*).

Nota: és important que ens familiaritzem amb expressions en anglès com *missing values*, ja que la indústria té com a referència aquest idioma i, tot i que treballis a Catalunya, de ben segur que les hauràs de fer servir.

Anem a veure, primer, un exemple de dades anormals:

	EDAT	PES	ALÇADA	CIUTAT	SEXЕ
JOAN	35	80	1,75	BARCELONA	HOME
ALBA	32	65	1,68	TARRAGONI	DONA
MIQUEL	47	74	1,78	TARRAGONA	HOME
MERITXELL	44	59	1,06	LLEIDA	DONA

En una exploració ràpida, podríem sospitar que l'alçada de la Meritxell pot tenir un valor anormal (probablement, degut a un error a l'hora de posar la dada). També veiem que el valor de ciutat de l'Alba és Tarragoni i, per tant, no correspon amb el valor real. Aquests errors, tard o d'hora, poden aparèixer i, si no els detectem, la informació que extraiem pot ser de baixa qualitat. A vegades, és pràcticament impossible detectar-los. Per això, és important treballar amb una base de dades àmplia, que tingui prou observacions per tal que aquests errors no ens alterin excessivament els nostres models predictius.

En els casos de cel·les sense valors, les causes poden ser molt diverses. Si, per exemple, registrem amb un termòmetre la temperatura cada cert període de temps i veiem cel·les sense valors, serà probablement degut a un mal funcionament de l'instrument de mesura. A continuació, mostrem un exemple d'un sistema format per cinc termòmetres, el qual ens

serveix per mesurar la temperatura ambient cada 30 minuts en diverses zones d'una determinada ciutat.

L'objectiu d'aquesta taula podria ser, per exemple, veure si hi ha certes parts de la ciutat que són més caloroses que d'altres. L'estudi ens podria aportar coneixement important si podem relacionar les variacions de temperatura amb l'arquitectura del municipi o amb el trànsit. Això ens permetria proposar canvis urbanístics, si es considerés oportú.

	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: les columnes són els diferents termòmetres situats a diferents punts de la ciutat.

Les cel·les de color gris no contenen valors dins de la franja horària de 9:30 a 10:00. Per alguna raó desconeguda, el termòmetre no ha registrat la temperatura. En un cas real, hauríem d'analitzar si aquesta manca d'informació podria alterar significativament el nostre coneixement.

## 2.2 EXEMPLE D'EXPLICACIÓ DE DADES

Hola!

Ara carregarem una taula de dades i n'extremem la informació bàsica. En primer lloc crearem un projecte nou. Després, carregarem el mòdul de Python anomenat *Pandas*. Aquest mòdul ens permetrà treballar amb bases de dades, en anglès *Data Frames*. Així que introduïm aquest codi a la primera cel·la:

```
import pandas as pd
```

Hem anomenat el nostre mòdul *Pandas* com *as pd*. A partir d'ara, cada vegada que ens hi referim o el cridem amb el nostre codi, farem servir *pd*.

El següent pas és crear una variable per emmagatzemar-la a la nostra base de dades. Abans, però, tinguem en compte que hem de tenir guardada la nostra base de dades en format CSV. Podem exportar un document amb format CSV des del programa Excel.

Et pots descarregar el fitxer (la base de dades) en aquest enllaç ([...../iris.csv](#)). Guarda'l a la mateixa carpeta on tenim el projecte.

Llavors, introduïm aquest codi:

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

Aquesta base de dades recull les longituds i amplades del sèpal i del pètal d'una sèrie de flors.

De fet, aquesta variable és un objecte que es crea tot assolint les funcions del paquet Pandas. Més endavant, haurem de fer ús d'aquestes funcions. Recorda que a Python tot són objectes.

Fixa't que estem fent servir funcions. `read_csv()` és una funció del mòdul `pd` (Pandas) que ens permet carregar fitxers del tipus CSV.

Executem la variable `data_frame`, per tal de visualitzar la base de dades.

També podem veure només un determinat nombre d'observacions. Si apliquem la funció `head()` a la base de dates, escrivint el codi següent:

```
data_frame.head()
```

Ara només apareixen els cinc primers registres; però, si posem el nombre de registres que volem visualitzar com a argument de la funció, la taula mostrerà els que li diguem. Per exemple, si volem veure els deu primers, escriurem:

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

Fins i tot podrem visualitzar només una sèrie d'observacions i només dels atributs que vulguem. Per exemple, si escrivim aquest codi:

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

Aquí veurem les dades de les observacions 0, 1, 6..., és a dir, de la primera, segona i setena..., respecte a la longitud de l'atribut sèpals.

Ara, si apliquem la funció `describe()` a tota la base de dades amb aquest codi:

```
data_frame.describe()
```

veiem una taula que recull una sèrie de dades per a cada columna. Anem a analitzar què volen dir cadascun d'aquests resultats.

- Count: és el nombre d'observacions per a cada atribut. I, clar, totes tenen el mateix nombre d'observacions.
- Mean: ens indica la mitjana, considerant tots els valors de la columna.
- std: ens mostra la desviació dels resultats.
- min: el valor més petit de l'atribut.
- 25%: exposa el valor que és més gran que el 25 % de la sèrie.
- 50%: el valor que és més gran que el 50 % de la sèrie.
- 75%: el valor que és més gran que el 75 % de la sèrie.
- Max: el valor més gran de la columna.

Ara ja coneixem les funcions bàsiques que ens permeten carregar i explorar una base de dades. Aquesta primera exploració és l'inici que ens permetrà veure, després, quin model predictiu necessitarem fer servir per aconseguir el nostre objectiu. Entendre les dades és la primera tasca i un pas que mai no ens hem de saltar.

Seguim!

## 2.3 MODELS PREDICTIUS

Un model predictiu és un mecanisme que ens permet calcular, amb un grau de certesa determinat, un resultat futur a través de la relació d'unes dades prèviament donades. Cal dir que els models són molt diversos i l'ús d'uns o d'altres variarà en funció del tipus d'informació que volem aconseguir (per exemple, dependrà de si volem obtenir un valor determinat o, en canvi, si volem conèixer una situació entre diverses de possibles). L'elecció d'un model també variarà segons el tipus de dades i les relacions que s'estableixin entre elles. El més important és saber que cap model és perfecte.

Per determinar el grau de certesa d'un model, l'haurem d'entrenar. Això vol dir que part de les observacions de la nostra base de dades les utilitzarem per preveure si el model genera resultats correctes o no. Una vegada hem entrenat diversos models i hem vist quin funciona millor, podrem escollir-ne un i desenvolupar el treball.

Per escollir un bon model, primer hem d'entendre les dades: de quina font venen, a què es refereixen, quina relació tenen amb el nostre objectiu, etc. La qualitat de les dades és el primer punt a tenir en compte. Si, per arribar a un objectiu, escollim les dades incorrectes, la informació que gestionem ens portarà a conclusions errònies que provocaran que prenguem decisions incorrectes.

Els models es poden classificar en dos grans grups: els de regressió i els de classificació, tot i que n'hi ha molts que poden servir pels dos casos. Les prediccions de regressió procuren predir un valor numèric discret o determinat, mentre que els de classificació busquen predir un estat d'entre diversos escenaris possibles. A continuació, mostrem una taula amb diversos models i a quin grup pertanyen:

Model	Grup
<i>Random forest</i>	Classificació
<i>Lineal regression</i>	Regressió
<i>KNN</i>	Regressió i classificació
<i>Boosted Tree</i>	Regressió i classificació
<i>Gradient Boosted Machines</i>	Regressió i classificació

Per tal que les dades generin patrons entre elles, han d'estar relacionades. Dit d'una altra manera, les dades que conté una determinada variable (atribut) estan relacionades, amb més o menys afinitat, amb la dada que conté una altra variable.

La correlació és l'eina estadística que mesura com n'és d'intensa una relació entre un atribut de valor numèric i un altre. Els valors de correlació entre dues variables numèriques poden assumir valors des de 0 a 1 o a -1. Si la correlació és positiva, això voldrà dir que, si el valor d'una dada augmenta, també ho farà l'altre. També serà positiva si el valor d'una baixa i el de l'altra també. En canvi, si la correlació és negativa, significa que, si el valor d'una augmenta, el de l'altra baixarà, i al revés.

L'ús de la prova d'independència Chi-Square permet al personal investigador avaluar si la relació observada entre les variables nominals (no numèriques) en una mostra particular també es pot trobar a la població. Tot i així, aquesta pot no ser adequada, si la mida de la mostra no és prou extensa.

A continuació, analitzem alguns exemples d'indústries que fan servir models predictius per millorar la seva competitivitat:

- La indústria aeroespacial necessita models predictius que prediquin la durada dels components per augmentar així el rendiments dels seus avions i reduir els costos de manteniment.
- Les assegurances fan servir, com a eina principal, els models predictius, per tal de dissenyar productes de risc per a diferents tipus de clientela.
- Els bancs calculen mitjançant models predictius, l'endeutament límit que pot assolir una persona que encaixi dins d'un determinat grup de la població.
- La medicina fa servir models predictius per relacionar el so respiratori amb el risc de patir un atac d'asma.
- Un departament de màrqueting sempre necessita saber si els nous productes que està dissenyant l'àrea d'I+D seran acceptats pel mercat. El risc de dissenyar un producte nou es pot reduir fent un estudi previ sobre el comportament de la clientela i de productes relacionats.

Els passos per generar i optimitzar un model predictiu són els següents:

- Una vegada hem estudiat el tipus de dades amb les que estem treballant i quin objectiu volem assolir, escollim un o més models, basant-nos en la nostra experiència.
- Lectura dels paràmetres que mostren la certesa de la nostra predicció: optimització del model mitjançant la configuració de certs paràmetres i comparació amb la resta dels models escollits.
- Entendre els resultats. És molt important saber dominar els resultats amb una mirada clara a la situació en la que ens trobem. Les dades donen valor quantitatius que necessita una valoració qualitativa, per tal que es converteixin en coneixement aplicable.
- Validació del model i integració en aplicacions, pàgines web, dispositius, etc.

Ara ja tens els coneixements necessaris per començar a aplicar models predictius.

## 2.4 MODELS DE REGRESSIÓ

Actualment vivim envoltats d'una gran quantitat de dades, computadores potents i intel·ligència artificial. Això només és el començament. La ciència de dades i l'aprenentatge automàtic impulsen el reconeixement d'imatges, el desenvolupament de vehicles autònoms, decisions en els sectors financer i energètic, avenços en medicina, augment de les xarxes socials i molt més. Els models de regressió juguen un paper molt important en aquest camp. Predir el comportament d'un sistema ens permetrà prendre decisions més encertades o, fins i tot, dissenyar nous productes que tinguin èxit en els mercats.

Un model predictiu de regressió es fa servir, principalment, per predir la resposta o resultat d'una variable, segons els canvis que assoleixen unes variables que estan relacionades amb aquesta. Posem pel cas que volem estimar el creixement de les vendes d'una empresa en funció de les condicions econòmiques actuals. Tenim les dades recents de l'empresa que indiquen que el creixement de les vendes és al voltant de dues vegades i mitja el creixement de l'economia. Amb aquesta perspectiva, podem predir les vendes futures de l'empresa, a partir d'informació actual i anterior.

La regressió lineal és la tècnica de regressió més senzilla i la que ofereix la interpretació de resultats més senzilla. Anem a descobrir els seus fonaments matemàtics.

Quan implementem una regressió lineal d'alguna variable dependent  $y$  del conjunt de variables independents  $\mathbf{x} = (x_1, \dots, x_r)$ , on  $r$  és el nombre de predictors, assumim una relació lineal entre  $y$  i  $\mathbf{x}$ :  $y = \beta_0 + \beta_1 x_1 + \dots + \beta_r x_r + \varepsilon$ . Aquesta equació és la de regressió.  $\beta_0, \beta_1, \dots, \beta_r$  són els coeficients de regressió i  $\varepsilon$  és l'error aleatori.

$\mathbf{Y}$  es correspondria a l'atribut del qual volem preveure el resultat. Els valors d' $\mathbf{x}$  corresponen amb els atributs dels quals se suposa una certa relació amb  $y$ .

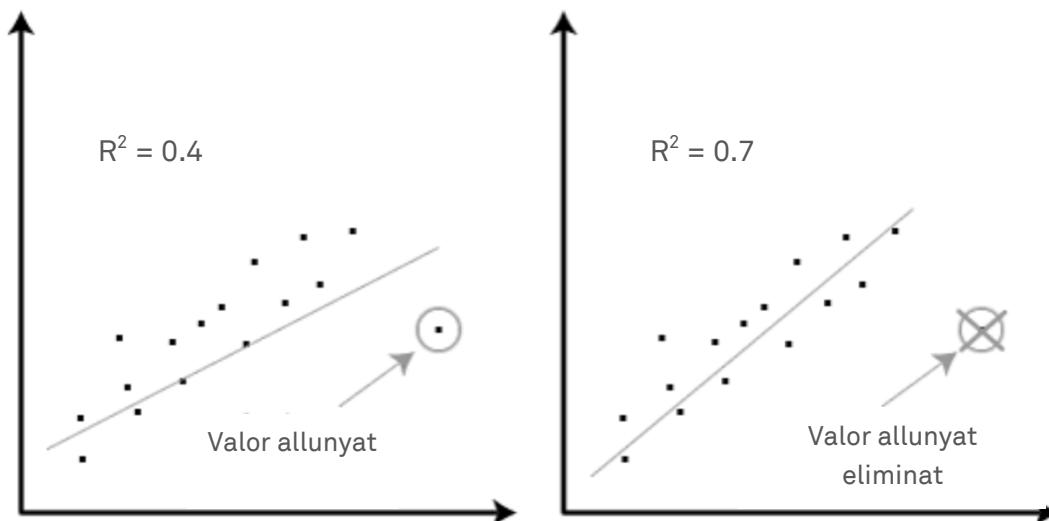
La variació de les respostes reals  $y_i$ ,  $i = 1, \dots, n$ , es produeix, en part, a causa de la dependència de l'atribut  $x_i$ . Tot i això, també hi ha una diferència inherent a la sortida. El coeficient de determinació, conegut com a  $R^2$ , indica quina quantitat de variació en  $y$  es pot

explicar per a la dependència d' $x$  mitjançant el model de regressió. El major  $R^2$  indica un millor ajustament i significa que el model pot explicar millor la variació de la sortida amb diferents entrades. El valor  $R^2 = 1$  s'adapta perfectament, ja que els valors de les respostes previstes i reals s'ajusten i coincideixen perfectament.

La regressió lineal més simple és aquella en què la variable  $y$  només depèn d'una altra variable  $x$ . En els casos pràctics, en ciència de dades, aquest cas no s'acostuma a donar. Els casos reals són més complexos i normalment tenen més d'una variable  $x$  relacionada amb un atribut  $y$ . En aquests casos reals, estarem parlant, doncs, de regressió lineal múltiple.

Si només hi ha dues variables independents, la funció de regressió estimada és  $f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2$ . L'objectiu de la regressió és determinar els valors dels coeficients  $b_0$ ,  $b_1$  i  $b_2$ , de tal manera que els resultats estiguin el més a prop possible de les respostes reals.

Quan generem els models, hem de veure si aquests poden tenir un coeficient de correlació massa baix o massa alt (molt a prop d'1). En el primer cas, estarem davant d'un model que genera uns resultats massa allunyats dels reals. Estarem davant d'una situació d'*underfitting*. En el segon cas, estarem davant d'un sistema d'*overfitting*. Això voldrà dir que el model encaixa perfectament amb les dades donades, però corre el risc de preveure molt malament quan s'aplica a altres dades amb les quals no ha estat entrenat. Com que l'error és la distància quadrada entre el punt de dades i la línia de regressió, les grans distàncies tenen errors desproporcionadament grans, i fan que l'anàlisi de regressió esdevingui una solució amb un coeficient de correlació baix. Com a tal, aquells valors que estiguin significativament allunyats de la recta, haurien de ser eliminats del conjunt de dades. Aquests valors són aquells que es podrien descobrir durant l'exploració i depuració de dades.



En les dues gràfiques que es mostren just a dalt, es veu com una vegada hem eliminat el valor allunyat, el nostre model predictiu ha millorat considerablement. Per facilitar-nos la interpretació dels resultats, en ciència de dades és molt útil comptar amb gràfiques que mostrin les dades de manera visual.

## 2.5 EXEMPLE D'UN MODEL DE REGRESIÓ

Hola!

En aquest vídeo, farem servir el nostre primer model. Concretament, crearem un model de regressió. Aquest ens servirà per veure com podem predir l'amplada dels pètals, mirant la relació que hi ha entre les amplades i les longituds dels pètals.

Bé, primer creem un projecte nou. Segur que ja ho saps fer, però, si no, és ben fàcil. Cliquem a “New” i seleccionem “Python 3”. Després, importem el mòdul *Pandas* i carreguem la base de dades amb la qual volem treballar. Farem servir l'anomenada ‘iris.csv’.

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

Una vegada tenim la base de dades carregada, necessitem importar altres mòduls que ens permetin seleccionar part de les dades, per poder entrenar el model.

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

Després, importarem concretament el model que volem fer servir a partir d'una regressió lineal.

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

Després d'importar aquestes llibreries, el que farem serà seleccionar només les columnes “Petal.Width” i “Petal.Length”, i les guardarem en una altra variable. Aquesta variable estarà guardant una altra base de dades.

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

Ara, ens toca seleccionar de manera aleatòria una sèrie d'observacions per tal d'entrenar el model. Concretament, estem agafant el 80 % dels registres i deixarem el 20 % restant per testejar el nostre model.

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

Abans de continuar, guardarem una còpia del *train\_set* en una nova variable.

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

Una vegada hem fet això, aplicarem la funció *corr()* per veure com estan relacionats els valors de les dues columnes de la part d'entrenament. Recorda que ara ja no estem treballant amb tota la base de dades inicial. Primer, hem seleccionat les columnes que volíem estudiar i, després, n'hem agafat el 80 % per entrenar el model.

```
df_copy.corr()
```

Un valor de 0,9625 és prou alt. La relació és molt estreta. Recorda que R mesura com estan de relacionades entre sí dues variables. Com més proper sigui aquest valor a 1 o -1, més estreta serà la relació. Això voldrà dir que un canvi en una variable afectarà directament a l'altra.

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

Ara seria interessant visualitzar els valors de les dues columnes en una gràfica. Per això, importem el mòdul següent:

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

Assignem, ara, quina columna correspondrà a l'eix *X* i quina a l'eix *Y*, i ja ho tenim.

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

Es veu clarament com les dues variables estan estretament lligades. Quan una creix, també ho fa l'altra, tal com ens indicava el resultat de la funció *corr()*.

Bé, ara ve el pas decisiu: aplicar el model escollit.

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

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

Amb aquest codi, el que estem fent és dir-li al model quina serà la variable independent i quina serà la variable de la qual volem predir els resultats en funció de la primera.

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

Creem una instància de l'objecte `LinearRegression()` i fem servir la funció `fit()` que hi conté, per aplicar el model. A més, aquest objecte `lin_reg` també incorpora `coef` i `intercept`.

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

Els resultats que obtenim són:

*Coefficients: [0.41323829]*

*Intercept: -0.3566680410565528*

Amb aquests valors, ja podem escriure la fórmula matemàtica que defineix la regressió lineal:

*Petal.width = 0.4132 \* Petal.Length - 0.3566*

Però, en comptes de fer-ho a mà, l'objecte `lin_reg` ens dona la funció `predict()`, en la qual, si posem el valor de la llargada del pètal com a argument, aquesta ens tornarà el valor de l'amplada que hi prediu.

Aquest és un exemple senzill, però prou didàctic per entendre el concepte. La vida real té problemes més complexos per resoldre, però aquest és un bon inici.

Seguim!

## 2.6 MODELS DE CLASSIFICACIÓ

La classificació és una funció de gestió de dades que assigna elements d'una col·lecció a categories, classes o també als anomenats segments. L'objectiu de la classificació és predir amb certesa la classe d'objectiu per a un element donat de la col·lecció. Per exemple, es podria utilitzar un model de classificació per identificar a quina categoria de risc (baix, alt o mitjà) es poden assignar les persones sol·licitants de préstecs. En aquest cas, les persones equivaldrien als elements de la col·lecció (en aquest cas, una població) i els riscos serien les categories.

Un model de classificació comença amb un conjunt de dades de les quals es coneixen les assignacions de les categories. Per exemple, es podria desenvolupar un model de classificació que preveu el risc de crèdit basat en dades observades per a moltes persones sol·licitants de préstecs ja existents, els quals tenen assignat un risc alt, baix o mitjà. A més de la qualificació de crèdit històric, les dades podrien fer un seguiment de l'historial d'ocupació, propietat o lloguer d'habitatges, anys de residència, nombre i tipus d'inversions,

etc. La qualificació creditícia seria l'objectiu, els altres atributs serien predictors i les dades de cada client o clienta constituirien una observació.

La classificació més simple és la classificació binària, en la qual l'atribut només té dos valors possibles; per exemple, qualificació creditícia alta o qualificació creditícia baixa.

En el procés de creació de models, un algoritme de classificació troba relacions entre els valors dels atributs i la categoria o segment on volem col·locar una observació. Diferents algoritmes de classificació utilitzen tècniques diverses per trobar aquestes relacions. Les relacions, doncs, es codifiquen en un determinat model, que serveix després per predir quins elements d'una col·lecció cauran en una categoria o en una altra. Els models de classificació es proven mitjançant la comparació dels valors predictius amb els valors objectiu coneguts. Les dades històriques d'un projecte de classificació es divideixen generalment en dos conjunts de dades: un per construir el model; l'altre per provar el model. Suposem que volem predir quins dels nostres clients i clientes podrien augmentar la despesa, si se'ls proporciona una targeta d'afinitat. Podem crear un model mitjançant dades demogràfiques sobre clientela que ha utilitzat anteriorment una targeta d'afinitat. Com que volem predir una resposta positiva o negativa (saber si augmentarà o no la despesa), aquest cas és un clar exemple d'un model de classificació binari.

Les dades de prova han de ser compatibles amb les dades que s'utilitzen per crear el model i s'han de preparar de la mateixa manera que es van preparar les dades de creació. Normalment, les dades de creació i les dades de prova provenen del mateix conjunt de dades històriques. Un percentatge dels registres s'utilitza per crear el model; els registres restants s'utilitzen per provar-lo.

Les mètriques de prova s'utilitzen per avaluar la precisió del model de predicció dels valors coneguts. Si el model funciona bé i compleix els requisits comercials, es pot aplicar a dades noves, per predir el futur.

A continuació, anem a descriure els paràmetres que mesuren la qualitat del model emprat:

- La precisió (CVA) fa referència al percentatge de prediccions correctes realitzades pel model, respecte a les classificacions reals de les dades de prova.
- Confusion matrix és una matriu que mostra el nombre de prediccions correctes i incorrectes realitzades pel model, respecte a les classificacions reals de les dades de prova. La matriu és n-per-n, on n és el nombre de classes o segments.
- L'elevació o lift mesura el grau en què les prediccions d'un model de classificació són millors que les prediccions generades aleatoriament. L'elevació només s'aplica a la classificació binària i requereix la designació d'una classe positiva. (Consulteu “Classes positives i negatives”). Si el model en si no té una destinació binària, podem calcular la designació, designant una classe com a positiva i combinant totes les altres classes com una classe negativa.

Aquest paràmetre s'utilitza habitualment per mesurar el rendiment dels models de resposta en aplicacions de màrqueting. L'objectiu d'un model de resposta és identificar segments de la població amb concentracions potencialment altes d'enquestats positius en una campanya

de màrqueting. L'elevació revela quina part de la població s'ha de sol·licitar per obtenir el percentatge més alt de possibles persones que responguin.

ROC és una representació gràfica de la sensibilitat davant de l'especificitat per a un sistema de classificació binari, segons varia el llindar de discriminació.

## 2.7 EXEMPLE D'EXPLICACIÓ DE DADES

Hola!

En aquest vídeo, farem servir un model de classificació. Concretament el KNN, per tal de poder predir quina categoria d'estrelles (entre 1 i 5) tindrà un comentari, dependent del seu nombre de paraules i del valor del sentiment (valors entre -4 i 4). El valor del sentiment fa referència al grau d'intensitat que s'afegeix arbitràriament per donar més pes o menys als comentaris. Tot a punt? Doncs, comencem!

Com veiem, ja tenim el projecte creat. Podem descarregar-nos la base de dades amb la qual treballarem a través de l'enllaç que trobem a sota del vídeo. Les primers línies de codi són:

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

Tot seguit, podem veure un primer resum estadístic. Recordes quina era la funció que podem fer servir? Sí, és aquesta:

```
df.describe()
```

Aquí veiem, doncs, que la categoria mínima dels comentaris és 1 i la màxima és 5. I que el nombre de paraules d'un comentari és 1 i el màxim és 103.

Apropem-nos a aquestes dades d'una manera més visual. Escrivim aquest codi:

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

Passem, ara, a preparar el model. Primer, importem aquests dos mòduls:

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

Una vegada fet aquest pas, preparam les dades de tal manera que tinguem *Star Rating* com a variable resultant i *wordcount* i *sentimentvalue* com a variables dependents.

```
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)
```

Abans d'aplicar el model, necessitem importar-lo des del mòdul on està present:

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

I ara sí que ho tenim a punt. Escrivim el codi:

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

Per tal de veure si el model és bo o no, haurem de calcular l'*Accuracy*. Ho fem de la següent manera:

```
print('Accuracy del model K-NN pel train set és: {:.2f}'
.format(knnModel.score(X_train, y_train)))
```

El resultat que obtenim és:

*Accuracy del model K-NN pel train set és: 0.90*

El que ens està indicant és que la precisió pel paquet d'entrenament o *train set* és del 90 %.

Fixa't que estem fent servir un model de classificació perquè volem predir una categoria (1, 2, 3, 4 o 5) i no un número de valor quantitatiu, com en l'exercici anterior.

Per treballar sobre la precisió del model, necessitem importar les funcions següents:

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

I, després, posar-les a treballar sobre el nostre model:

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

Com veiem, el resultat “f1-score” és del 87 % i, per tant, podem considerar que la precisió és prou bona.

Ara, ens queda veure quina categoria assigna el nostre model, considerant un nombre de paraules (*wordcount*) i el valor de sentiment (*sentimentvalue*) determinats. Introduïm aquest codi:

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

Introduïm el *wordcount* i el *sentimentvalue* a la funció *predict()*:

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

I el model ens diu que, en aquest cas, la categoria més probable seria 5.

Podem fer servir la funció *predict\_proba()*, per tal de veure quina probabilitat tindria cada categoria per a un determinat comentari, sabent el nombre de paraules i el valor de sentiment. Ho farem així:

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

El resultat que obtenim és:

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

Això vol dir que la probabilitat de cada categoria per a un comentari de 20 paraules amb un valor de sentiment de 0.0 és la següent:

Categoría 1 – 0,3 %

Categoría 2 – 2,52 %

Categoría 3 – 97,09 %

Categoría 4 – 0 %

Categoría 5 – 0 %

Ja hem vist un model de classificació per predir quina categoria tindria un comentari, tenint en compte el seu nombre de paraules i el sentiment de valor.

Seguim!

## 2.8 IDEES CLAU: EXPLORACIÓ INICIAL A LES DADES

El més important abans de començar a fer servir cap model és entendre les dades que tractem, detectar els errors més comuns, identificar les cel·les sense dades i els valors allunyats. És una feina essencial trobar-los i netejar-los. Altrament, aquests afectarien negativament la qualitat dels models. Per tal de fer una exploració inicial de les dades, hem après a fer ús de funcions bàsiques com ara són `head()` i `describe()`.

Tenim dos tipus bàsics de model predictiu: el de regressió i el de classificació. El primer recull aquells models que procuren predir valors numèrics quantitatius, mentre que els models que formen part del segon tipus es fan servir quan el resultat és un valor qualitatiu.

Tots els models necessiten que una part de la taula es faci servir per a l'entrenament. L'altra part restant es pot fer servir per mesurar si les prediccions dels models són prou bones o no. En els models de regressió lineal, el paràmetre de qualitat és el coeficient de regressió, mentre que, per a un model de classificació, és *l'accuracy*.

## 3 VISUALITZACIÓ DE DADES AMB MATPLOTLIB

Per visualitzar les dades que tractem és útil aprendre a generar gràfics. Els gràfics són l'eina més poderosa per transmetre al públic no tècnic la nostra feina d'anàlisi de les dades: mostren visualment molta informació que d'altra manera podria passar desapercebuda.

En aquest mòdul, aprendrem a construir gràfics lineals, de barra i de dispersió. Depenent del tipus de dades amb les qual estem treballant i les qüestions que vulguem resoldre, en farem servir un o un altre. L'instrument que emprarem serà Matplotlib.

## 3.1 GRÀFIQUES LINEALS I DE BARRES

Ha arribat el moment d'aprendre a fer gràfics amb les dades. La visualització gràfica és, segurament, una de les eines més potents per entendre ràpidament quina informació ens poden transmetre les dades. En aquest tutorial, aprendrem a construir gràfics lineals i de barres fent ús de la llibreria Matplotlib de Python.

Començarem iniciant un nou projecte amb Jupyter Notebook. Una vegada tinguem el projecte carregat, el primer que haurem de fer és importar la llibreria `matplotlib.pyplot`.

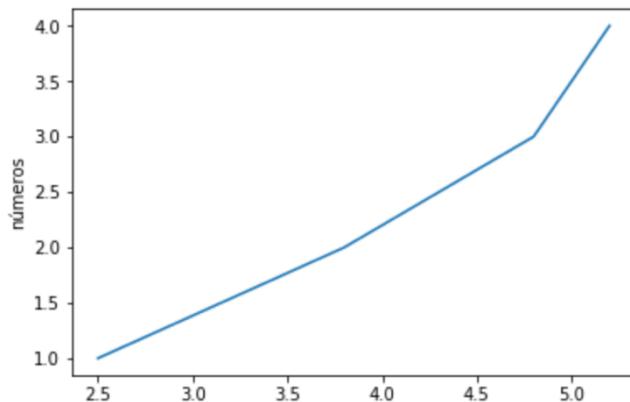
```
import matplotlib.pyplot as plt
```

Nota: la paraula *plt* just al darrere d'*as* és el nom que assignem a Pyplot, a l'hora de fer-lo servir en el nostre codi.

A continuació, farem un gràfic lineal molt senzill. Escrivim el codi que es mostra a continuació en una nova cel·la i l'executem:

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

Si tot va bé, aquest és el gràfic que apareixerà.



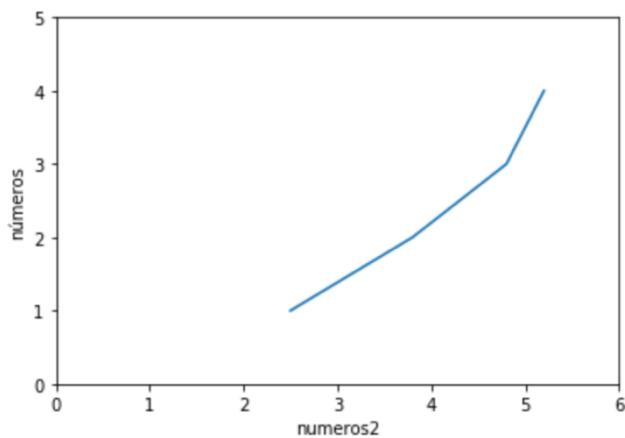
Repassem les línies de codi que hem fet servir per generar el gràfic.

La primera línia, `plt.plot([2.5,3.8,4.8,5.2],[1, 2, 3, 4])`, conté dues llistes: la primera conté els valors de l'eix *x* i la segona els valors de l'eix *y*. La segona línia, `plt.ylabel('números')`, indica el títol de l'eix *y*. La tercera línia, `plt.show()`, és l'ordre que fa imprimir el gràfic. Si volem, podem afegir una altra línia abans de `plt.show()` amb la sentència `plt.xlabel('...')`, per tal de donar nom a l'eix *x*.

Fixem-nos, però, que la intersecció entre l'eix *y* i *x* no mostra el punt (0,0). Si volem mostrar que la intersecció coincideixi amb el punt (0,0), hem de fer servir la funció `axis()`, que també ens permet fixar el valor màxim de l'eix *y* i el de l'eix *x*.

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

Si executem el codi, aquest serà el gràfic que obtindrem:



Normalment, els valors de les llistes que formen els punts de l'eix  $x$  i  $y$  ja vindran donats per una llista que estarà guardada en una variable. Suposem que tenim el nombre d'alumnes aprovats i aprovades d'una assignatura en relació amb diferents cursos. Obrim una altra cel·la i escrivim aquest codi:

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

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

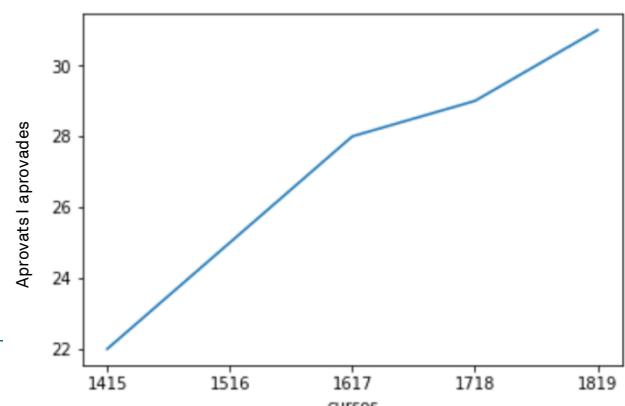
plt.plot(cursos,aprovats)

plt.xlabel('cursos')

plt.ylabel('aprovats i aprovades')

plt.show()
```

Aquest és el gràfic que obtindrem:



La informació que ens mostra el gràfic és que el nombre d'aprovats i aprovades augmenta any rere any.

Anem, ara, a modificar el disseny dels gràfics fent ús de `setp()`. Modifiquem el codi, tot introduint una nova sentència amb la funció `setp()`:

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

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

```
dades = plt.plot(cursos,aprovats)
```

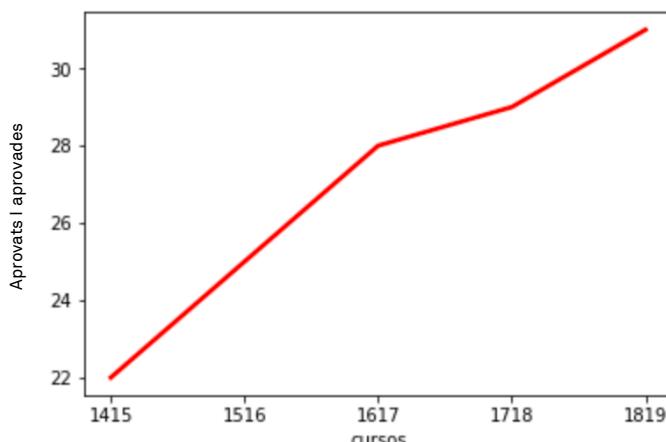
```
plt.xlabel('cursos')
```

```
plt.ylabel('aprovats i aprovades')
```

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

```
plt.show()
```

El gràfic quedarà modificat d'aquesta manera:



La funció `setp()` agafa com a arguments el gràfic format per les dades de l'eix *x* i *y*, el color amb el qual volem pintar la línia i el gruix de la mateixa. Podem modificar aquests paràmetres i el gràfic s'ajustarà a les noves instruccions.

Nota: el paràmetre `linewidth` fa referència al gruix de la línia.

Deixem, ara, els gràfics lineals i passem a construir gràfics de barres. Si agafem l'exemple dels aprovats, només caldrà canviar la funció `plot()` per la funció `bar()`. El codi quedarà així:

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

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

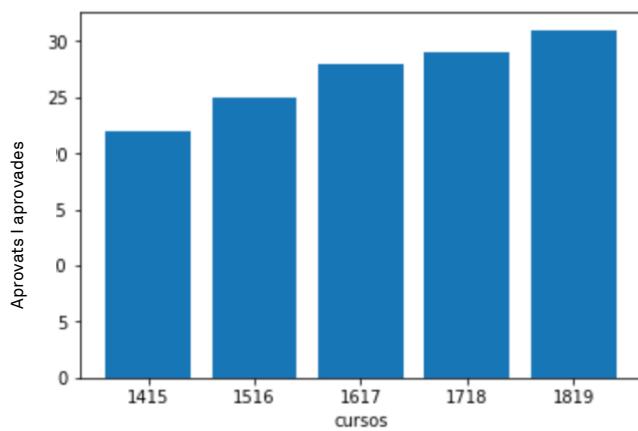
```
plt.bar(cursos,aprovats)
```

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

```
plt.ylabel('aprovats i aprovades')
```

```
plt.show()
```

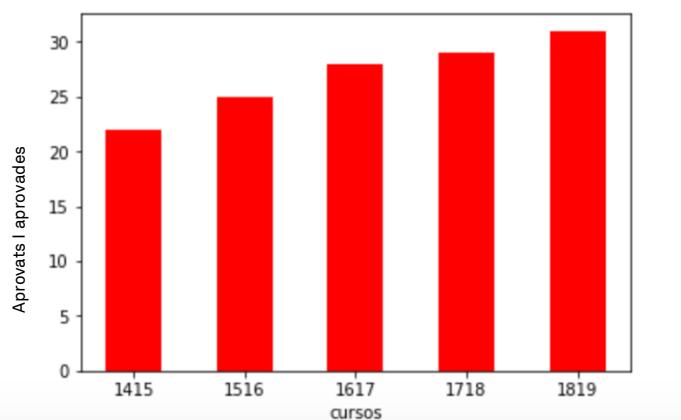
El gràfic que obtindrem és el següent:



La funció `bar()` permet introduir més paràmetres, per tal de personalitzar el disseny del gràfic. Per exemple, podem modificar la funció, escollint el color de les barres i la seva amplada. Ho farem així:

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

El nou gràfic serà aquest:



Nota: el paràmetre *width* fa referència a l'amplada de les barres.

També tenim l'opció d'afegir un color a l'entorn de la barra i donar-li, fins i tot, un gruix. Ho farem així:

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

Nota: *edgecolor* fa referència al color de l'entorn de la barra i *linewidth* al gruix d'aquesta línia.

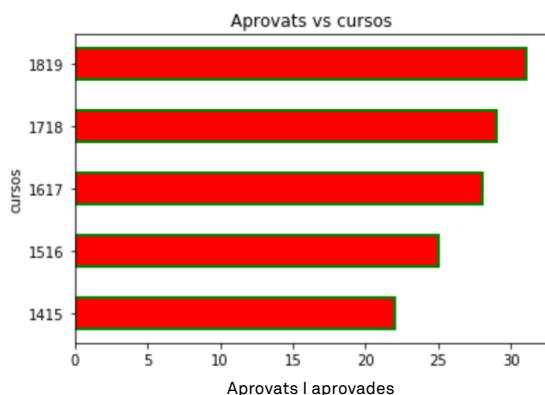
Si, en lloc de treballar amb un gràfic de barres vertical, volem que sigui horitzontal, haurem de canviar de funció. En aquest cas, la funció serà *barh()*. Seguim amb les dades d'aprovats, segons el curs i, escrivim aquest codi:

```
cursos = ['1415', '1516', '1617', '1718', '1819']
aprovats = [22,25,28,29,31]
plt.barh(cursos,aprovats, color='red', height=0.5, edgecolor='green', linewidth='2.0')
plt.xlabel('aprovats i aprovades')
plt.ylabel('cursos')
plt.show()
```

Fixem-nos que els paràmetres de les dues funcions: *bar()* i *barh()* són pràcticament els mateixos, tret del paràmetre *width* de *bar()*, que, en la funció *barh()*, passa a ser *height*. També hem intercanviat els valors de *xlabel* i *ylabel*, per ser coherents amb la nova orientació. Per últim, cal dir que, qualsevol gràfic fet amb el mòdul *plt*, pot tenir un títol només cridant la funció *plt.title()* i, posant com a argument, el mateix títol. Per exemple, podem escriure:

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

Fent el canvi d'orientació i afegint-hi el títol, el nou gràfic pot quedar així:



## 3.2 SCATTERPLOTS

Un gràfic de dispersió (*scatter plot*, en anglès) mostra els punts que relacionen els valors de l'eix *x* amb els valors de l'eix *y*. Si, per exemple, agafem les dades d'alumnes aprovats i aprovades respecte als cursos, com podem generar un gràfic de dispersió? Si canviem la funció *plot()* per *scatter()*, ja ho tindrem. Escrivim aquest codi:

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

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

```
dades = plt.scatter(cursos,aprovats)
```

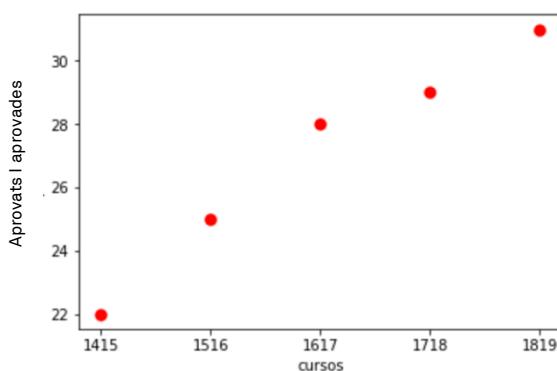
```
plt.xlabel('cursos')
```

```
plt.ylabel('aprovats i aprovades')
```

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

```
plt.show()
```

La funció *plt.scatter* rep els paràmetres que formen els valor de l'eix *x* (cursos) i *y* (persones aprovades). El gràfic que obtenim, quan executem el codi, és:



Si suposem que tenim més d'un grup fent la mateixa assignatura, per exemple, tres grups, podem visualitzar el nombre de persones aprovades per any d'aquests tres grups en el mateix gràfic. Ho farem així:

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

```
aprovats1 = [22,25,28,29,31]
```

```
aprovats2 = [15,20,37,24,30]
```

```
aprovats3 = [20,12,22,30,25]
```

```
plt.scatter(cursos,aprovats1, color='yellow')
```

```
plt.scatter(cursos,aprovats2, color='red')
```

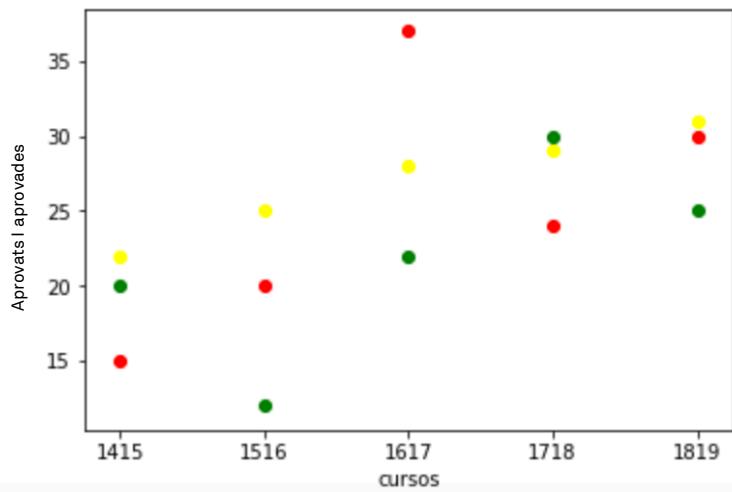
```
plt.scatter(cursos,aprovats3, color='green')
```

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

```
plt.ylabel('aprovats i aprovades')
```

```
plt.show()
```

Els i les alumnes aprovades del grup 1 (aprovats1) es representaran de color groc, els del grup 2 de color vermell i els del grup 3 de color verd.



Anem, ara, a treballar amb una taula de dades més gran. Fes clic [aquí](#) per descarregar-te el fitxer en format CSV. Aquesta taula és un exemple d'una empresa d'assegurances. Després, escriu aquest codi:

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

insurance_data = pd.read_csv(file_path)

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

Veuràs les deu primeres observacions de la taula. Anem a veure la relació entre les dades *bmi* i *charges*:

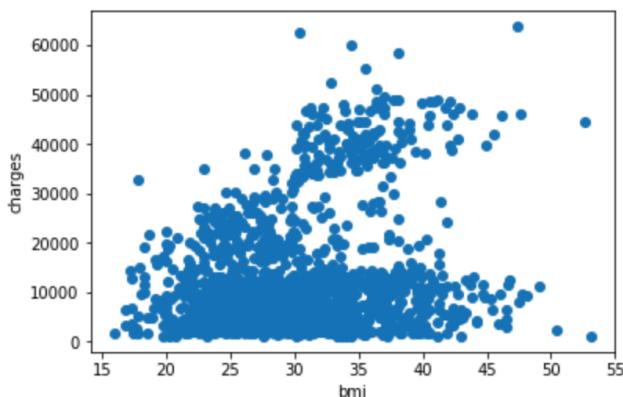
```
plt.scatter(insurance_data['bmi'], insurance_data['charges'])

plt.xlabel('bmi')

plt.ylabel('charges')

plt.show()
```

Aquest és el gràfic que obtenim:



Nota: *bmi*és el valor de *Body Mass Index* i *charges*és l'import de risc de l'assegurança.

Fixa't que, en el codi, la llista de valors de l'eix *x* i l'eix *y* estan dins d'*insurance\_data['bmi']*, *insurance\_data['charges']*.

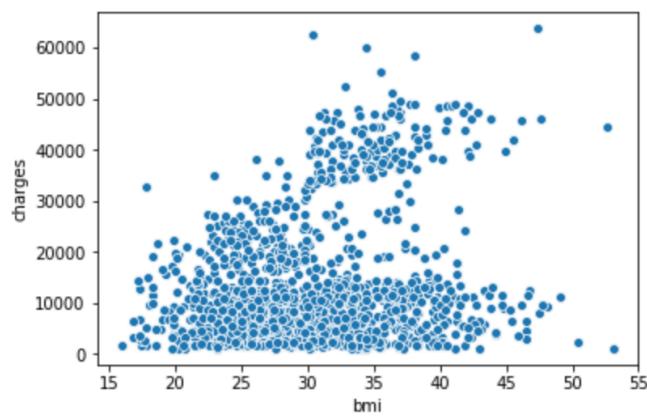
A continuació, utilitzarem una altra llibreria basada en *Matplotlib* que es diu *Seaborn*. Per tal de carregar aquesta llibreria i poder fer-la servir, hem d'importar-la.

```
import seaborn as sns
```

Podem generar un gràfic de dispersió amb les mateixes dades que en el cas anterior:

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

El gràfic que obtenim és el següent:

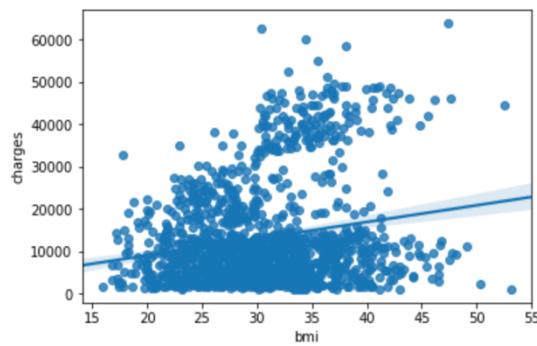


Fixa't que els punts tenen més definició i que no hem hagut de declarar el nom dels eixos. La mateixa funció `scatterplot()` ja els ha generat.

Anem a veure, ara, si hi ha una correlació entre els valors *bmi* i el preu de la pòlissa. Per fer això, hem de fer servir una altra funció anomenada `regplot()`:

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

El gràfic que obtenim és el següent:

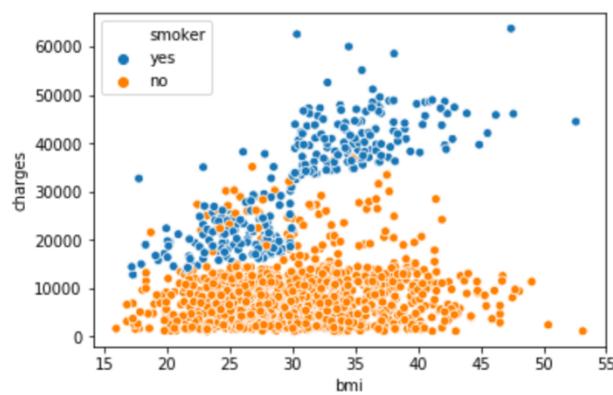


Segons el gràfic, efectivament hi ha una correlació positiva. A mesura que el valor *bmi* d'una persona augmenta, també creix el preu.

Si et fixes en les dades de la taula, veuràs que hi ha una columna anomenada *smoke*, que indica si aquella persona és fumadora o no. Ara farem que els punts del gràfic de dalt tinguin un color o un altre, dependent de si la persona és fumadora o no. Afegim un nou paràmetre a la funció *scatterplot()*:

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

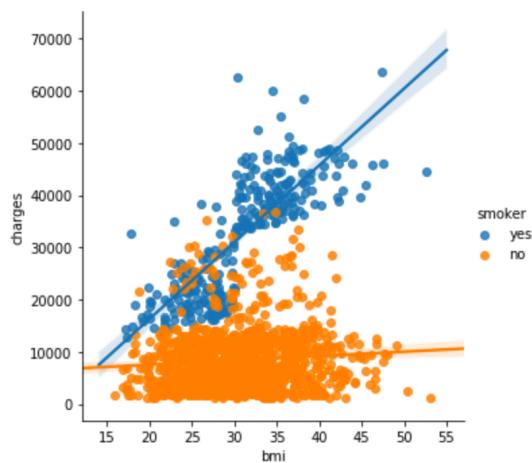
El gràfic que obtenim és el següent:



Aquí veiem que, clarament, si la persona és fumadora, el preu de la seva assegurança serà més alt. I, si volem veure la relació entre *bmi* i *charges* és diferent, en funció de si una persona és fumadora o no hem d'executar aquest codi:

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

El gràfic generat per la funció *lmplot()* és el següent:



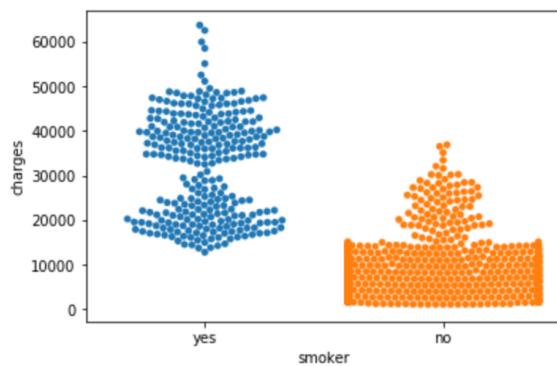
Efectivament, veiem que, per a una persona fumadora, el preu de la seva pòlissa augmenta considerablement quan té un *bmi* més alt, mentre que les persones no fumadores tenen un augment molt més petit.

Fins ara, hem fet gràfics de dispersió entre dues variables numèriques, però també podem fer que una variable sigui categòrica (no numèrica). Per exemple, podem veure la dispersió de preus de les pòlisses, en funció de si la persona és fumadora o no.

En aquest cas, farem servir la funció *swarmplot()*. Escriurem el codi següent:

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

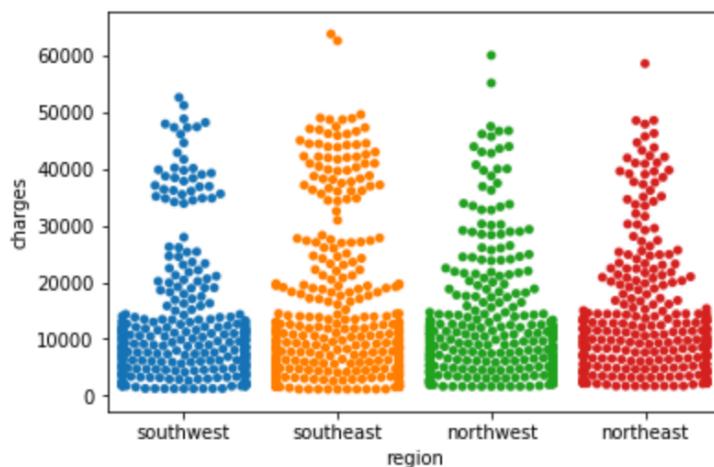
El gràfic que genera la funció és aquest:



Aquest gràfic ens ensenya que la gran majoria de persones no fumadores tenen un preu inferior al de les persones fumadores.

Si ara volem veure com són els preus, depenent de la regió, hem d'escriure el codi següent:

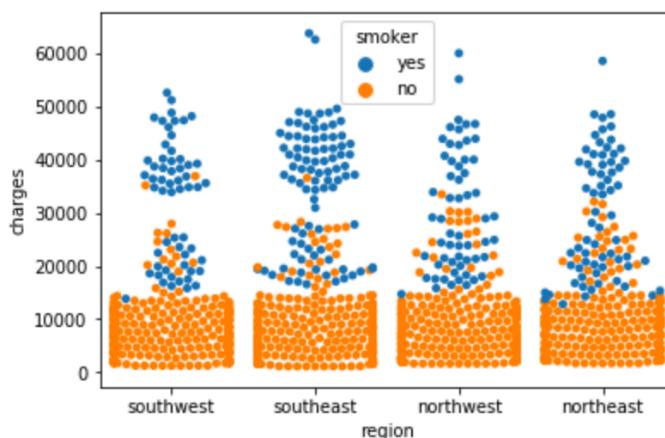
```
sns.swarmplot(x=insurance_data['region'], y=insurance_data['charges'])
```



Amb el gràfic generat, veiem la dispersió de preus agrupats en les quatre regions que té la taula. En principi, la majoria de la població no presenta diferents preus, segons la seva regió. I què passa, si afegim a l'anàlisi la variable *smoke?* Per saber-ho, haurem d'afegir a la funció *swarmplot* el paràmetre *hue* d'aquesta manera:

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

I visualitzem aquest gràfic:



Aquí, una altra vegada, comprovem que el factor més determinant per veure com varia el preu d'una assegurança és saber si la persona és fumadora o no.

Seguim!

### 3.3 EXEMPLE PER GENERAR ELS GRÀFICS APRESOS

Hola!

Repassem el que hem fet fins ara i descobrim alguna eina nova.

Com pots veure, tenim un projecte nou amb els mòduls que necessitarem. Tu ja saps com crear un projecte i també com importar els mòduls, així que ja podem començar.

Primer, carreguem la base de dades que té per nom *insurance.csv* amb la qual hem treballat. Aquesta base de dades conté dades d'una mostra de persones sobre la seva edat, sexe, bmi (***Body Mass Index***), si té fills o no, si és fumadora o no, la regió on viu i el preu de la seva pòlissa d'assegurança.

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

Obtindrem un resum de la informació més important que hi conté, fent ús de la funció `describe()`. Afegim aquest codi a la cel·la:

```
insurance_data.describe()
```

I ja podem executar-ho.

Amb les dades que ens torna la funció `describe()`, podem veure que l'edat mitjana és de 39,20 anys, essent l'edat màxima 64 anys i la mínima 18. Pel que fa a la dada `bmi`, la mitjana és de 30,66, essent el valor més gran 53,13 i el més petit 15,96. Les persones d'aquesta taula tenen una mitjana de 1,09 fills o filles, essent 5 i 0 el valor màxim i mínim respectivament. Per últim, podem veure que el preu mitjà és 13.270, el valor màxim és 63.770 i el preu mínim 1.121.

Aquesta primera anàlisi ja ens dona molta informació sobre les dades de l'estudi.

Vegem un histograma que ens ensenyi quantes observacions tenen 0, 1, 2, 3, 4 o 5 fills o filles. Ho escriurem així:

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

Veiem que molta gent no té fills ni filles, i molt poca en té 4 o 5.

Visualitzem, ara, els punts de dispersió que relacionen els `bmi` i els preus, i veiem la diferència en funció del nombre de fills o filles. Aquest és el codi.

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

El gràfic és força visual, amb diferents colors, ja que hem imposat que els punts assoleixin un color diferent, en funció del nombre de fills o filles que tinguin. Aquí, la dependència queda marcada en el codi que hi ha dins la funció `scatterplot()` amb el paràmetre `hue`. En qualsevol cas, la veritat és que la informació que podem extreure'n no és de qualitat, perquè no es pot veure cap tendència. Anem, doncs, a fer ús de la funció `lmplot()`, per tal de generar una recta de regressió per a cada grup, en funció del nombre de fills o filles.

El codi que hem d'escriure és:

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

Amb aquesta gràfica, el que veiem clarament és que, per a tots els casos (excepte per a aquells en els quals el nombre de fills o filles és 5), els preus augmenten a mesura que augmenta el `bmi`. És curiós veure com, per al cas de 5 fills o filles, el preu disminueix amb un

*bmi* més gran. Possiblement, la realitat serà diferent, o no, però certament el cas requereix un estudi més profund. En qualsevol cas, la zona difusa que hi ha entorn de cada línia indica prou marge d'error com per a què la correlació sigui positiva.

## 3.4 IDEES CLAU: VISUALITZACIÓ DE DADES AMB MATPLOTLIB

En aquest apartat, hem après a extreure informació de les dades mitjançant representacions gràfiques, les quals ens ajuden a visualitzar tendències i patrons amb només un cop d'ull.

Concretament, hem treballat amb gràfics de línia, de barra i de dispersió.

Els **gràfics de línia** mostren una línia continua que enllaça tots els punts que relacionen els valors de l'eix *x* i el valor de l'eix *y* corresponents.

Els **gràfics de barres** són aquells que utilitzen barres, les quals representen l'altura que té un punt respecte l'eix *x*.

Els **gràfics de dispersió** mostren els punts i les coordenades formades pel valor en l'eix *x* i el valor en l'eix *y* corresponent.

Per tal de construir els diferents tipus de gràfics, hem treballat amb funcions dels mòduls *Matplotlib.pyplot* i *Seaborn*.

El mòdul Pyplot proporciona funcions molt útils per generar els diferents gràfics. Per treballar amb gràfics de línia, farem servir *plot()*, de barra *bar()* i de dispersió *scatter()*. Aquestes funcions incorporen diferents paràmetres per configurar els gràfics. Per exemple, *xlabel()* dona nom a l'eix *x*, *ylabel()* a l'eix *y*, *title()* serveix per posar un títol, *color()* proporciona el color, *linewidth()* l'amplada de la línia, etc.

## 4 ANÀLISI DE DADES AMB PANDA

Cada projecte, sigui gran o petit, incorporarà una base de dades amb la qual haurem de treballar. Independentment del volum, aquesta base de dades, probablement, requerirà una sèrie d'accions per tal de descobrir la informació rellevant i útil que conté.

En aquest mòdul aprendrem a treballar i processar bases de dades. Sabrem com crear-ne una des de zero i, després, llegir-ne el contingut i escriure-hi de nou. Tot seguit, farem front al primer bloc d'operacions bàsiques de processament de base de dades, que estarà format per les accions de seleccionar, filtrar i indexar. A continuació, aprendrem a ordenar i agrupar dades i a gestionar dades no disponibles.

Finalment, posarem a prova els nous coneixements amb un exercici pràctic. Certament, l'habilitat en saber gestionar bases de dades s'adquireix, primer, amb una bona base teòrica i, després, necessàriament, amb exercicis pràctics. Comencem!

## 4.1 CREAR, LLEGIR I ESCRIURE UNA TAULA DE DADES

Començarem aprenent a crear una base de dades des de zero. El primer que hem de fer, una vegada ja tenim el projecte iniciat amb Jupyter Notebook, és importar la llibreria Pandas. Ho farem de la mateixa manera que en els mòduls anteriors:

```
import pandas as pd
```

Recorda que una base de dades està formada per files i columnes. En construirem una que, inicialment, només té una fila i dues columnes. La primera columna indica que hi ha 30 pomes i la segona que hi ha 20 maduixes. Aquest és el codi:

```
fruites = pd.DataFrame([[30,20]], columns=['Pomes', 'Maduixes'])
```

```
fruites
```

La taula que veiem una vegada executem el codi és:

	Pomes	Maduixes
0	30	20

El que fa el codi que hem escrit és generar un objecte (una base de dades que anomenem *fruites*) a partir del seu objecte superior DataFrame, tot assolint una sèrie de paràmetres que s'especifiquen dins del parèntesi. El primer paràmetre indica els valors de les files i el segon el nom que tindrà cada columna. Tant el valor de les files com el nom de les columnes s'introdueixen fent ús de llistes.

Avancem una mica més i afegim, ara, més files a aquesta base de dades. Escrivim aquest codi:

```
fruites_compradors=pd.DataFrame([[30,20],[28,14]],index=['Miquel','Julia'],columns=['Pomes', 'Maduixes'])
```

```
fruites_compradors
```

Si ens fixem en el primer paràmetre, allà on definim el valor de les files, el que hem fet és afegir-hi una altra llista (una altra fila). Després, hem definit un altre paràmetre anomenat *index* que serveix per donar nom a les files.

	Pomes	Maduixes
Miquel	30	20
Julia	28	14

Fins aquí hem construït una base de dades a partir de valors agrupats en files. També podem crear-ne una introduint-hi les dades agrupades en columnes. Fem-ho:

```
fruites_compradors_2 = pd.DataFrame({'Pomes':[30,28], 'Maduixes':[20,14]}, index=['Miquel', 'Julia'])
```

```
fruites_compradors_2
```

Hem creat una nova variable on guardem la base de dades que s'ha introduït amb els valors agrupats en columnes. Essencialment, el codi que hem escrit ha de generar la mateixa base de dades que en el cas anterior.

	Pomes	Maduixes
Miquel	30	20
Julia	28	14

Efectivament, veiem que la nova base de dades presenta els mateixos valors que l'anterior. Si estudiem el codi, veiem que les columnes s'han introduït fent ús de diccionaris, els quals tenen, com a clau, el nom de la columna i, com a valors, les dades que formen aquesta columna. Fixem-nos que, per a cada columna, les dades s'introdueixen mitjançant una llista.

Tot i que les dues bases de dades presenten la mateixa informació, no són realment la mateixa base de dades. De fet, són objectes diferents perquè cadascun està guardat en un espai de memòria diferent (en dues variables diferents). Comprovem-ho:

```
print(fruites_compradors_2 is fruites_compradors)
```

El resultat que ens torna Python és *False*, la qual cosa confirma la nostra sospita. Per tal de fer que les dues bases de dades siguin realment el mateix objecte, el que hem de fer és que apuntin al mateix espai de memòria. Així que necessitem escriure aquesta línia de codi:

```
fruites_compradors = fruites_compradors_2
```

Comprovem, ara, si són el mateix objecte:

```
print(fruites_compradors_2 is fruites_compradors)
```

Confirmat, ara són el mateix. Python ens torna *True* i, per tant, podem afirmar que sí que ho són. Això vol dir que, si fem un canvi en qualsevol de les dues bases de dades (*fruites\_compradors* o *fruites\_compradors\_2*), aquest canvi també s'aplicarà en l'altra base de dades perquè s'han convertit en el mateix objecte. Comprovem-ho:

Primer de tot, assignem un valor nou a la primera posició de la columna que hem anomenat *Pomes*, de la primera base de dades creada:

```
fruites_compradors['Pomes'][0]=15
```

Per tal d'assignar aquest valor nou a la posició desitjada, hem de cridar la base de dades que volem canviar, assignar-li la columna (*Pomes*) i fixar la posició (0).

Comprovem que el canvi s'ha materialitzat correctament:

```
print(fruites_compradors['Pomes'][0])
```

Python ens torna 15, la qual cosa vol dir que el canvi s'ha fet. I, ara, només cal veure si el canvi també es pot observar si fem servir el nom de la variable *fruites\_compradors\_2*:

```
print(fruites_compradors_2['Pomes'][0])
```

Sí, també obtenim 15.

Si el canvi el fem començant per la segona base de dades creada, aquest també es podrà veure quan fem ús del nom de la variable que fa referència a la primera base de dades creada. Escrivim aquest bloc de codi:

```
fruites_compradors_2['Pomes'][0]=2
```

```
print(fruites_compradors_2['Pomes'][0])
```

```
print(fruites_compradors['Pomes'][0])
```

Correcte. En els dos casos, veiem que el valor de la primera posició de la columna *Pomes* ara és 2.

Si volem veure com queda l'objecte final i volem comprovar que, fent ús de qualsevol nom de variable, ens imprimeix la mateixa taula, només cal anomenar el nom de cada variable:

```
fruites_compradors
```

	<b>Pomes</b>	<b>Maduixes</b>
<b>Miquel</b>	2	20
<b>Julia</b>	28	14

*fruites\_compradors\_2*

	Pomes	Maduixes
Miquel	2	20
Julia	28	14

Efectivament, veiem que el canvi s'ha materialitzat en la primera posició de la columna *Pomes*.

Nota: recorda que la primera posició en una llista s'identifica amb un 0 i no amb un 1.

## 4.2 INDEXAR, SELECCIONAR I FILTRAR

Seguim avançant i aprofundint en les bases de dades.

A continuació, crea un nou projecte amb Jupyter Notebook i descarrega't aquesta base de dades des d'aquest [enllaç](#). Guarda-la a la mateixa carpeta on tens el projecte nou.

Primer, carreguem la llibreria *Pandas* i li assignem el nom *pd*:

```
import pandas as pd
```

Importem la base de dades i la guardem en una variable anomenada *data\_base*. Després, veiem les cinc primeres observacions:

```
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

Aleshores, ens apareixen totes les columnes amb les seves cinc primeres observacions, però no sabem com és de gran aquesta base de dades. Si volem imprimir només un determinat nombre de files, però volem, a més, veure el nombre total de files, ho haurem de fer així:

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

```
data_base
```

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

1338 rows × 7 columns

Aquí, ens apareixen els títols de les columnes i quatre files amb les seves dades. A sota de la taula, veiem que està formada per 1.338 files (observacions) i 7 columnnes (atributs o variables).

Si volem específicament la informació que conté una variable determinada, només haurem de picar aquest codi:

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

Python ens mostrerà aquesta informació:

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

Ara, veiem només informació corresponent a la variable *region*.

Si vols veure més observacions, hauràs de canviar el paràmetre de la funció *set\_option()*. En comptes de cinc, n'hi pots posar deu. Si fas el canvi, hauràs de tornar a executar els codis, per tal de veure més files a la base de dades.

Nota: tu pots escollir una altra variable, com pot ser *smoker*, *sex*, *age*, etc.

Suposem, ara, que vols saber el valor més gran que té la columna *bmi*. Això ho pots saber, cridant la funció *describe()*. Te'n recordes? O bé seleccionant només la columna *bmi* i fent ús de la funció *max()*:

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

En tots dos casos, el resultat és 53,13.

I com ho podem fer per seleccionar només els 30 primers valors d'una columna? Doncs així:

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

```
bmi_30
```

Ara, mirem a veure quin és el valor màxim dels 30 primers valors de la columna *bmi*:

```
bmi_30.max()
```

El valor obtingut és 42,13. El valor 53,13, doncs, no està dins dels 30 primers valors.

Anem a indexar, ara, no només valors d'una columna, sinó files i columnes conjuntament. Si volem, per exemple, treballar amb les deu primeres files i les dues primeres columnes, com ho fem? Necessitem treballar amb l'operador *iloc* i ho farem així:

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

El que hem fet, aquí, és dir que volem les deu primeres files (0:10) i les dues primeres columnes (0:2) de *data\_base*. Hem guardat aquesta taula reduïda en un nou espai de memòria, a través d'un nom nou que és *data\_base\_2*. Això és un objecte nou, una nova base de dades. Si cridem aquesta nova base de dades, obtenim:

```
data_base_2
```

Podem, també, en comptes de seleccionar un interval de files o columnes, seleccionar específicament unes determinades files o columnes:

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

```
data_base_3
```

	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

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

El que hem fet aquí és obtenir les files 0,5 i 6 i les columnes 0 i 4. Després, les hem guardat com una altra base de dades reduïda en un altre espai de memòria.

Per a les columnes, podem fer servir els seus noms propis per seleccionar-les. En aquest cas, hem de fer servir l'operador *loc*. Mirem aquest exemple:

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

```
data_base_4
```

El que hem fet és seleccionar les vuit primeres files de les columnes *children* i *age*. Les dades que es mostren són les següents:

Fem un pas endavant i anem a crear taules reduïdes a partir de la taula mare, no fent ús dels índexs de files o columnes, sinó de criteris condicionals. Imagina que només volem aquelles observacions que es troben a la regió *southwest*. Aquest és el codi que necessitem:

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

I volem treure, per pantalla, només les set primeres files i les dues primeres columnes:

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

La taula que es veu per pantalla és aquesta:

Fixa't que la numeració de files fa referència a les files 0, 12, 15, 18, 19, 21 i 29, les quals són les set primeres files que pertanyen a la regió *southwest* de la taula mare *data\_base*.

Si volem veure aquelles observacions de la base de dades principal que tinguin un *bmi* entre 35 i 45, haurem de declarar dos condicionals:

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

	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

Hem aplicat la funció `describe()`, per veure quins valors màxims i mínims té la columna `bmi` d'aquesta taula reduïda:

	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

Efectivament, veiem que, en aquesta taula reduïda, només tenim aquelles files que tenen un valor de `bmi` entre 35 i 45.

Per últim, anem a crear una columna nova. Els seus valors dependran del valor de `bmi` de cada fila. El codi és el següent:

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

El que estem fent és donar un valor a cada fila de la columna `approved` de `True` o `False`, segons si el valor de `bmi` de la mateixa fila és superior o inferior a 45. Vegem els cinc primers registres amb la nova columna:

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

## 4.3 ORDENAR I AGRUPAR DADES NO DISPONIBLES

En aquest article, començarem descobrint una nova funció que ens permetrà generar una sèrie de dades agrupades entorn dels valors d'una columna. El primer pas, com és habitual, és carregar la llibreria i crear un nou objecte amb la base de dades que hem fet servir en l'article anterior:

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

Anem a veure com es distribueixen els valors de *region*:

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

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

Aquí veiem quantes observacions tenen cada una de les diferents regions de la taula mare.

Si volem veure quin és el valor màxim de *bmi* per a cada regió, fem:

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

```

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

```

I, si volem saber la mitjana de *bmi* per a cada regió, teclegem:

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

```

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

```

Aquí, veiem que la mitjana de valors de *bmi* més elevada la trobem a la regió *southeast* i la més baixa a la *northeast*.

Tenim l'opció de generar la llista d'aquestes mitjanes de manera ordenada. Ho podem fer amb la funció *sorted()*. El codi seria aquest:

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

La llista que obtindríem seria:

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

Ara, farem una agrupació més complexa. Mostrarem les mitjanes dels preus de les pòlies agrupades, en un primer nivell, per la regió i, en un segon nivell, per la variable *smoker*. Aquest és el codi:

```
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

```

Veiem clarament que, independentment de la regió, la mitjana de preus sempre és més alta, si la persona és fumadora, que quan no ho és.

Si ara mantenim l'agrupació de la regió, però canviem la variable *smoker* per *sex* en el segon nivell de l'agrupació, necessitem només fer aquest petit canvi:

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

La taula que obtenim és:

```

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

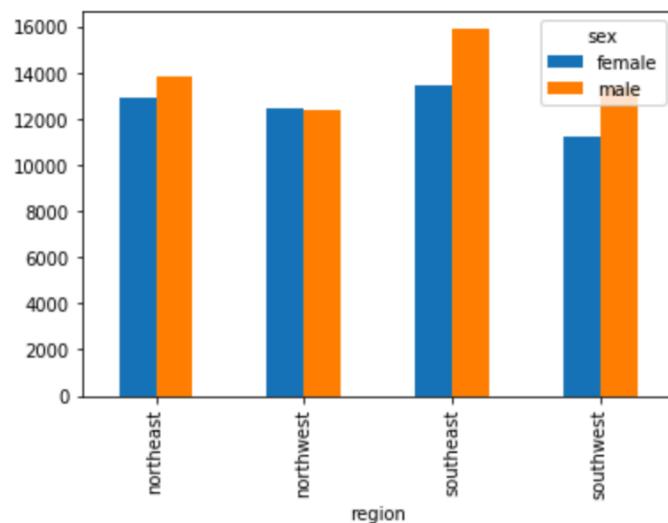
```

Veiem que, en totes les regions, excepte a *southwest*, la mitjana de preus és més alta per als homes que per a les dones.

Si assignem aquesta agrupació a una nova variable i apliquem la funció *plot.bar()*, obtindrem una gràfica de barres coherent amb l'agrupació:

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

```
group.unstack(fill_value=0).plot.bar()
```



El gràfic mostra, d'una manera molt més visual, el que la taula de dalt ens deia.

Si, donada la base de dades *data\_frame*, volem mostrar-la de manera ordenada segons els valors d'una variable determinada, només ens cal fer servir la funció *sort\_values()*. Ho farem així:

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

Si volem mostrar només els primers 12 registres d'aquest ordre, haurem d'aplicar la funció *head()* i especificar el nombre de registres que volem mostrar:

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

	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 taula que obtenim, doncs, serà aquesta:

El primer criteri per ordenar les files correspon al valor de l'edat (*age*) i el segon correspon a *bmi*. El codi, primer, ordena per edats i, en cas que hi hagi files amb la mateixa edat, el codi aplica un segon criteri, el *bmi*. Per defecte, l'ordre s'aplica sempre de manera ascendent, és a dir, de valor més petit a més gran. Si això ho volem invertir, haurem d'afegir un altre paràmetre d'aquesta manera:

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

Ara, la taula que obtenim és la següent:

	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

El paràmetre *ascendint*, per defecte, assoleix el valor *True* i és per això que les taules sempre mostren un ordre ascendent. Si especifiquem que l'*ascending* ha de tenir un valor *False*, obtindrem una taula amb ordre descendent, com aquesta última que hem obtingut.

Fins ara, hem suposat que totes les cel·les d'una taula tenien valors. La realitat, en canvi, sempre és diferent i aquesta ens presentarà, sovint, taules que tenen cel·les sense valors. Aquests valors es diuen *Nas*.

Una de les funcions més utilitzades del paquet Pandas, per veure si tenim *Nas* o no, és la funció *isna()*. Aquesta funció torna, com a resultat, *False*, quan el valor existeix i torna *True*, quan el valor és *Na*. Per exemple, si volem saber si entre els registres 230 i 238 de la columna *smoker* hi ha *Nas* o no, necessitem escriure aquest codi:

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

La llista de valors que obtenim és:

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

Observant els resultats, podem concloure que, en aquest interval de valors, no tenim cap *Na*, ja que tots els valors que torna la funció *isna()* són *False*.

Anem, ara, a copiar la base de dades *data\_frame* en una nova variable:

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

Important: *data\_frame2* i *data\_frame* no són el mateix objecte. No hem apuntat realment *data\_frame* a la nova variable, ja que n'hem fet servir una còpia amb l'operador `[:]`.

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

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

La taula que obtenim ara sí que té un *Na*, segons indica la funció *isna()*:

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

La funció `notna()` funciona justament al contrari. Allà on hi ha un `Na`, torna un `False` i, on no n'hi ha, torna un `True`. Ho veiem?

```
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
```

Seguim!

## 4.4 REPÀS DEL TEMARI AMB EXERCICI PRÀCTIC

Hola a tothom!

Repassem el que hem fet fins ara amb uns exercicis.

Obre un projecte nou amb Jupyter Notebook i introduceix aquest codi que apareix en pantalla. Primer, carrega la llibreria `Pandas` i, després, crea una base de dades que guardem en una variable que podem anomenar, per exemple, `notes`.

```
In [1]: 1 import pandas as pd
In [2]: 1 notes = pd.DataFrame([[6.5, 6.8, 5.8, 7.9, 8.8, 'Barcelona'],
                           [7.8, 6.2, 6.5, 4.1, 8.0, 'Tarragona'],
                           [6.9, 7.2, 4.7, 8.8, 6.2, 'Lleida']],
                           columns=['Història', 'Matemàtiques', 'Física', 'Anglès', 'Biologia', 'Ciutat'],
                           index = ['Joan', 'Laura', 'Enric', 'Georgina'])
Out[2]:   Història Matemàtiques Física Anglès Biologia Ciutat
          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
```

Bé, veiem que es tracta d'una taula que recull les notes de quatre estudiants i la seva ciutat de procedència. Per començar, revisem com podem extreure la nota que ha tret la Laura a Física i la ciutat d'en Joan. Saps com fer-ho?

Només cal mencionar el nom de la base de dades, seleccionar la columna de l'atribut, en aquest cas `física` o `ciutat` i, després, indicar la posició de l'observació, és a dir, la posició de l'estudiant del qual volem extreure'n la informació:

```
In [3]: 1 notes['Física'][1]
         2 notes['Ciutat'][0]
```

T'has fixat que estem treballant amb diccionaris? Sí, el nom de la columna és la clau que dona accés a la seva llista de valors i, només indicant la posició, obtenim la dada que volem.

Anem, ara, a fer un filtre. Si volem veure les observacions (estudiants) que han aprovat física, com ho faríem? Doncs anomenem la base de dades i, dintre dels claudàtors, introduïm la condició. És fàcil, oi?

```
In [4]: 1 notes[notes.Física > 5]
```

Si mostrem la taula, veurem que la Georgina és qui no ha aprovat física. I si ha estat un error i realment sí que ho ha aprovat? Si la seva nota és de 7.0, com la podem canviar? Doncs bé, només cal accedir a la seva nota i canviar-la.

	Història	Matemàtiques	Física	Anglès	Biologia	Ciutat
<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 notes['Física'][3] = 5.0
         2 notes
```

Si mostrem la taula de nou, veurem que el canvi s'ha aplicat amb èxit. Ara ningú no ha suspès Física. I això són bones notícies!

Out[6]:

	Història	Matemàtiques	Física	Anglès	Biologia	Ciutat
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	5.0	8.8	6.2	Lleida

A continuació, extraiem una taula reduïda que estigui formada per les notes de Biologia de només les noies. La clau està en fer servir l'operador `loc[]`.

In [6]: 1 notes.loc[['Laura', 'Georgina'], ['Biologia']]

Out[6]:

	Biologia
Laura	9.2
Georgina	6.2

Per últim, anem a fer quelcom una mica més difícil. Afegim una columna que mostri les notes de cada estudiant i la seva desviació. Construïm el codi! Primer, hem de crear dues llistes. Després, crear un cicle `for` que ens permeti recórrer la taula fila per fila. Per això farem servir l'operador `iloc[]`. L'operador formarà una llista, a la qual podrem aplicar la funció `max()` per calcular la mitjana, i l'operador `std()`, per calcular la desviació. Important! Cada vegada que apliquem les funcions, el resultat s'ha de carregar a les llistes corresponents: la mitjana a la seva llista i la desviació a la seva. És important donar noms coherents a les variables, per fer que el nostre codi sigui fàcil d'entendre, sense haver d'afegir-hi comentaris.

```
In [12]: 1 mitjanes = []
2 desviacio = []
3 for i in range(4):
4     mitjana = notes.iloc[i, 0:5].mean()
5     desv = notes.iloc[i, 0:5].std()
6     mitjanes.append(mitjana)
7     desviacio.append(desv)
8 notes['Mitjanes'] = mitjanes
9 notes['Màxima'] = desviacio
10 notes
```

Out[12]:

	Història	Matemàtiques	Física	Anglès	Biologia	Ciutat	Mitjanes	Màxima
<b>Joan</b>	6.5	6.8	5.8	7.9	8.8	Barcelona	7.16	1.188697
<b>Laura</b>	6.8	7.0	6.5	8.9	9.2	Girona	7.68	1.267675
<b>Enric</b>	7.8	6.2	5.0	4.1	8.0	Tarragona	6.22	1.706458
<b>Georgina</b>	6.9	7.2	4.7	8.8	6.2	Lleida	6.76	1.494323

És bàsic practicar per avançar. Anima't i procura jugar una mica més amb aquesta base de dades. Aquí, hem vist diverses coses, però segur que pots trobar més accions a fer.

Sort!

## 4.5 IDEES CLAU: ANÀLISIS DE DADES AMB PANDA

En aquest mòdul hem après a treballar amb bases de dades i hem vist com poden ser de potents, si sabem com aplicar-hi diverses accions destinades a l'anàlisi de dades.

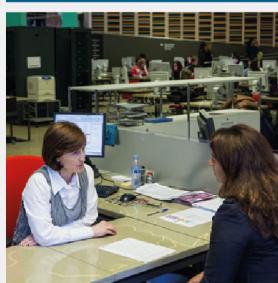
Hem començat a **construir-ne una des de zero**, fent ús de la classe **DataFrame**. Després, hem vist com llegir i modificar bases de dades, tot accedint a les seves posicions.

Aplicant els operadors *iloc[]* i *loc[]*, podem **crear bases de dades més petites**, partint d'una base de dades principal.

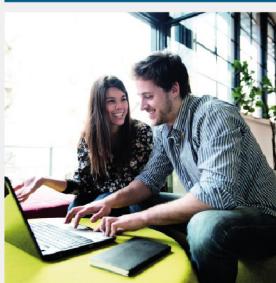
També hem après a **seleccionar files**, depenent d'una determinada condició, o més d'una, fet que ens permet filtrar la base de dades en relació amb els diferents valors que poden tenir els atributs o variables.

Podem **agrupar bases de dades** entorn als valors d'una o més variables. Una vegada hem agrupat aquests valors, és possible aplicar diferents funcions com, per exemple, la mitjana, el nombre de casos o les desviacions d'aquestes agrupacions.

# Descobreix tot el que Barcelona Activa pot fer per a tu



Acompanyament durant tot el procés de recerca de feina  
[barcelonactiva.cat/treball](http://barcelonactiva.cat/treball)



Suport per posar en marxa la teva idea de negoci  
[barcelonactiva.cat/emprendoria](http://barcelonactiva.cat/emprendoria)



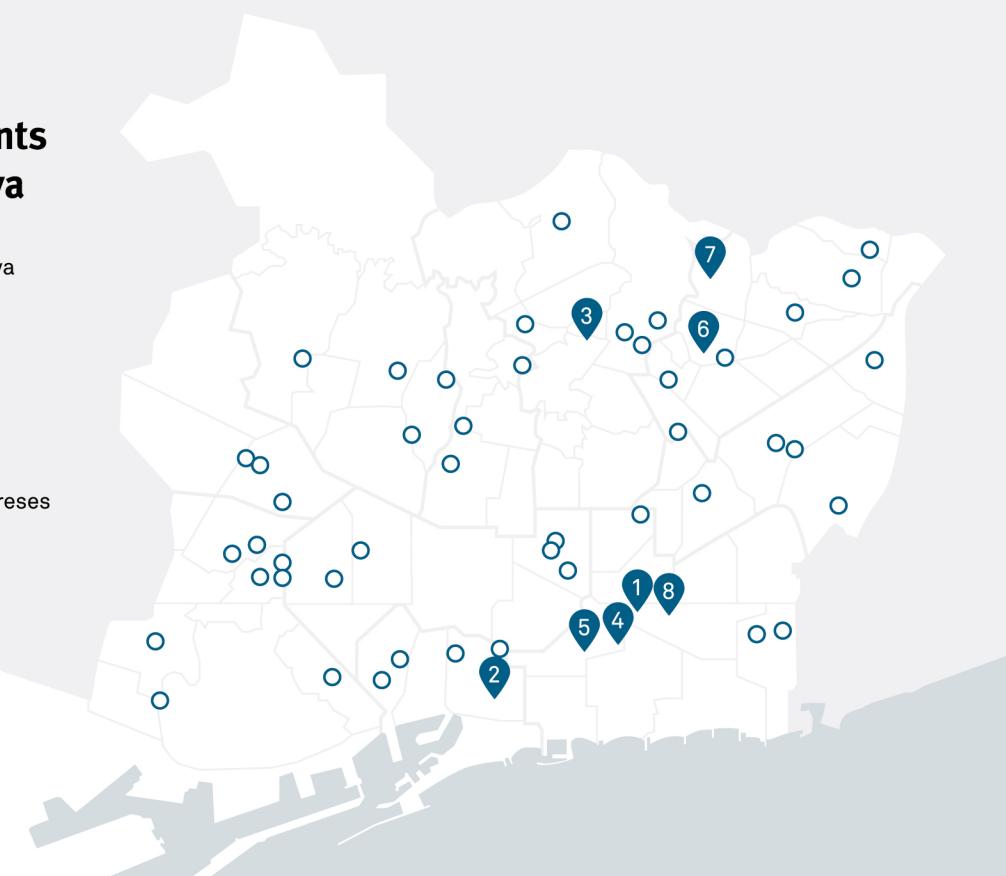
Serveis a les empreses i iniciatives socioempresariales  
[barcelonactiva.cat/empreses](http://barcelonactiva.cat/empreses)



Formació tecnològica i gratuïta per a la ciutadania  
[barcelonactiva.cat/cibernarium](http://barcelonactiva.cat/cibernarium)

## Xarxa d'equipaments de Barcelona Activa

- 1 Seu Central Barcelona Activa  
Porta 22  
Centre per a la Iniciativa Emprenedora Glòries  
Incubadora Glòries
- 2 Convent de Sant Agustí
- 3 Ca n'Andalet
- 4 Oficina d'Atenció a les Empreses Cibernàrium  
Incubadora MediaTIC
- 5 Incubadora Almogàvers
- 6 Parc Tecnològic
- 7 Nou Barris Activa
- 8 innoBA
- Punts d'atenció a la ciutat



© Barcelona Activa  
Darrera actualització 2019

Cofinançat per:



Segueix-nos a les xarxes socials:

-  [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)