Apunts CE_5073 3.1

lloc:	Institut d'Ensenyaments a Distància de les Illes Balears	Imprès per:	Juan Ignacio Akrich Vazquez
Curs:	Programació d'intel·ligència artificial	Data:	divendres, 19 de desembre 2025, 14:43
Llibre:	Apunts CE_5073 3.1		

Taula de continguts

1. Introducció

2. Biblioteques per a IA
2.1. Aprenentatge automàtic
2.2. Processament del llenguatge natural
2.3. Xarxes neuronals
2.4. Visió per computador
2.5. Sistemes experts
2.6. Robòtica

3. Gestió de dependències i entorns virtuals
3.1. pip
3.2. virtualenv
3.3. venv
3.4. Poetry
3.5. Conda

4. Cas pràctic: configuració de l'entorn
4.1. Instal·lació de Conda
4.2. Primer entorn virtual
4.3. Creació d'un script Python
4.4. Configuració dels quaderns Jupyter

5. Cas pràctic: implementació
5.1. Preparació de les dades
5.2. Anàlisi d'importància de les propietats
5.3. Enginyeria de propietats
5.4. Entrenament del model
5.5. Prediccions
5.6. Serialització del model
5.7. Desplegament del model
5.8. I què més?

6. Cas pràctic: publicació del projecte

7. Bibliografia

1. Introducció

En el lliurament anterior vàrem veure les biblioteques **NumPy**, **pandas** i **Matplotlib**, base per a moltes tasques relacionades amb la ciència de dades i la intel·ligència artificial. Però existeixen moltes altres biblioteques amb propòsits més específics dins de l'àmbit de la intel·ligència artificial i l'aprenentatge automàtic. En l'apartat 2 mencionarem les més importants en diversos camps d'aplicació.

El fet d'haver de treballar amb multitud de biblioteques sovint ens planteja un nou problema: potser per un projecte necessitam una versió concreta de l'intèrpret de Python o d'una biblioteca concreta, però per un altre projecte necessitam altres versions. Per solucionar-ho tenim els gestors de dependències i d'entorns virtuals, que permeten definir entorns aïllats de la resta, on podrem treballar amb les versions específiques que necessitam tant de Python com de les biblioteques que hagi d'emprar. A l'apartat 3 introduirem algunes de les eines més utilitzades per a aquest propòsit. Una d'elles, Conda, és la que emprarem en la resta del lliurament.

En els apartats 4, 5 i 6 plantejarem un cas pràctic: a partir d'un dataset de dades de clients d'una companyia, volem predir mitjançant un model d'aprenentatge automàtic basat en regressió logística quins clients són propensos a donar-se de baixa dels serveis que ofereix la companyia. En l'apartat 4 ens centrarem en la instal·lació i configuració de l'entorn, en l'apartat 5 tractarem en detall la implementació del projecte i, per últim, en l'apartat 6 veurem com publicar el nostre projecte en GitHub.

2. Biblioteques per a IA

Existeixen multitud de biblioteques que podem utilitzar per a dotar a les nostres aplicacions de comportament intel·ligent. La majoria d'elles seran aplicables en funció del context del problema que es pretengui resoldre, i altres actuaran com a dependències d'altres per a construir biblioteques més completes.

Així, podem classificar les biblioteques d'IA segons el seu principal camp d'aplicació:

1. Aprenentatge automàtic
2. Processament del llenguatge natural
3. Xarxes neuronals
4. Visió per computador
5. Sistemes experts
6. Robòtica

En els subapartats següents veurem un llistat de les principals biblioteques de cada un d'aquests sis grups. Algunes d'aquestes biblioteques que s'hi mencionen es veuran amb molt més detall en lliuraments posteriors, així com en els mòduls de Models d'Intel·ligència Artificial i Sistemes d'Aprenentatge Automàtic.

Totes les biblioteques que es mencionaran inclouen suport per al llenguatge Python, ja sigui de manera interna pels propis desenvolupadors o de manera externa per la comunitat. Donat el ric ecosistema de biblioteques que existeix, la corba d'aprenentatge del llenguatge i el suport de plataformes d'execució, Python s'ha erigit com el llenguatge *de facto* per al desenvolupament de sistemes d'IA.

2.1. Aprenentatge automàtic

scikit-learn (**https://scikit-learn.org/**)

És una biblioteca que facilita la implementació d'algorismes d'aprenentatge automàtic. Inclou algorismes per a classificar objectes, construir regressions, agrupament d'objectes similars (*clustering*), preprocessament de dades i selecció automàtica de models. La biblioteca està basada en NumPy, SciPy i Matplotlib.

Ja hem començat a fer feina amb scikit-learn al mòdul de Sistemes d'aprenentatge automàtic.

TensorFlow (https://www.tensorflow.org/)

És un *framework* creat per Google per a facilitar l'ús d'algorismes complexos d'aprenentatge automàtic i aprenentatge profund basats en xarxes neuronals artificials (ANN). Els desenvolupadors creen fluxos de dades en els quals cada node (o «neurona») representa un càlcul concret especificat pel programador, i es tria entre un dels molts algorismes d'aprenentatge automàtic/profund ja implementats en la biblioteca TensorFlow per a executar-ho. Tot això es facilita mitjançant una API d'alt nivell per a aprenentatge profund anomenada **Keras** (https://keras.io/).

XGBoost (https://xgboost.readthedocs.io/)

XGBoost són les sigles de «eXtreme Gradient Boosting». Aquesta biblioteca se centra en ajudar els desenvolupadors a classificar dades i construir regressions utilitzant algorismes d'arbres de decisió potenciats. Aquests arbres estan formats per fills de models de regressió més febles (que representen diferents tasques de càlcul). A mesura que s'entrena el model, s'afegeixen nous models de regressió més febles per a emplenar els buits fins que no es puguin fer més millores.

2.2. Processament del llenguatge natural

NLTK (https://www.nltk.org/)

NLTK són les sigles de «Natural Language Toolkit». És una biblioteca que simplifica l'ús de la llengua gràcies a una sèrie de funcions i interfícies definides. Des de la tokenització i l'etiquetatge de text, fins a la identificació d'entitats amb nom i fins i tot la visualització d'arbres d'anàlisi sintàctica, NLTK és una biblioteca de PLN de propòsit general que encaixa en qualsevol projecte basat en l'anàlisi del llenguatge.

spaCy (https://spacy.io/)

Aquesta biblioteca fa, a través de la seva API extremadament senzilla, que el processament de grans quantitats de text sigui ràpid i eficient. En proporcionar i integrar el tokenitzador, l'etiquetador, l'analitzador sintàctic, els vectors de paraules preentrenats i les funcions de reconeixement d'entitats amb nom en una sola biblioteca, spaCy pot ajudar els programes a comprendre tots els aspectes d'un text, o simplement a preprocessar-lo perquè alguna de les altres biblioteques d'IA s'ocupi d'això posteriorment.

Gensim (https://radimrehurek.com/gensim/)

L'objectiu de Gensim és facilitar el procés d'identificació del tema subjacent d'un text (conegut com a modelatge de temes). S'encarrega de tot el procés de modelització, des del processament del text (en un diccionari de *tokens*) fins a la construcció del propi model temàtic, tot això sense haver de carregar tot el text en la memòria.

2.3. Xarxes neuronals

A més de **TensorFlow**, de la qual hem parlat en l'apartat d'aprenentatge automàtic, podem destacar aquestes biblioteques:

FANN (https://github.com/libfann/fann)

Fast Artifitial Neural Network Library (FANN), implementa xarxes neuronals artificials en C (el que fa que sigui fins a 150 vegades més ràpida que altres biblioteques), al mateix temps que les fa accessibles en diferents llenguatges, inclòs Python. És molt fàcil d'usar, ja que permet crear, entrenar i executar una xarxa neuronal artificial amb només tres crides a funcions.

PyTorch (https://pytorch.org/)

Aquesta biblioteca està construïda per a tasques de càlcul de tensors (utilitzant l'acceleració de la GPU) i la construcció de xarxes neuronals profundes més duradores, fent que les xarxes neuronals construïdes no hagin de tornar a crear-se cada vegada que canvia el cas d'ús, la qual cosa millora la velocitat i l'escalabilitat. Els seus principals casos d'ús són la substitució de NumPy per a aprofitar la potència de les GPUs (enfront de les CPUs), i com a plataforma de recerca d'aprenentatge profund altament personalitzable i ràpida.

2.4. Visió per computador

OpenCV (https://opencv.org/)

Open Source Computer Vision Library (OpenCV) proporciona als desenvolupadors més de 2.500 algorismes optimitzats per a una gran varietat de casos d'ús relacionats amb la visió per computador. Des de la detecció/reconeixement de rostres fins a la classificació d'accions humanes, OpenCV fa que la comprensió de la informació visual sigui tan simple com anomenar a la funció correcta i especificar els paràmetres adequats.

SimpleCV (http://simplecv.org/)

Mentre que OpenCV se centra en l'exhaustivitat i la personalització, SimpleCV se centra en fer que la visió per computador sigui senzilla. La corba d'aprenentatge és molt menor, fins al punt que obtenir imatges d'una càmera és tan senzill com inicialitzar una càmera (usant *Camera()*) i obtenir la seva imatge (usant *Camera.getImage()*). D'aquesta manera, el desenvolupador pot enfocar-se en el domini del problema i realitzar prototips de la solució de manera ràpida i senzilla.

2.5. Sistemes experts

PyCLIPS (http://pyclips.sourceforge.net/)

Desenvolupat al Centre Espacial Johnson de la NASA de 1985 a 1996, el C Language Integrated Production System (CLIPS) és un llenguatge de programació basat en regles útil per crear sistemes experts i altres programes on una solució heurística és més fàcil d'implementar i mantenir que una solució algorítmica. Escrit en C per a la portabilitat, CLIPS es pot instal·lar i utilitzar en una àmplia varietat de plataformes. Des de 1996, CLIPS ha estat disponible com a programari de domini públic.

PyCLIPS és un mòdul de Python per a integrar CLIPS en Python. Proporciona un motor d'inferència basat en regles com a mòduls binaris dins de la biblioteca als quals s'accedeix mitjançant classes i funcions. El motor en si mateix roman resident en un espai de memòria separat de l'espai de Python, per la qual cosa les inferències i les regles es mantenen a mesura que el programa creix en funcionalitat.

PyKnow (https://github.com/buguroo/pyknow)

També inspirat en CLIPS, aquesta biblioteca és un motor de regles que associa un conjunt de fets amb un conjunt de regles basades en aquests fets. Després, s'executen accions basades en aquestes regles. Tots els fets i les regles són mantinguts pel motor de coneixement implementat que determina la sortida del sistema expert quan és invocat.

2.6. Robòtica

AirSim (https://microsoft.github.io/AirSim/)

És un simulador basat en Unity/Unreal Engine construït per Microsoft, que permet als desenvolupadors provar i experimentar amb algorismes de vehicles autònoms sense necessitat de posseir el maquinari físic per a això. D'aquesta manera, proporciona un entorn de proves per a vehicles autònoms sense els costos i els problemes de seguretat que caldria superar en el món real.

Carla (http://carla.org/)

Mentre que AirSim pot atendre una àmplia varietat de vehicles autònoms (com a cotxes i drons), Carla es dirigeix específicament a la recerca de la conducció autònoma. Compta amb característiques més específiques per al conductor, com a sensors flexibles per al vehicle, i condicions ambientals, així com una àmplia varietat d'edificis i vehicles ja implementats.

3. Gestió de dependències i entorns virtuals

A l'hora de desenvolupar projectes, o simplement quan volem executar exemples de codi o seguir casos pràctics, resulta convenient mantenir les dependències entre biblioteques ben organitzades.

Potser per a un projecte necessitam una versió de Python concreta, amb unes versions de biblioteques específiques. Però a un altre projecte, potser necessitam altra versió de Python i de les mateixes (o altres) biblioteques .

Per gestionar correctament aquestes dependències, Python ens proporciona els anomenats entorns virtuals o *virtualenvs*. Mitjançant aquests entorns, podem aïllar les versions de les dependències entre projectes sense alterar la instal·lació global de la nostra màquina.

Al llarg dels anys han sorgit diverses eines que tracten de facilitar la gestió de dependències i l'ús d'entorns virtuals en els projectes Python. Aquí en veurem les que probablement són les més utilitzades. Començarem xerrant de **pip**, el gestor de paquets de Python. Però pip no proporciona la capacitat de crear diversos entorns aïllats en una màquina. Així que la funcionalitat de pip s'ha de completar amb **virtualenv** o amb **venv**, dos gestors d'entorns virtuals. El primer, més antic, és de tercers, mentre que el segon forma part de la biblioteca estàndard de Python des de la versió 3.3. Per últim, **Conda** i **Poetry** són dues eines completes, que suporten tant la gestió d'entorns virtuals com la instal·lació senzilla de biblioteques. Conda prové de la distribució Anaconda i serà l'eina que utilitzarem en el cas pràctic d'aquest lliurament.

Existeixen altres eines com pyenv, virtualenvwrapper, pipenv, pip-tools, que no tractarem aquí.

3.1. pip

pip és, bàsicament, el gestor de paquets per a Python.

El nom "pip" és un acrònim recursiu que vol dir "Pip Installs Packages" (Pip Instal·la Paquets).

És una eina fonamental en l'ecosistema de Python, ja que permet als desenvolupadors instal·lar, actualitzar i gestionar dependències de manera senzilla i eficient. La majoria dels paquets es poden trobar al **Python Package Index (PyPI)**, un repositori en línia amb una àmplia col·lecció de paquets de software de Python. En tot cas, també es poden emprar altres índexs.

La utilització de pip és molt simple, mitjançant la seva interfície de línia d'ordres. Per instal·lar un paquet, hem d'utilitzar la següent ordre:

```
pip install nom-del-paquet
```

La desinstal·lació d'un paquet té una ordre molt similar:

```
pip uninstall nom-del-paquet
```

Una característica important de pip és que permet gestionar llistes de paquets i les seves versions mitjançant un arxiu de requisits. Això ens permet reinstal·lar un conjunt de paquets en un entorn separat, com pot ser una altra màquina. Ho podem fer mitjançant l'ordre següent, si tenim l'arxiu requisits.txt, formatat de manera adequada (no entrarem aquí en el format):

```
pip install -r requisits.txt
```

Així doncs, pip ens ajuda no només a instal·lar fàcilment paquets, sinó també a replicar la configuració de biblioteques que tenim en una màquina a una altra. Però no arriba a ser un gestor d'entorns virtuals: no podem configurar diferents entorns per a diferents projectes en una mateixa màquina.

Podeu trobar més informació sobre pip a **https://pypi.org/project/pip/**

3.2. virtualenv

virtualenv és una eina externa, que no pertany a la biblioteca estàndard de Python, per a la creació d'entorns aïllats (o entorns virtuals) de Python. És una de les més populars, degut principalment a la seva simplicitat.

En una mateixa màquina podem tenir diversos entorns. Cada un d'ells té els seus propis directoris d'instal·lació, de manera que no comparteixen biblioteques entre ells, ni tampoc amb les biblioteques instal·lades globalment en el servidor. Així podem tenir un entorn amb una versió d'una biblioteca i un altre entorn amb una altra versió de la mateixa biblioteca.

Podem instal·lar virtualenv amb pip:

```
pip install virtualenv
```

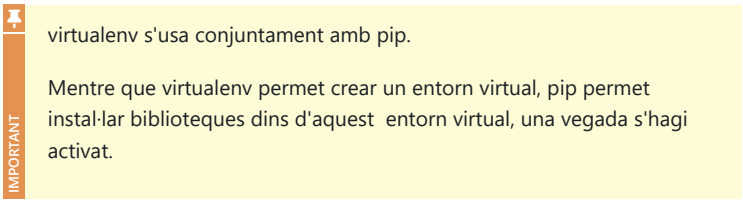
Podem crear un entorn amb l'ordre (hi ha moltes opcions, aquesta és la més simple):

```
virtualenv nom-entorn
```

Aquesta ordre crea un directori anomenat *nom-entorn* que conté els fitxers necessaris per a un entorn virtual. Abans d'instal·lar dependències en l'entorn, s'ha d'activar, cosa que es fa mitjançant una ordre específica segons el sistema operatiu. Per exemple, en Linux és:

```
source nom-entorn/bin/activate
```

Una vegada activat l'entorn, podem fer instal·lacions dins d'aquest entorn mitjançant pip. Com dèiem, les biblioteques instal·lades d'aquesta manera estaran disponibles només dins d'aquest entorn específic, garantint que les dependències del projecte no interfereixin amb altres projectes ni amb el sistema global.



Podeu trobar més informació sobre virtualenv a **https://virtualenv.pypa.io/**

3.3. venv

venv (Virtual Environment) és una eina integrada a la biblioteca estàndard de Python que proporciona una forma senzilla i lleugera de crear entorns virtuals. Com ja hem comentat, un entorn virtual és una carpeta que conté una instal·lació de Python aïllada del sistema global, permetent als desenvolupadors gestionar dependencies específiques per a cada projecte.

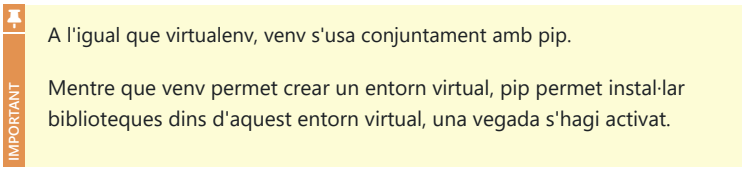
venv va aparèixer amb la versió 3.3 de Python, com a un subconjunt de virtualenv. No és tan complet com virtualenv, però el fet que sigui part de la biblioteca estàndard de Python, a més de ser simple i lleuger, ha fet que guany molt en popularitat, especialment en projectes petits.

Per crear un entorn virtual amb venv, s'utilitza la següent ordre:

```
python -m venv nom-entorn
```

El funcionament és molt semblant al de virtualenv. Això crea un directori específic per a l'entorn virtual. Hem d'activar l'entorn, de manera que podrem fer instal·lacions mitjançant pip només per a l'entorn en particular.

No obstant això, venv té algunes limitacions. En concret, no inclou funcionalitats avançades com la gestió de dependències directa o la gestió de versions de paquets. Per a projectes més complexos, és més convenient fer feina amb gestors més complexes com el propi virtualenv, o sobretot, conda o poetry.



Podeu trobar més informació sobre venv a **https://docs.python.org/3/tutorial/venv.html**

3.4. Poetry

Poetry ens proporciona un entorn integrat per a la gestió de paquets, la resolució de dependències i l'empaquetat dels nostres projectes seguint les últimes recomanacions del llenguatge, amb l'objectiu de poder compartir els nostres projectes de codi i crear entorns aïllats que siguin reproduïbles per altres usuaris.

Poetry simplifica la gestió de dependències i el desplegament de projectes, proporcionant una estructura de directoris per al projecte, on trobam el fitxer **pyproject.toml** que serveix per definir la configuració del projecte i les seves dependències.

Podem crear un nou projecte amb l'ordre:

```
poetry new nom-projecte
```

Això crea un directori (amb una estructura determinada) amb el fitxer pyproject.toml. Podem editar manualment el fitxer o emprar ordres de Poetry. Per exemple, podem afegir una nova dependència amb l'ordre:

```
poetry add biblioteca
```

Això crea un entorn virtual (si no estava ja creat) per al projecte i hi instal·la la biblioteca especificada. Aquesta instal·lació afecta només a aquest entorn virtual, no als altres que puguem tenir, ni al Python general del sistema.

D'aquesta manera podem tenir diversos entorns virtuals en la mateixa màquina. Amb l'ordre següent podem veure la llista dels entorns virtuals disponibles:

```
poetry env list
```

Podeu també trobar més informació sobre Poetry a **https://python-poetry.org/**

3.5. Conda

Conda és un gestor de paquets i d'entorns de codi obert, multiplataforma i de llenguatge agnòstic.

Conda va ser desenvolupat originalment per a resoldre els reptes difícils de gestió de paquets als quals s'enfronten els científics de dades de Python, i avui és un gestor de paquets popular per a Python i R (i també està disponible per a altres llenguatges). Conda va nèixer com a una part de la distribució de Python **Anaconda**, desenvolupada per Anaconda Inc., tot i que més tard es va separar com un paquet independent, publicat sota llicència BSD. En qualsevol cas, Conda forma part de totes les versions de les distribucions Anaconda i Miniconda.

Conda és un gestor de paquets semblant a pip. Però, a més també és un gestor d'entorns virtuals. Amb la següent ordre, podem crear un nou entorn (podem especificar moltes més opcions):

```
conda create --name nom-entorn
```

Després podem activar l'entorn mitjançant l'ordre:

```
conda activate nom-entorn
```

I a continuació podem instal·lar paquets dins l'entorn activat, mitjançant:

```
conda install nom-paquet
```

Conda gestiona paquets binaris precompilats. A diferència de pip, Conda no només es limita a paquets de Python, sinó que pot gestionar paquets d'altres llenguatges, cosa que el fa més versàtil per a projectes que involucren múltiples llenguatges.

Conda integra les dues funcionalitats que volem: gestió d'entorns i de dependències.

A més, Conda és multi-llenguatge, molt utilitzat sobretot en Python i R.

Veurem amb més detall com instal·lar i treballar amb Conda en el cas pràctic.

Podeu trobar més informació sobre Conda a **https://docs.conda.io/**

4. Cas pràctic: configuració de l'entorn

Anam a plantejar un cas pràctic en el qual s'intentarà predir mitjançant un model d'aprenentatge automàtic quins clients són propensos a deixar d'utilitzar els serveis que ofereix una determinada companyia. Donarem més detalls en l'apartat següent.

Per desenvolupar el nostre projecte, farem feina amb una eina de desenvolupament com Visual Studio Code i treballarem amb quaderns Jupyter. Com que haurem de fer feina amb diverses biblioteques, és important dur a terme una gestió de dependències adequada, per a la qual utilitzarem Conda.

En aquest apartat ens centrarem en la instal·lació i configuració de l'entorn, mentre que en l'apartat 5, tractarem en detall la implementació del projecte, i en l'apartat 6 veurem com publicar el nostre projecte en GitHub.

IMPORTANT

Al final del cas pràctic, el projecte que anirem desenvolupant quedarà publicat a GitHub.

Així que podeu trobar tot el codi que anirem escrivint durant aquests apartats a:

https://github.com/rc-iedib/cas-practic

4.1. Instal·lació de Conda

La instal·lació és un procés guiat senzill que simplement tracta d'executar l'instal·lador i seguir les instruccions. Per començar, s'ha d'anar a la pàgina oficial de **Conda** i descarregar l'instal·lador de **Miniconda** en funció del sistema operatiu. Aquest instal·lador només instal·la Conda i Python mentre que la resta de paquets s'hauran d'instal·lar manualment. Per altra part, **Anaconda** és una instal·lació més gran que ja inclou paquets populars preinstal·lats.

IMPORTANT

A Windows, en finalitzar la instal·lació de Conda, s'ha d'elegir si marcar l'opció *Add Anaconda to my PATH environment variable*. En el cas de marcar-ho, Conda serà accessible des de la terminal cmd que a la vegada està integrada a Visual Studio Code. En cas contrari, només es podrà utilitzar l'Anaconda Prompt.

Per verificar la instal·lació quan hagi acabat, s'ha d'obrir un nou terminal, ja sigui el terminal d'Anaconda Prompt a Windows o una terminal de macOS/Linux, i executar el següent comandament:

```
conda --version
```

Si la instal·lació ha estat correcta, sortirà la versió que s'ha instal·lat i la paraula (base) a l'esquerra de la línia de comandes, indicant que l'entorn base de Conda és el que es troba actiu.

Algunes altres instruccions interessants per començar són conda info per obtenir informació de l'entorn i conda update --all per assegurar-se que Conda estigui actualitzat:

```
conda info
conda update --all
```

4.2. Primer entorn virtual

Ara que ja hem instal·lat Conda, anam a crear un nou entorn per al nostre cas pràctic, amb les dependències que haurem d'utilitzar. Per fer-ho, executam l'ordre:

```
conda create --name cas-practic
```

Això ens crea un nou entorn que el podem activar amb el següent ordre:

```
conda activate cas-practic
```

La línia de comandes canviarà per mostrar el nom de l'entorn actiu. Per llistar els entorns podem fer:

```
conda env list
```

Anam a instal·lar ara les dependències que necessitarem en el nostre projecte: **scikit-learn** per a la creació d'un model d'aprenentatge automàtic, **pandas** per al tractament de les dades i **ipykernel** per a l'execució interactiva de Python en quaderns Jupyter. Per instal·lar-les, hem d'executar l'ordre següent:

```
conda install scikit-learn pandas ipykernel
```

Vegem que crea un entorn virtual amb els paquets especificats més altres paquets dels quals hi ha dependència:

```
conda list
```

```
```

Instruccions per exportar o copiar un entorn de Conda (opcional)

Convé poder exportar l'entorn de Conda per a la reproductibilitat del nostre projecte. Amb l'entorn que volem exportar en actiu, executam la comanda següent per a exportar l'entorn de Conda a un fitxer YAML (.yml). Aquest fitxer conté el nom de l'entorn, el canal utilitzat i la llista exacta de tots els paquets i les seves versions.

```
conda env export > environment.yml
```

Per replicar aquest entorn només s'haurà d'executar la següent ordre que utilitzarà automàticament les dades del fitxer environment.yml:

```
conda env create
```

En el cas que volguéssim crear una còpia amb un altre nom a partir del fitxer environment.yml, s'haurà d'executar la següent ordre:

```
conda env create -n altre_nom -f environment.yml
```

En el cas que volguéssim crear una còpia amb un altre nom a partir d'un altre entorn, s'haurà d'executar la següent ordre:

```
conda create -n altre_nom --clone cas-practic
```

Per finalitzar, si volem eliminar un entorn, primer l'haurem de desactivar per tornar a l'entorn base i, a continuació, eliminar l'entorn:

```
conda deactivate

conda remove --name altre_nom --a
```

4.3. Creació d'un script Python

Anam a treballar amb Visual Studio Code com a eina de desenvolupament. Es tracta d'un editor lliure, de codi obert, disponible per a diverses plataformes. Podem descarregar-ho des de https://code.visualstudio.com/download

A https://code.visualstudio.com/docs/setup/setup-overview trobareu les instruccions detallades d'instal·lació per a cada plataforma.

> ACLARIMENT
>
> Si ho preferiu, en lloc de Visual Studio Code, podeu emprar el vostre entorn de desenvolupament preferit. Cercau la documentació corresponent per configurar Conda.

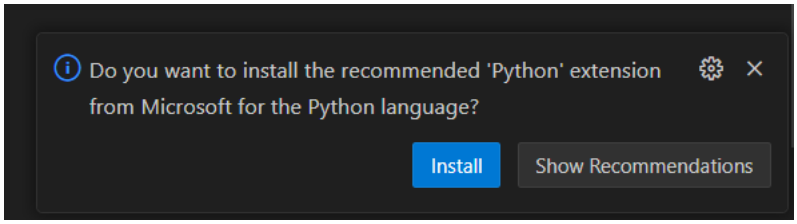
Una vegada instal·lat Visual Studio Code, crearem el nostre projecte. Per fer-ho, crearem la carpeta cas-practic a on ubicarem els nostres arxius que anirem creant. A continuació, des de la terminal amb l'entorn cas-practic actiu, ens posicionam a la carpeta creada amb cd i l'obrim a Visual Studio Code amb l'ordre:

```
code .
```

Ara anam a fer una prova: crearem un script de Python que fa servir la llibreria pandas, per comprovar que tot el nostre entorn s'ha configurat correctament. Cream el directori tests i, dins aquest directori, crearem un nou fitxer (botó *New File)*, al qual li direm **prova-pandas.py**. En aquest fitxer escriurem aquest codi:

```
import pandas as pd
dades = {
    'illa': ['Mallorca', 'Menorca', 'Eivissa', 'Formentera'],
    'superficie': [3620, 692, 577, 83],
    'poblacio': [ 923608, 94885, 147914, 11708]
}
df = pd.DataFrame(dades)
print("Superfície total:", df['superficie'].sum(), "km2")
```

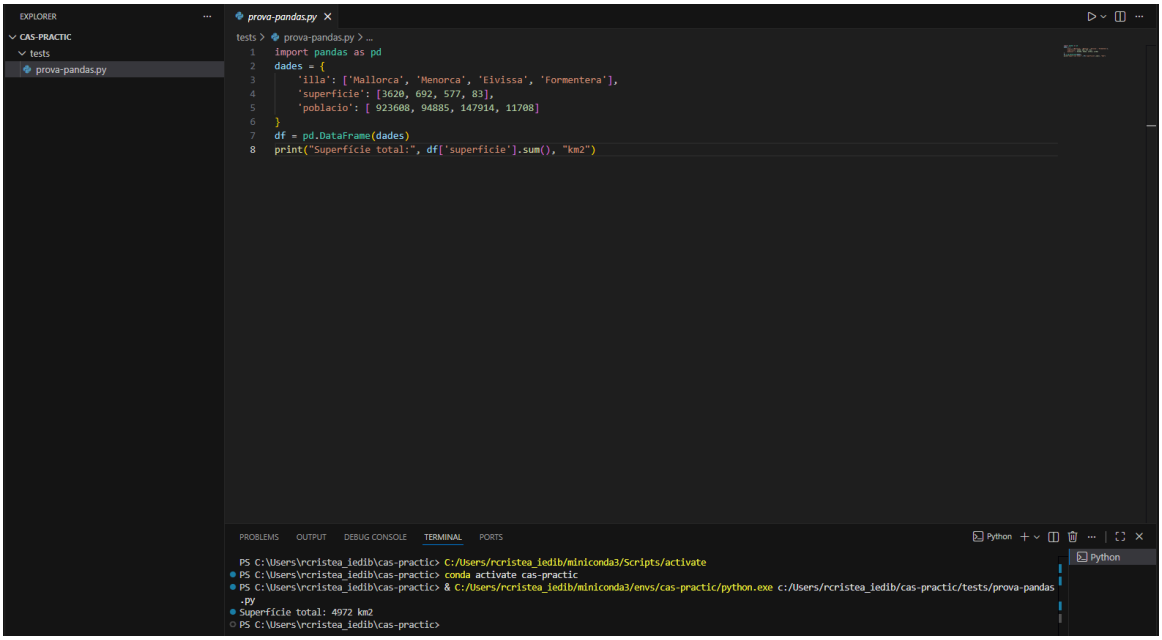
Si és la primera vegada que utilitzam Visual Studio Code, encara no tendrem l'entorn configurat per a treballar amb Python. El més probable és que aparegui una advertència com aquesta:



Imatge: *Avís per instal·lar extensió de Python per a Visual Studio Code*

Podem instal·la l'extensió de Python recomanada, la qual cosa ens permetrà que l'editor detecti la sintaxi de Python, ressalti paraules claus, recomani opcions mentre escrivim, etc.

Una vegada guardat el nostre fitxer, podem executar-lo des de Visual Studio Code (amb el botó del triangle a la part superior dreta). Quan ara executem el nostre script, es farà dins del nostre entorn virtual. Fixau-vos en el panell de TERMINAL, on podreu veure que l'ordre *python* que s'està executant és la del nostre entorn virtual.



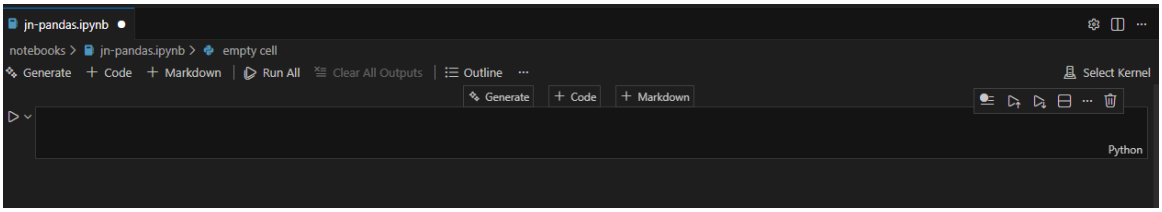
Imatge: *Projecte en Visual Studio Code.*

4.4. Configuració dels quaderns Jupyter

Anam ja a treballar amb quaderns Jupyter.

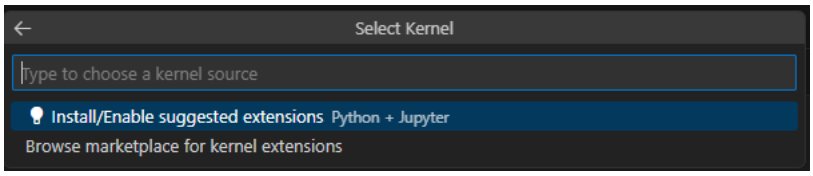
Des de Visual Studio Code, crearem una carpeta, anomenada **notebooks**, en el nostre projecte (botó *New Folder*) per a guardar-hi els quaderns de Jupyter que anirem creant en el nostre cas pràctic. I crearem ja el nostre primer quadern (botó *New File)*, al qual li direm **jn-pandas.ipynb**, per comprovar que tot l'entorn està ben configurat i permet fer feina amb quaderns Jupyter.

Una vegada creat el quadern de Jupyter haurem de crear una primera cel·la de codi (botó + *Code*) que després executarem amb el botó del triangle. Com en el cas de Google Colab, també tindrem disponible el botó per afegir noves cel·les de text amb format (Markdown):



Imatge: *Primera vista del quadern jn-pandas.ipynb.*

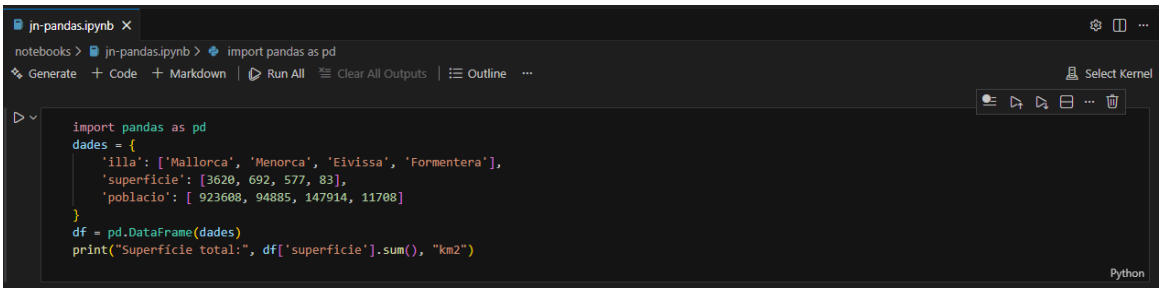
Si és la primera vegada que utilitzam Visual Studio Code, també és convenient que instal·lem l'extensió per a Jupyter. Si pitjam el botó per executar la cel·la, segurament apareixerà un missatge com aquest:



Imatge: *Avís per instal·lar extensió de Python i Jupyter per a Visual Studio Code*

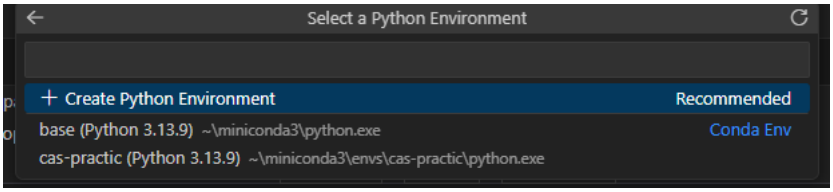
Podem instal·lar l'extensió recomanada. També ho podem gestionar des del panell d'extensions.

Ja instal·lada l'extensió, provem d'escriure el nostre codi amb pandas en la primera cel·la del quadern:



Imatge: *Primer quadern Jupyter amb pandas*

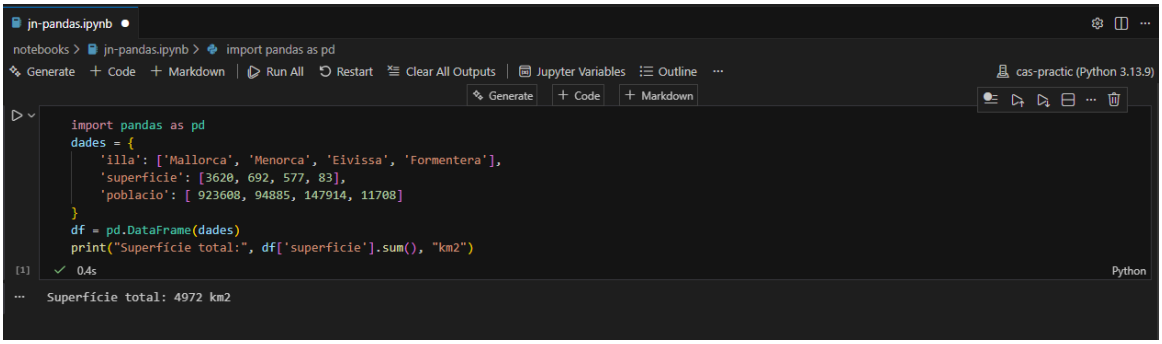
Ara, abans d'executar el quadern, hem de seleccionar l'entorn on volem que s'executi. En intentar executar la primera cel·la o pitjant a l'opció *Select Kernel* ens apareixerà un diàleg com aquest:



Imatge: *Selecció d'entorn.*

I aquí hem de seleccionar el nostre entorn virtual de Conda, el que comença per *cas-practic*.

Ara ja podem executar el nostre quadern, que s'executarà sense problemes en el nostre entorn virtual, amb les dependències que toca:



Imatge: *Execució del quadern de prova*

5. Cas pràctic: implementació

Tal i com ja havíem anunciat, en el nostre cas pràctic volem predir mitjançant un model d'aprenentatge automàtic quins clients són propensos a deixar d'utilitzar els serveis que ofereix una determinada companyia.

Saber això és crucial per a una empresa, ja que li permetria emprendre accions per a retenir al client abans que es produeixi la pèrdua definitiva.

A partir de les dades de fugides de clients antics, és possible crear un model per a identificar els clients potencials de cancel·lar la seva subscripció amb els serveis que ofereix la companyia. Per tant, es tracta d'un problema de classificació binària, on s'intenta predir si un determinat client es donarà de baixa o no.

Per a il·lustrar l'exemple s'utilitzarà una anàlisi de les dades mitjançant un model estadístic de **regressió logística**, per ser un dels més simples i ràpids d'utilitzar però que al mateix temps ens permetrà realitzar prediccions sobre variables categòriques (aquelles que poden adoptar un nombre finit de categories).

5.1. Preparació de les dades

Per a aquest cas pràctic anam a partir d'un dataset que conté tota la informació que necessitam. Concretament, utilitzarem el dataset de *Telco Customer Churn*, disponible en el lloc web de Kaggle: https://www.kaggle.com/blastchar/telco-customer-churn. El dataset conté les dades dels clients d'una empresa que ofereix serveis de telecomunicacions.

D'acord amb la seva descripció, el dataset inclou informació de:

- Els clients que s'han donat de baixa en l'últim mes (columna *Churn*).
- Serveis contractats per cada client: telèfon, línies múltiples, Internet, seguretat en línia, còpies de seguretat en línia, protecció de dispositius, assistència tècnica, TV i pel·lícules.
- Informació del compte del client: quant temps ha estat client, contracte, mètode de pagament, facturació digital, càrrecs mensuals i càrrecs totals.
- Informació demogràfica sobre els clients: sexe, rang d'edat, i si tenen parella i persones al seu càrrec.

Una vegada descarregat (us haureu de registrar a Kaggle), ho descomprimirem i emmagatzemarem l'arxiu CSV contingut en un nou directori ***datasets*** en l'arrel del nostre projecte.

Anam ara a realitzar una anàlisi inicial de les dades. Per a això, començarem creant un nou quadern Jupyter anomenat CasPractic.ipynb. Una vegada creat, importarem la biblioteca de pandas:

```
import pandas as pd
```

Carregarem el dataset que acabem de descarregar i veurem quantes files té i la capçalera:

```
df = pd.read_csv('datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv')
print(len(df))
df.head().T
```

Podem veure que el dataset conté 7043 files i diverses columnes amb dades:

```
[1]  import pandas as pd
     ✓ 0.2s

[2]  df = pd.read_csv('../datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv')
     print(len(df))
     df.head().T
     ✓ 0.0s

...  7043
```

	0	1	2	3	4
customerID	7590-VHVEG	5575-GNVDE	3668-QPYBK	7795-CFOCW	9237-HQITU
gender	Female	Male	Male	Male	Female
SeniorCitizen	0	0	0	0	0
Partner	Yes	No	No	No	No
Dependents	No	No	No	No	No
tenure	1	34	2	45	2
PhoneService	No	Yes	Yes	No	Yes
MultipleLines	No phone service	No	No	No phone service	No
InternetService	DSL	DSL	DSL	DSL	Fiber optic
OnlineSecurity	No	Yes	Yes	Yes	No
OnlineBackup	Yes	No	Yes	No	No
DeviceProtection	No	Yes	No	Yes	No
TechSupport	No	No	No	Yes	No
StreamingTV	No	No	No	No	No
StreamingMovies	No	No	No	No	No
Contract	Month-to-month	One year	Month-to-month	One year	Month-to-month
PaperlessBilling	Yes	No	Yes	No	Yes
PaymentMethod	Electronic check	Mailed check	Mailed check	Bank transfer (automatic)	Electronic check
MonthlyCharges	29.85	56.95	53.85	42.3	70.7
TotalCharges	29.85	1889.5	108.15	1840.75	151.65
Churn	No	No	Yes	No	Yes

Imatge: *Vista general del dataset*

Aquesta és la descripció de cada columna:

- CustomerID: l'identificador del client.
- Gender: masculí o femení (*male/female*).
- SeniorCitizen: si el client és una persona major (*0/1*).
- Partner: si el client viu en parella (*yes/no*).
- Dependents: si el client té dependents (*yes/no*).
- Tenure: nombre de mesos des que es va iniciar el contracte (*numèric*).
- PhoneService: si el client té línia de telèfon (*yes/no*).
- MultipleLines: si el client té diverses línies telefòniques (*yes/no/no phone service*).
- InternetService: el tipus de servei d'internet contractada (*no/fiber/optic*).
- OnlineSecurity: si la seguretat està activada (*yes/no/no internet*).
- OnlineBackup: si el servei de còpies de seguretat en línia està activat (*yes/no/no internet*).
- DeviceProtection: si el servei de protecció de dispositius està activat (*yes/no/no internet*).
- TechSupport: si el client té contractat el *servcio de suport tècnic (*yes/no/no internet*).
- StreamingTV: si el servei de TV està activat (*yes/no/no internet*).
- StreamingMovies: si el servei de pel·lícules està activat (*yes/no/no internet*).
- Contract: el tipus de contracte (*monthly/yearly/two years*).
- PaperlessBilling: si la facturació és digital (*yes/no)*.
- PaymentMethod: la forma de pagament (*electronic check/mailed check/bank transfer/credit card*).
- MonthlyCharges: l'import mensual que es cobra (*numèric*).
- TotalCharges: l'import total cobrat des de que es va donar d'alta (*numèric*).
- Churn: si el client s'ha donat de baixa (*yes/no*).

Entre les propietats anteriors, la més rellevant per al nostre cas pràctic és ***Churn***, la qual establirem com a **variable objectiu**, és a dir, aquella sobre la qual el nostre model realitzarà les prediccions.

Quan pandas importa el dataset, intenta determinar automàticament el tipus de dada per a cada columna. Podem veure aquesta informació inspeccionant l'atribut *dtypes:*

```
[3]  df.dtypes
     ✓ 0.0s

...  customerID          object
     gender              object
     SeniorCitizen        int64
     Partner             object
     Dependents          object
     tenure               int64
     PhoneService        object
     MultipleLines       object
     InternetService     object
     OnlineSecurity      object
     OnlineBackup        object
     DeviceProtection    object
     TechSupport         object
     StreamingTV         object
     StreamingMovies     object
     Contract            object
     PaperlessBilling    object
     PaymentMethod       object
     MonthlyCharges     float64
     TotalCharges        object
     Churn               object
     dtype: object
```

Imatge: *Tipus de dades de les columnes*

Podem veure com gairebé tots els tipus de dades s'han identificat correctament (el tipus *object* equival a una mena de cadena de text), encara que podem veure unes certes particularitats en dues columnes:

- SeniorCitizen: aquesta columna s'ha identificat com a tipus numèric pel fet que pot prendre valors de 0 o 1. Això no afecta realment a la resta del cas pràctic, per la qual cosa podem ignorar-lo.
- TotalCharges: aquesta columna s'ha identificat com una cadena de text pel fet que algunes files contenen un espai en blanc per a representar un valor que falta.

Podem canviar el tipus de la columna TotalCharges mitjançant la funció *to_numeric* de Pandes i especificar que aquells valors que faltin siguin substituïts pel valor NaN, que en Python representa un valor de tipus numèric:

```
total_charges = pd.to_numeric(df.TotalCharges, errors='coerce')
df.TotalCharges = pd.to_numeric(df.TotalCharges, errors='coerce')
df.TotalCharges = df.TotalCharges.fillna(0)
df[total_charges.isnull()][['customerID', 'TotalCharges']]
```

```
total_charges = pd.to_numeric(df.TotalCharges, errors='coerce')
df.TotalCharges = pd.to_numeric(df.TotalCharges, errors='coerce')
df.TotalCharges = df.TotalCharges.fillna(0)
df[total_charges.isnull()][['customerID', 'TotalCharges']]
```

[4] ✓ 0.0s

	customerID	TotalCharges
488	4472-LVYGI	0.0
753	3115-CZMZD	0.0
936	5709-LVOEQ	0.0
1082	4367-NUYAO	0.0
1340	1371-DWPAZ	0.0
3331	7644-OMVMY	0.0
3826	3213-VVOLG	0.0
4380	2520-SGTTA	0.0
5218	2923-ARZLG	0.0
6670	4075-WKNIU	0.0
6754	2775-SEFEE	0.0

Imatge: *Canvi de tipus per a TotalCharges*

També es pot observar que els noms d'algunes columnes no són consistents amb la resta. Per exemple, alguns valors comencen per lletra minúscula i altres per majúscula. També podem veure que hi ha espais en els valors de les columnes. Així, canviarem de nom els noms de les columnes per a canviar-los per minúscules i substituirem els espais en els valors de les columnes per guions baixos:

```
replacer = lambda str: str.lower().str.replace(' ', '_')
df.columns = replacer(df.columns.str)
for col in list(df.dtypes[df.dtypes == 'object'].index):
    df[col] = replacer(df[col].str)
df.head().T
```

```
replacer = lambda str: str.lower().str.replace(' ', '_')
df.columns = replacer(df.columns.str)
for col in list(df.dtypes[df.dtypes == 'object'].index):
    df[col] = replacer(df[col].str)
df.head().T
```

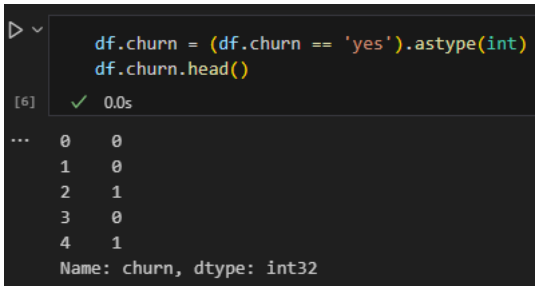
[5] ✓ 0.0s

	0	1	2	3	4
customerid	7590-vhveg	5575-gnvde	3668-qpybk	7795-cfocw	9237-hqitu
gender	female	male	male	male	female
seniorcitizen	0	0	0	0	0
partner	yes	no	no	no	no
dependents	no	no	no	no	no
tenure	1	34	2	45	2
phoneservice	no	yes	yes	no	yes
multiplelines	no_phone_service	no	no	no_phone_service	no
internetservice	dsl	dsl	dsl	dsl	fiber_optic
onlinesecurity	no	yes	yes	yes	no
onlinebackup	yes	no	yes	no	no
deviceprotection	no	yes	no	yes	no
techsupport	no	no	no	yes	no
streamingtv	no	no	no	no	no
streamingmovies	no	no	no	no	no
contract	month-to-month	one_year	month-to-month	one_year	month-to-month
paperlessbilling	yes	no	yes	no	yes
paymentmethod	electronic_check	mailed_check	mailed_check	bank_transfer_(automatic)	electronic_check
monthlycharges	29.85	56.95	53.85	42.3	70.7
totalcharges	29.85	1889.5	108.15	1840.75	151.65
churn	no	no	yes	no	yes

Imatge: *Canvi de noms en les columnes*

Finalment, en els problemes de classificació binària, els models solen esperar els valors de la variable objectius com a valors numèrics, per la qual cosa modificarem els valors de columna *churn* per a substituir les cadenes de text "yes" per un 1 i les cadenes de text "no" per un 0:

```
df.churn = (df.churn == 'yes').astype(int)
df.churn.head()
```



Imatge: *Canvi de tipus en la columna churn*

Una vegada tinguem el dataset preparat, podem procedir a dividir-ho en dades d'entrenament i dades de prova. Per a això, utilitzarem la funció ***train_test_split*** de scikit-learn per a mantenir un 80% de les dades com a dades d'entrenament i el restant 20% com a dades de prova. Aquesta funció barrejarà les dades del dataset aleatòriament i després les dividirà en els dos conjunts. Perquè aquesta mescla es realitzi de la mateixa manera en invocacions successives, proporcionarem un valor constant al tercer paràmetre de la funció (*random_state*). Repetirem el procés de nou sobre les dades d'entrenament per a reservar un 33% de les dades d'entrenament per a la seva validació. Finalment, eliminarem la columna *churn* de les dades d'entrenament per a assegurar-nos que aquesta columna no s'utilitza accidentalment durant l'entrenament del model, no sense abans fer una còpia per a utilitzar-la posteriorment.

```
from sklearn.model_selection import train_test_split
df_train_full, df_test = train_test_split(df, test_size=0.2, random_state=1)

df_train, df_val = train_test_split(df_train_full, test_size=0.33, random_state=1)
y_train = df_train.churn.values
y_val = df_val.churn.values

del df_train['churn']
del df_val['churn']

df_train.head().T
```

```python
from sklearn.model_selection import train_test_split
df_train_full, df_test = train_test_split(df, test_size=0.2, random_state=1)

df_train, df_val = train_test_split(df_train_full, test_size=0.33, random_state=1)
y_train = df_train.churn.values
y_val = df_val.churn.values

del df_train['churn']
del df_val['churn']

df_train.head().T
```

[8] ✓ 0.0s

	4204	7034	5146	5184	1310
customerid	4395-pzmsn	0639-tsiqw	3797-fkogq	7570-welny	6393-wryze
gender	male	female	male	female	female
seniorcitizen	1	0	0	0	0
partner	no	no	no	yes	yes
dependents	no	no	yes	no	no
tenure	5	67	11	68	34
phoneservice	yes	yes	yes	yes	yes
multiplelines	no	yes	yes	yes	yes
internetservice	fiber_optic	fiber_optic	fiber_optic	fiber_optic	fiber_optic
onlinesecurity	no	yes	no	yes	no
onlinebackup	yes	yes	no	yes	no
deviceprotection	no	yes	no	no	no
techsupport	no	no	no	no	no
streamingtv	no	yes	no	no	yes
streamingmovies	yes	no	yes	no	yes
contract	month-to-month	month-to-month	month-to-month	two_year	month-to-month
paperlessbilling	yes	yes	no	yes	yes
paymentmethod	electronic_check	credit_card_(automatic)	electronic_check	bank_transfer_(automatic)	electronic_check
monthlycharges	85.55	102.95	86.2	84.7	97.65
totalcharges	408.5	6886.25	893.2	5711.05	3207.55

Imatge: *Separació de les dades per a entrenaments i proves*

5.2. Anàlisi d'importància de les propietats

Abans de passar al procés d'entrenament, hem d'identificar **quines variables tenen un major impacte sobre la variable objectiu** que pretenem predir.

En el cas de les variables categòriques, podem estudiar la seva importància calculant el **grau de dependència** entre ella i la variable objectiu. Si dues variables són dependents, conèixer el valor d'una d'elles ens donarà una certa informació sobre l'altra. D'altra banda, si una variable és completament independent de la variable objectiu, no ens serà útil, per la qual cosa podrem eliminar-la amb seguretat del conjunt de dades.

Variables categòriques

Per a les variables categòriques, una d'aquestes mètriques és la **informació mútua**, que ens indica quanta informació obtenim sobre una variable si coneixem el valor d'una altra. Aquesta mètrica s'utilitza sovint en l'aprenentatge automàtic per a mesurar la dependència mútua entre dues variables: a major valor d'informació mútua, major serà la dependència entre totes dues variables (i per tant aquesta variable serà rellevant per a predir l'objectiu).

Utilitzant la funció ***mutual_info_score*** de scikit-learn, podem calcular el valor d'informació mútua entre la variable objectiu (*churn*) i cadascuna de les nostres variables categòriques. Primer definim dues llistes de quines són les variables (columnes) categòriques i quines numèriques:

```
categorical = ['gender', 'seniorcitizen', 'partner', 'dependents',
 'phoneservice', 'multiplelines', 'internetservice', 'onlinesecurity',
 'onlinebackup', 'deviceprotection', 'techsupport', 'streamingtv',
 'streamingmovies', 'contract', 'paperlessbilling', 'paymentmethod']
numerical = ['tenure', 'monthlycharges', 'totalcharges']
```

I després calculam els valors d'informació mútua per a les variables categòriques:

```
from sklearn.metrics import mutual_info_score

calculate_mi = lambda col: mutual_info_score(col, df_train_full.churn)

df_mi = df_train_full[categorical].apply(calculate_mi)
df_mi = df_mi.sort_values(ascending=False).to_frame(name='MI')
df_mi
```

```
from sklearn.metrics import mutual_info_score

calculate_mi = lambda col: mutual_info_score(col, df_train_full.churn)

df_mi = df_train_full[categorical].apply(calculate_mi)
df_mi = df_mi.sort_values(ascending=False).to_frame(name='MI')
df_mi
```

✓ 0.0s

	MI
contract	0.098320
onlinesecurity	0.063085
techsupport	0.061032
internetservice	0.055868
onlinebackup	0.046923
deviceprotection	0.043453
paymentmethod	0.043210
streamingtv	0.031853
streamingmovies	0.031581
paperlessbilling	0.017589
dependents	0.012346
partner	0.009968
seniorcitizen	0.009410
multiplelines	0.000857
phoneservice	0.000229
gender	0.000117

Imatge: *Càlcul dels valors d'informació mútua*

D'aquesta anàlisi, podem veure que les variables *contract*, *onlinesecurity* i *techsupport* es trobarien entre les propietats més importants.

Variables numèriques

Ens queda per quantificar el grau de dependència de les tres variables numèriques, per la qual cosa hem d'aplicar alguna altra tècnica per a això. Un mètode estadístic que podem aplicar és el **coeficient de correlació de Pearson** entre dues variables numèriques. En el nostre cas, podem aplicar-lo assumint que els valors de la variable objectiu s'han convertit a valors numèrics (0 i 1). Com ja hem vist en el mòdul de Sistemes de big data, el coeficient pot prendre valors entre -1 i 1:

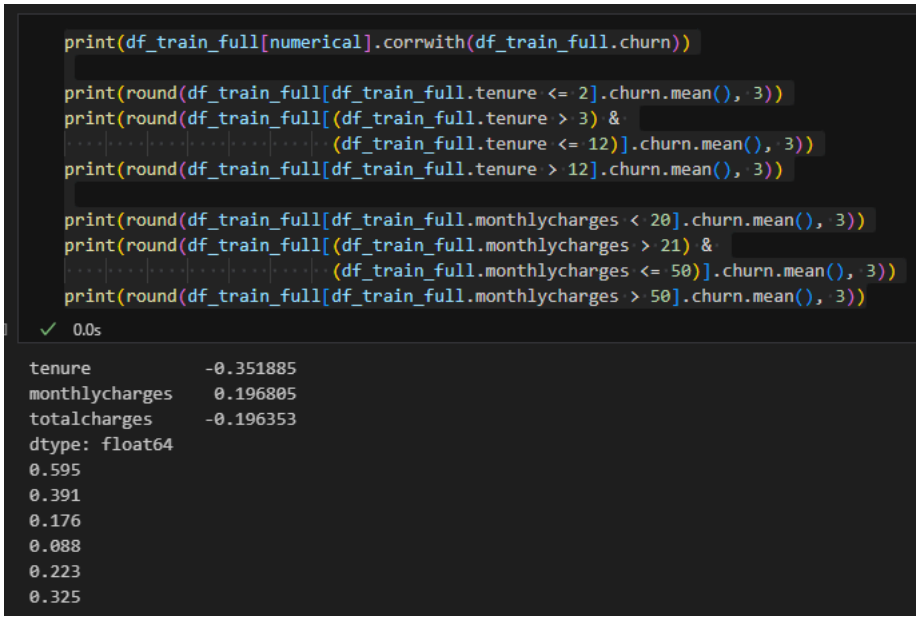
- Una correlació positiva implica que quan el valor d'una variable augmenta, també ho fa el de l'altra variable.
- Una correlació de zero indica que no hi ha relació entre les dues variables.
- Una correlació negativa implica que quan el valor d'una variable augmenta, el valor de l'altra variable disminueix.

Així, podem aplicar el coeficient de correlació a cadascuna de les nostres tres variables numèriques per a estudiar la correlació de cadascuna amb la nostra variable objectiu mitjançant la funció ***corrwith*** de pandas. Per confirmar aquesta anàlisi, veurem també com canvia la mitjana de les baixes (*churn*) en diferents intervals de les variables *tenure* i *monthlyCharges*.

```
print(df_train_full[numerical].corrwith(df_train_full.churn))

print(round(df_train_full[df_train_full.tenure <= 2].churn.mean(), 3))
print(round(df_train_full[(df_train_full.tenure > 3) &
                          (df_train_full.tenure <= 12)].churn.mean(), 3))
print(round(df_train_full[df_train_full.tenure > 12].churn.mean(), 3))

print(round(df_train_full[df_train_full.monthlycharges < 20].churn.mean(), 3))
print(round(df_train_full[(df_train_full.monthlycharges > 21) &
                          (df_train_full.monthlycharges <= 50)].churn.mean(), 3))
print(round(df_train_full[df_train_full.monthlycharges > 50].churn.mean(), 3))
```



Imatge: *Anàlisi de correlacions*

La correlació més forta és amb la variable *tenure* (en aquest cas, una correlació negativa). Podem deduir que com més temps du un client amb l'empresa, menor és la probabilitat que es doni de baixa dels serveis. En canvi, com més paga al mes un client pels serveis contractats, major és la probabilitat que es doni de baixa.

5.3. Enginyeria de propietats

El darrer pas abans de procedir a entrenar el nostre model serà el de l'enginyeria de propietats (o *feature engineering*, en anglès). Els models d'aprenentatge automàtic treballen amb matrius numèriques, per la qual cosa haurem de convertir totes les variables categòriques a variables numèriques que puguem codificar en forma de matriu de dades.

Per a això, podem simplement aplicar la tècnica de codificació d'etiquetes i donar-li un valor numèric a cada cadena de text. No obstant això, aquest enfocament podria presentar el problema que els valors numèrics siguin malinterpretats per alguns algorismes. Per això, sorgeix la tècnica de **codificació *one-hot***, que consisteix en la creació d'una columna per a cada valor únic que existeixi en la propietat que estam codificant i, per a cada registre, marcam amb un 1 la columna a la qual pertanyi aquest registre i deixam a 0 les altres.

La biblioteca de scikit-learn ens proporciona diverses maneres de realitzar aquesta codificació, essent una d'elles ***DictVectorizer***. Per a utilitzar-la, primer haurem de convertir el nostre dataset a una llista de diccionaris *columna-valor* mitjançant la funció *to_dict(orient='records')* de pandas. Vegem com fer la conversió i com queda el primer registre del dataset d'entrenament:

```
train_dict = df_train[categorical + numerical].to_dict(orient='records')
train_dict[0]
```

Imatge: *Conversió a una llista de diccionaris*

Una vegada convertit, podem passar a utilitzar *DictVectorizer* per a realitzar la codificació de les propietats. Per a això, crearem una instància d'aquesta classe i la inicialitzarem amb les dades d'entrenament perquè infereixi els valors per a cada propietat; si la propietat és categòrica, aplica l'estratègia de codificació *one-hot* i si és numèrica, la deixarà intacta. Després, podrem utilitzar la funció ***transform*** per a convertir la llista de diccionaris a una matriu.

Si explorem el resultat de l'operació, veurem com s'ha creat una llista de 45 columnes amb les possibles combinacions dels valors de les propietats categòriques per a cada fila del nostre dataset.

```
from sklearn.feature_extraction import DictVectorizer

dv = DictVectorizer(sparse=False)
dv.fit(train_dict)
```

```
X_train = dv.transform(train_dict)
X_train[0]
```

```
dv.get_feature_names_out()
```

Imatge: *Codificación one-hot amb DictVectorizer*

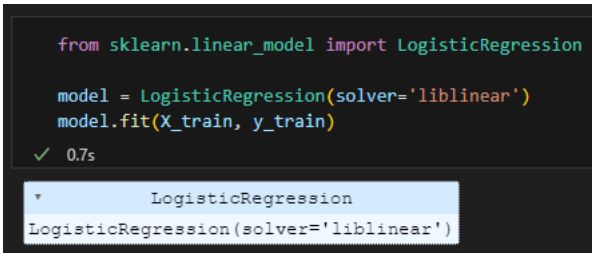
Una vegada tenim les nostres propietats codificades en forma de matriu de dades, ja podem procedir a realitzar l'entrenament del model.

5.4. Entrenament del model

Per a entrenar el model utilitzarem una **regressió logística**. Per a això, la biblioteca scikit-learn ens proporciona la classe ***LogisticRegression***, amb el mètode ***fit*** que podem utilitzar per a realitzar l'entrenament amb les dades a partir de la matriu de les dades d'entrenament (*X_train*) i els valors de la variable objectiu d'aquest conjunt de dades (*y_train*):

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)
```



Imatge: *Entrenament del model*

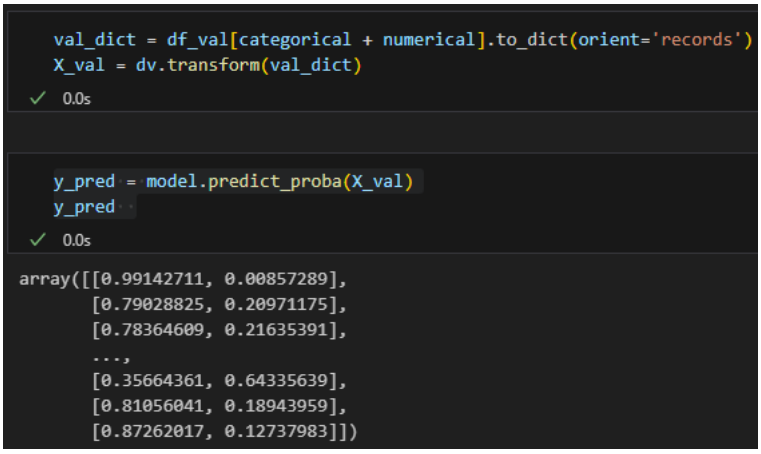
5.5. Prediccions

Una vegada entrenat el model, ja estaria llest per a realitzar prediccions sobre noves dades utilitzant el mètode ***predict_proba***. Per fer-ho, utilitzarem el conjunt de dades de validació que havíem reservat al començament del cas pràctic (*df_val*). Abans, però, hem de convertir *df_val*, primer a una llista de diccionaris (amb *to_dict(orient='records')*) i després transformar la llista en una matriu (amb *transform()*). El resultat és la matriu *X_val*.

```
val_dict = df_val[categorical + numerical].to_dict(orient='records')
X_val = dv.transform(val_dict)
```

Ara ja podem obtenir les prediccions mitjançant el mètode *predict_proba* del model:

```
y_pred = model.predict_proba(X_val)
y_pred
```

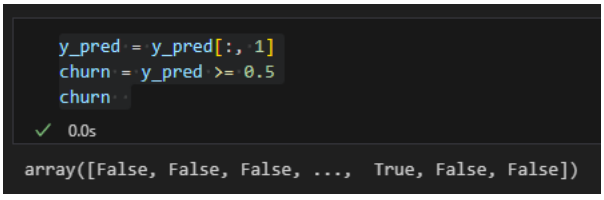


Imatge: *Obtenció de les prediccions*

El resultat d'executar el mètode *predict_proba* és una matriu bidimensional on la primera columna contindrà la probabilitat que el client no es doni de de baixa (**cas negatiu**) i la segona columna contindrà la probabilitat que el client sí que es doni de baixa (**cas positiu**). Com únicament ens interessa el cas positiu, podem simplificar l'estructura eliminant la primera columna de la matriu i discretitzar aquests valors numèrics de probabilitat a valors booleans: si el client es donarà de baixa, serà *true* i si no *false*.

Aquesta discretització de les probabilitats en valors booleans es realitza establint un punt de tall que utilitzarem per a fixar que aquells valors superiors al punt de tall seran "vertaders" i aquells inferiors seran "falsos". En el nostre exemple, hem establert que el punt de tall per a assumir que un client es donarà de baixa es trobarà a partir d'una probabilitat del 50% (0.5).

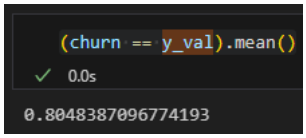
```
y_pred = y_pred[:, 1]
churn = y_pred >= 0.5
churn
```



Imatge: *Discretització de les prediccions*

Una vegada obtingudes les prediccions sobre les dades de validació, ens queda analitzar com de precís és el nostre model. Compararem les prediccions obtingudes (*churn*) amb els valors reals de la variable objectiu que havíem reservat al principi (*y_val*):

```
(churn == y_val).mean()
```



Imatge: *Comprovació de les prediccions amb els valors reals*

Podem veure que un 80,48% de les prediccions han estat correctes.

5.6. Serialització del model

Una vegada entrenat el nostre model d'aprenentatge automàtic, hauríem de ser capaços de poder utilitzar-lo per a realitzar prediccions sota demanda, ja sigui mitjançant algun servei web allotjat en un servidor d'Internet o directament mitjançant algun programa que s'executi en la nostra computadora localment.

En qualsevol cas, el model que hem entrenat fins ara resideix únicament en l'entorn virtual de Python que hem preparat. Una vegada desactivem l'entorn virtual (per exemple, quan tanquem Visual Studio Code), el model es perdrà i haurem de tornar a entrenar-lo de nou en cas que vulguem realitzar prediccions.

Per a evitar això, necessitam alguna forma de, en primer lloc, persistir el model entre diferents execucions de l'entorn virtual i, a més, poder integrar els models entrenats en algun altre sistema. Per a això, Python ens proporciona el mòdul **Pickle**, que ens permet serialitzar objectes en format binari i carregar-los més tard. En el nostre cas, a més del model, haurem d'emmagatzemar també la instància del *DictVectorizer* que vàrem inicialitzar amb el dataset, per la qual cosa podem guardar tots dos objectes com una tupla.

> ACLARIMENT
>
> Pickle és un mòdul de Python, així que no és necessari afegir noves dependències al projecte.

Ara serialitzarem el nostre model en un fitxer anomenat *churn-model.pck*, dins el directori *models*, situat en l'arrel del projecte.

```
import pickle

with open('../models/churn-model.pck', 'wb') as f:
    pickle.dump((dv, model), f)
```

Una vegada emmagatzemat el nostre model en disc, podem recuperar-lo de nou des del disc i realitzar prediccions sobre ell:

```
with open('../models/churn-model.pck', 'rb') as f:
    dv, model = pickle.load(f)
    X_val = dv.transform(val_dict)
    y_pred = model.predict_proba(X_val)

y_pred
```

D'aquesta manera, podríem construir una nova aplicació, o un altre quadern, que importi i sigui capaç de realitzar prediccions sobre el model a partir de les seves dades.

5.7. Desplegament del model

Ara que ja hem aconseguit fer el nostre model persistent, crearem un petit servidor, emprant el *framework* **Flask**, per a construir un servei que ens permeti realitzar prediccions mitjançant peticions web.

Abans de res, afegirem la dependència al projecte (des de la consola):

```
conda install anaconda::flask
```

Una vegada instal·lada, ens dirigirem al directori *cas_practic* i crearem un nou script de Python anomenat ***churn_predict_service.py*** amb una funció que ens permetrà realitzar prediccions sobre el model:

```
def predict_single(customer, dv, model):
    x = dv.transform([customer])
    y_pred = model.predict_proba(x)[:, 1]
    return (y_pred[0] >= 0.5, y_pred[0])
```

Aquest mètode ens retornarà una tupla. El primer element de la tupla indicarà si el client especificat es donarà de baixa dels serveis, mentre que el segon element de la tupla ens dona la probabilitat de fer-ho.

A continuació, crearem un script anomenat ***churn_predict_app.py***, també en el directori *cas_practic* del projecte. Aquest script inicialitzarà el servidor de Flask i exposarà un *endpoint* que podrem consumir per a realitzar les prediccions sobre el model utilitzant el servei anterior:

```
import pickle
from flask import Flask, jsonify, request
from churn_predict_service import predict_single

app = Flask('churn-predict')

with open('models/churn-model.pck', 'rb') as f:
    dv, model = pickle.load(f)


@app.route('/predict', methods=['POST'])
def predict():
    customer = request.get_json()
    churn, prediction = predict_single(customer, dv, model)

    result = {
        'churn': bool(churn),
        'churn_probability': float(prediction),
    }

    return jsonify(result)


if __name__ == '__main__':
    app.run(debug=True, port=8000)
```

Les línies

```
with open('../models/churn-model.pck', 'rb') as f:
    dv, model = pickle.load(f)
```

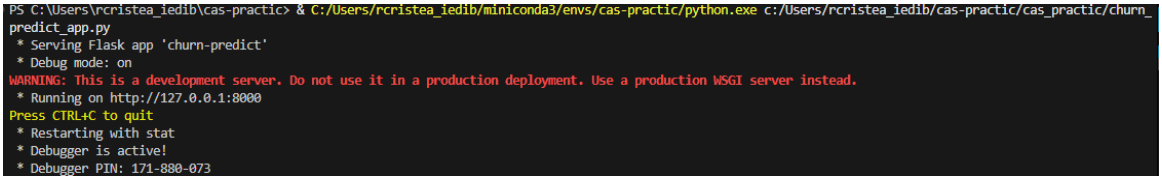
es fan servir per a carregar el model des de disc i passar-li com a paràmetre al servei de prediccions perquè l'utilitzi.

La resta del codi inicialitza una ruta en */predict* que respon a peticions HTTP de tipus POST, amb la informació del client en format JSON. Per últim, s'inicialitza el servidor Flask en el port 8000 de la màquina local.

Per executar l'aplicació ho farem mitjançant el nostre script *churn_predict_app.py* des de Visual Studio Code (amb el botó del triangle a la part superior dreta).

```

```



Imatge: *Execució de l'aplicació servidor*

Com veim, quan executam l'aplicació es posa en marxa el servidor Flask en **http://127.0.0.1:8000**.

Ara podem posar a prova la nostra aplicació, enviant una petició al servei amb les dades d'algun client. Ho farem mitjançant un nou quadern Jupyter (*client.ipynb* dins el directori *notebooks* del projecte), on fent servir l'ordre **curl**, enviam per POST el JSON corresponent a un possible client:

```
!curl --request POST "http://127.0.0.1:8000/predict" \
--header "Content-Type: application/json" \
--data-raw "{\
    \"gender\": \"female\",\
    \"seniorcitizen\": 0,\
    \"partner\": \"no\",\
    \"dependents\": \"no\",\
    \"tenure\": 41,\
    \"phoneservice\": \"yes\",\
    \"multiplelines\": \"no\",\
    \"internetservice\": \"dsl\",\
    \"onlinesecurity\": \"yes\",\
    \"onlinebackup\": \"no\",\
    \"deviceprotection\": \"yes\",\
    \"techsupport\": \"yes\",\
    \"streamingtv\": \"yes\",\
    \"streamingmovies\": \"yes\",\
    \"contract\": \"one_year\",\
    \"paperlessbilling\": \"yes\",\
    \"paymentmethod\": \"bank_transfer_(automatic)\",\
    \"monthlycharges\": 79.85,\
    \"totalcharges\": 3320.75\
}"
```

Com que les dades que enviam per POST en el paràmetre *data-raw* van entre cometes dobles, hem d'utilitzar el caràcter d'escapament \" per a les cometes dobles del document JSON.

El servidor ens respon aquest JSON:

```
{
 "churn": false,
 "churn_probability": 0.057754360975975874
}
```

Això ens indica que el model prediu que el client no es donarà de baixa, ja que la probabilitat de que ho faci és només d'un 5,78%.

Vegem una altra petició:

```
!curl --request POST "http://127.0.0.1:8000/predict" \
--header "Content-Type: application/json" \
--data-raw "{\
    \"gender\": \"female\",\
    \"seniorcitizen\": 1,\
    \"partner\": \"no\",\
    \"dependents\": \"no\",\
    \"phoneservice\": \"yes\",\
    \"multiplelines\": \"yes\",\
    \"internetservice\": \"fiber_optic\",\
    \"onlinesecurity\": \"no\",\
    \"onlinebackup\": \"no\",\
    \"deviceprotection\": \"no\",\
    \"techsupport\": \"no\",\
    \"streamingtv\": \"yes\",\
    \"streamingmovies\": \"no\",\
    \"contract\": \"month-to-month\",\
    \"paperlessbilling\": \"yes\",\
    \"paymentmethod\": \"electronic_check\",\
    \"tenure\": 1,\
    \"monthlycharges\": 85.7,\
    \"totalcharges\": 85.7\
}"
```

Que ens retorna el JSON següent:

```
{
  "churn": true,
  "churn_probability": 0.7930641120090199
}
```

En aquest cas, el model sí que prediu que el client es donarà de baixa, amb una probabilitat del 79,3%

Aquí hem cridat el servei web des d'un quadern Jupyter. Perquè quedi clar que es pot fer des de fora del nostre entorn virtual de Conda, podem obrir una consola i executar l'ordre *curl* directament (tot en una línia):

curl --request POST "http://127.0.0.1:8000/predict" --header "Content-Type: application/json" --data-raw "{\"gender\": \"female\",\"seniorcitizen\": 0,\"partner\": \"r

```
C:\>curl --request POST "http://127.0.0.1:8000/predict" --header "Content-Type: application/json" --data-raw "{\"gender\": \"female\",\"seniorcitizen\": 0,\
"partner\": \"no\",\"dependents\": \"no\",\"tenure\": 41,\"phoneservice\": \"yes\",\"multiplelines\": \"no\",\"internetservice\": \"dsl\",\"onlinesecurity\"
: \"yes\",\"onlinebackup\": \"no\",\"deviceprotection\": \"yes\",\"techsupport\": \"yes\",\"streamingtv\": \"yes\",\"streamingmovies\": \"yes\",\"contract\"
: \"one_year\",\"paperlessbilling\": \"yes\",\"paymentmethod\": \"bank_transfer_(automatic)\",\"monthlycharges\": 79.85,\"totalcharges\": 3320.75}"
{
  "churn": false,
  "churn_probability": 0.057754360975975874
}
```

Imatge: *Crida al servei web des de la consola*

Podem comprovar que el servidor ens respon, amb el mateix JSON que abans.

5.8. I què més?

En els apartats anteriors hem definit el servei web per interactuar amb el nostre model i poder fer prediccions a partir de les dades d'un nou client.

El client que hem elaborat és molt bàsic, ja que simplement envia unes peticions HTTP. Ens faltaria desenvolupar un client més elaborat: una aplicació web que gestioni la recollida de les dades d'un nou client mitjançant un formulari i, a partir d'elles, munti la petició corresponent al servei web, processi la resposta i generi la pàgina amb el resultat de la predicció.

La darrera passa seria desplegar la nostra aplicació en una plataforma del nigul. Una possibilitat seria AWS Lambda. Tot i que AWS Lambda és un servei de pagament, permet fins a un milió de peticions gratuïtes al mes amb el nivell gratuït d'AWS. Una altra alternativa, que per a petits projectes com aquest seria gratuïta, podria ser PythonAnywhere.

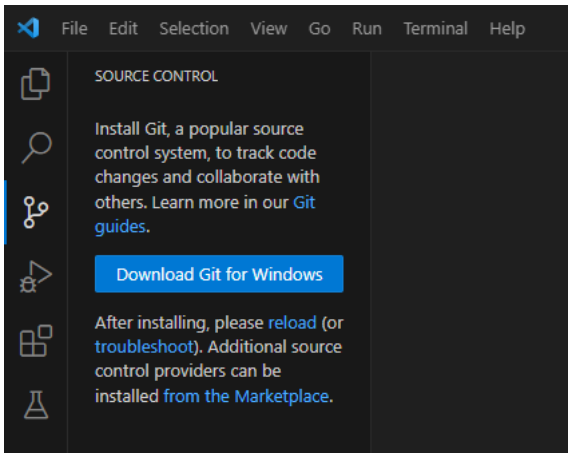
6. Cas pràctic: publicació del projecte

Normalment quan treballam en un projecte, sigui o no relacionat amb Intel·ligència Artificial, ho feim amb més gent i és important fer feina amb un **sistema de control de versions** (VCS, Version Control System).

Git és el sistema de control de versions distribuït més utilitzat. Git fa feina amb repositoris. Un repositori comprèn tota la col·lecció d'arxius i carpetes associats a un projecte, juntament amb l'historial de revisions de cada arxiu. Git emmagatzema tots els canvis que es fan en el projecte, de manera que qualsevol versió anterior pot recuperar-se en qualsevol moment.

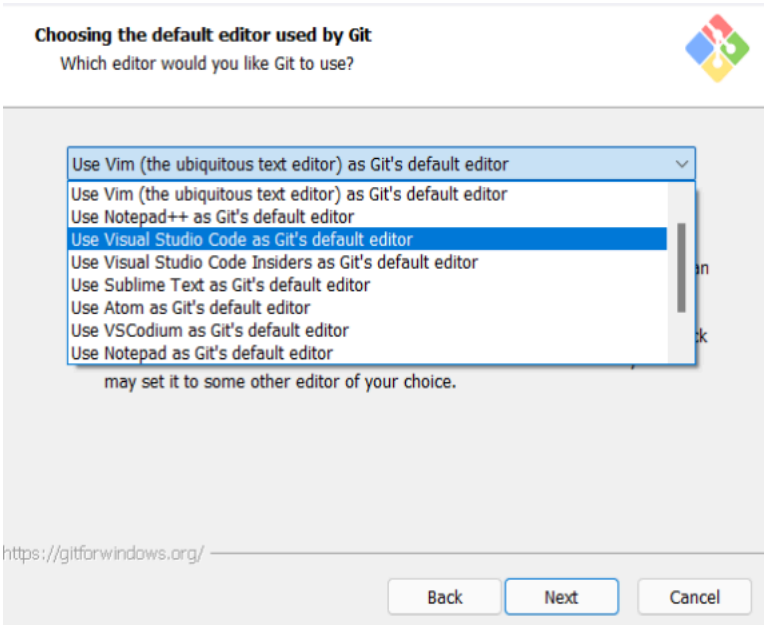
D'altra banda, **GitHub** és un servei de hosting de repositoris Git. D'aquesta manera, podem tenir allotjats els projectes de la nostra organització a un espai a Internet, accessible a tots els programadors de l'organització. GitHub també és molt utilitzat per publicar codi de forma oberta, perquè hi puguin accedir tercers.

Visual Studio Code està completament integrat amb GitHub, de manera que anam a veure com publicar el nostre projecte. Anam al panell de Source Control, pitjant a la icona  .



Imatge: *Panell de Source Control sense Git instal·lat*

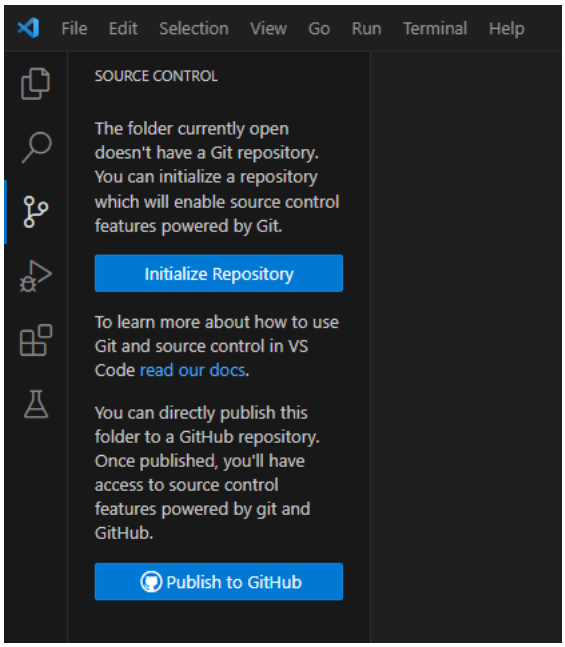
Ens diu que no tenim Git instal·lat en el nostre sistema i ens demana que descarreguem la versió corresponent al nostre sistema operatiu. Ho podem descarregar també directament des de https://git-scm.com/downloads. Una vegada descarregat l'instal·lador, podem deixar totes les opcions per defecte, excepte l'opció on ens demana quin editor de codi volem emprar per defecte. Aquí hem de seleccionar Visual Studio Code.



Imatge: *Editor de codi per defecte*

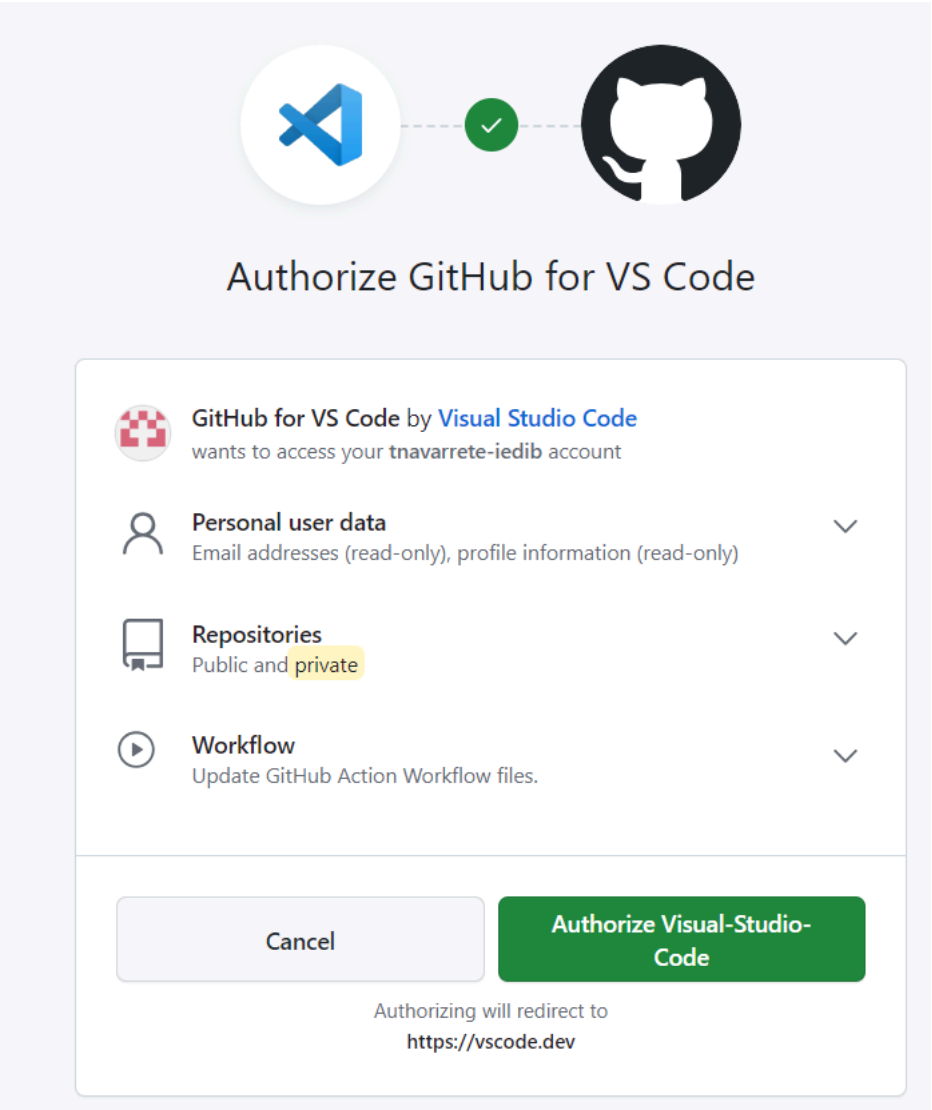
Una vegada instal·lat, si no el teníem anteriorment, hem de crear-nos un compte a GitHub: https://github.com/signup

Quan ara recarregam el panell de Source Control, ja podem comprovar que Git està instal·lat al nostre sistema:



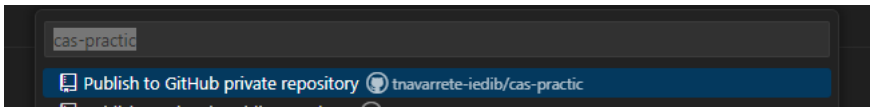
Imatge: *Panell de Source Control amb Git instal·lat*

Aquí hem de triar l'opció de publicar a GitHub ("Publish to GitHub"). Ens demanarà permís per connectar amb GitHub:



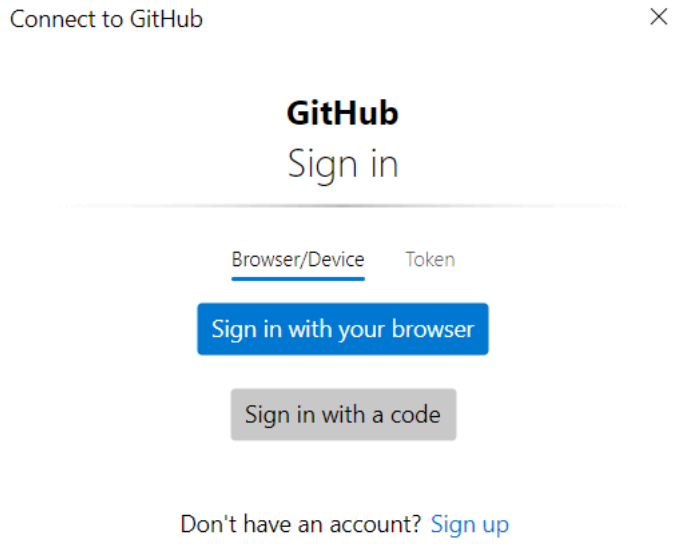
Imatge: *Autorització per connectar GitHub i VS Code*

I després ja podrem publicar el repositori:



Imatge: *Publicació del repositori*

Hem de deixar seleccionats totes les carpetes i arxius. Posteriorment, ens demanarà autenticar-nos al nostre compte de GitHub i confirmar que autoritzam accedir mitjançant el Git Credential Manager al nostre compte.

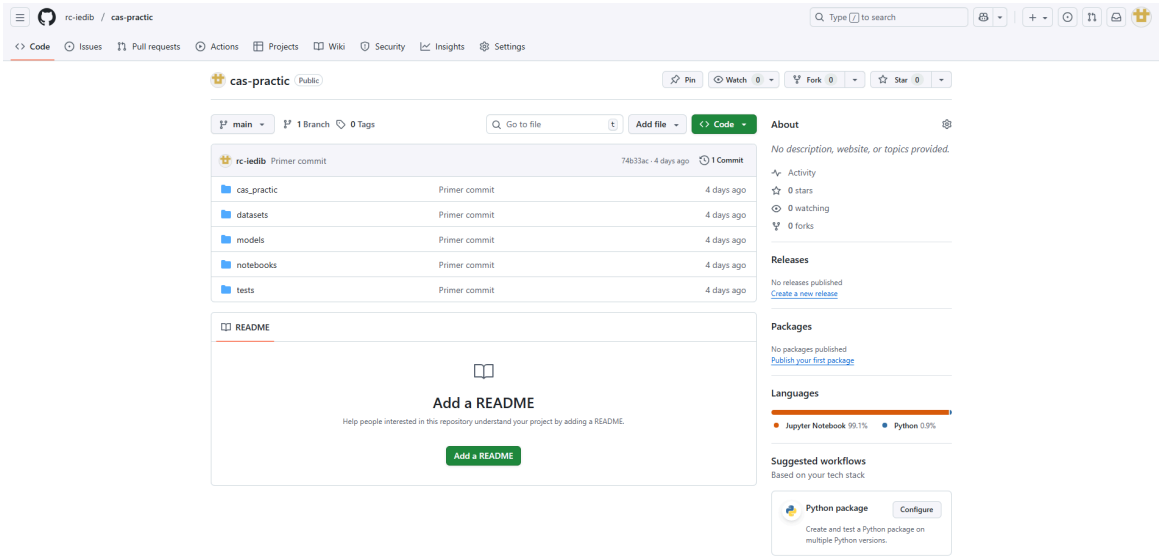


Imatge: *Sign in en GitHub*

Si és la primera vegada que utilitzam GitHub, també haurem de configurar el nom i el correu electrònic per a identificar els commits:

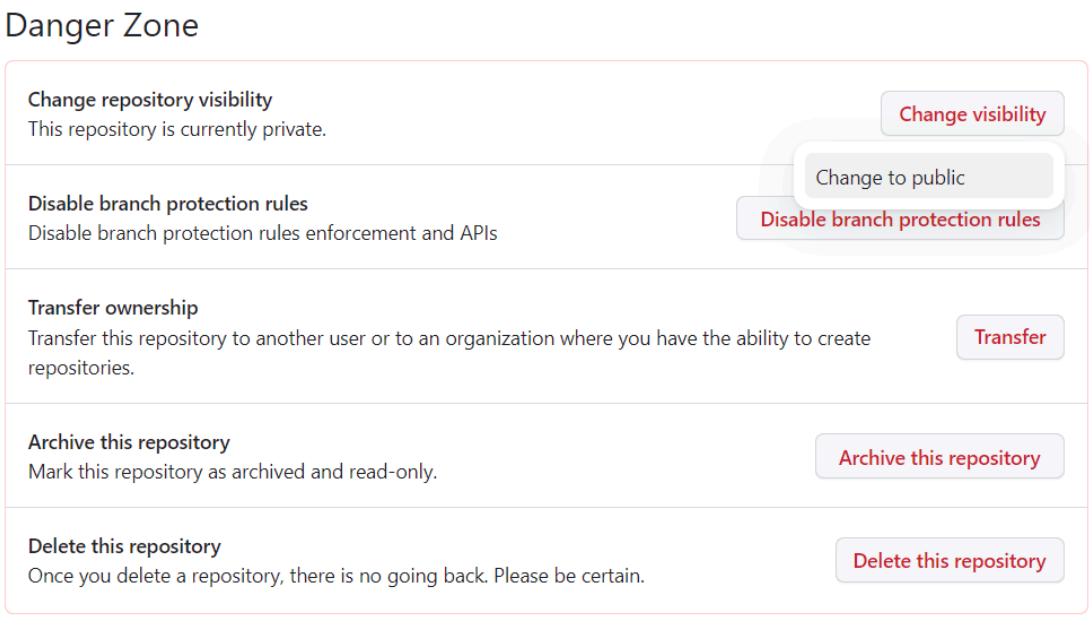
```
git config --global user.name "<user-name>"
git config --global user.email "<user-email>"
```

Amb això ja es publica el nostre projecte en GitHub:



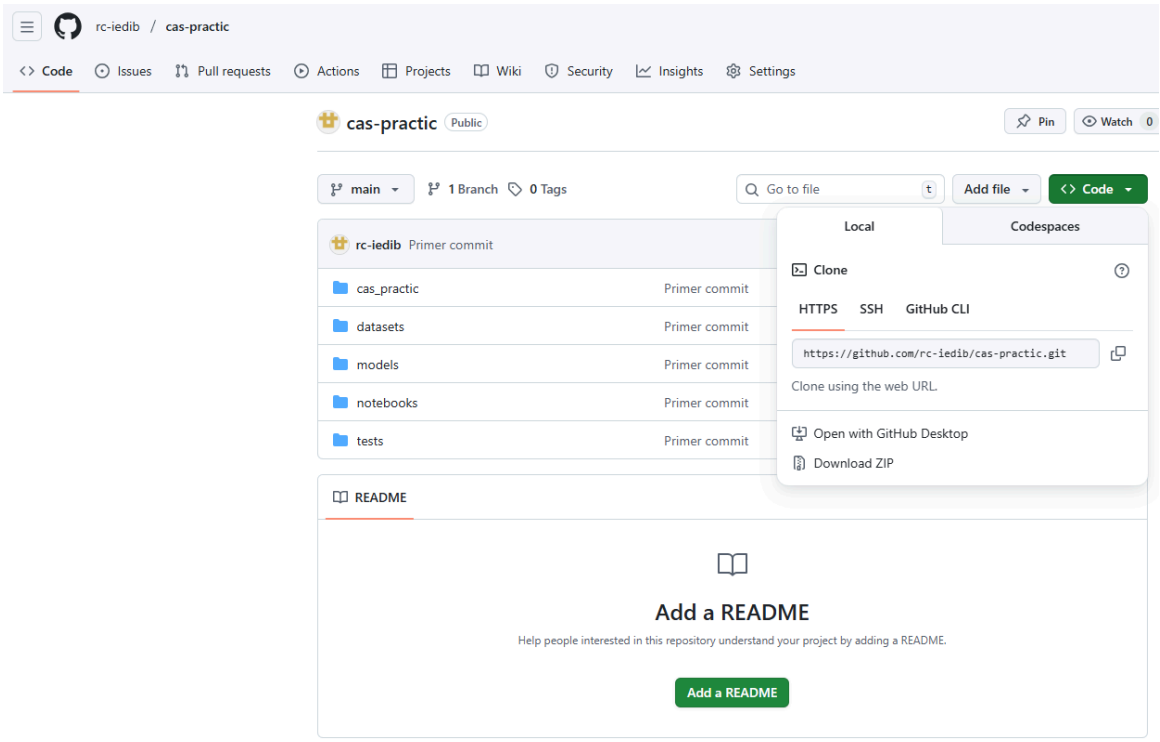
Imatge: *Projecte cas-practic en GitHub*

De moment el repositori és privat, només el puc veure jo. Si el volem fer públic, hem d'anar als Settings (botó a la part superior) i baixar fins a la Danger zone, on hem de canviar la visibilitat i fer-ho públic (haurem de confirmar que volem fer-ho):



Imatge: *Fer públic el repositori*

Ara qui vulgui pot descarregar un zip del nostre projecte.



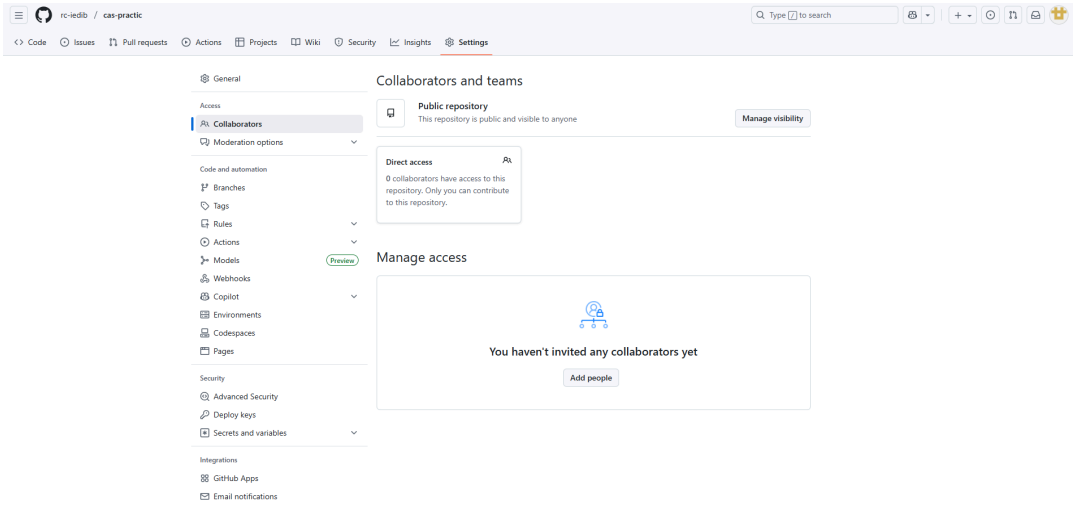
Imatge: *Descarregar un zip del repositori*

De totes maneres, és més senzill fer-ho directament des de Visual Studio Code, ja que té una opció "Clone Git Repository" (l'equivalent a ***git clone***) que automàticament descarregarà les carpetes i arxius i obrirà l'estructura en l'editor. Abans, s'haurà de crear l'entorn virtual per tenir les dependències instal·lades correctament a partir del fitxer YAML (.yml), tal com he vist a les instruccions per exportar o copiar un entorn de Conda.

AMPLIACIÓ

Tant si el repositori és públic com privat, podem convidar altres programadors a participar en el nostre repositori. Aquests col·laboradors tindran permisos per modificar-lo, afegint o eliminant nous arxius i carpetes.

Ho podem fer des dels Settings del repositori, entrant en l'opció Access-Collaborators:

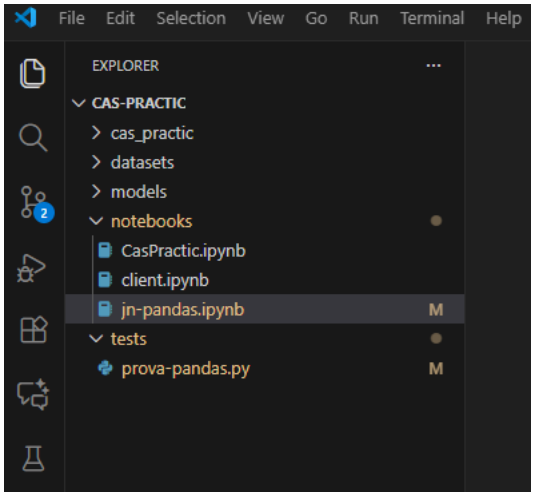


Imatge: *Convidar col·laboradors*

Git funciona amb una interfície d'ordres en una consola bash. I també podem treballar amb aquesta consola des de Visual Studio Code. Mitjançant aquesta interfície, Git permet moltes opcions als desenvolupadors. Especialment importants són les referents a les anomenades branques (versions del projecte). De totes formes, estudiar Git en profunditat s'escapa de l'objectiu d'aquest lliurament. Però sí que anam a veure com sincronitzar un canvi en el codi. Anam a modificar els fitxers prova-pandas.py i jn-pandas.ipynb. Concretament, modificarem la darrera línia, que quedarà així:

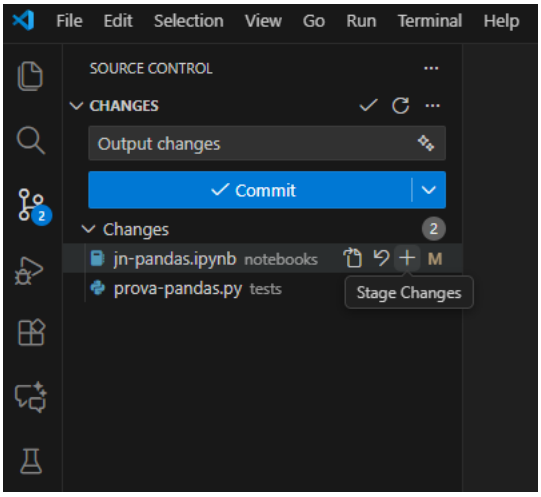
```
print("Superfície total en Km2", df['superficie'].sum())
```

Observem que, quan guardam els fitxers, ens indica amb una "M" que han estat modificats respecte de la versió que està en el repositori.



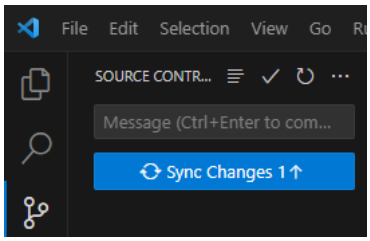
Imatge: *Modificacions en dos fitxers de codi*

Si ara anam al panell de Source Control podem actualitzar el repositori. Per fer-ho, passam els dos canvis a *stage* (l'equivalent a ***git add***) i escrivim un missatge de *commit*, per exemple "Output changes". Per finalitzar, pitjam el botó de "Commit" (l'equivalent a ***git commit -m "Output changes"***):



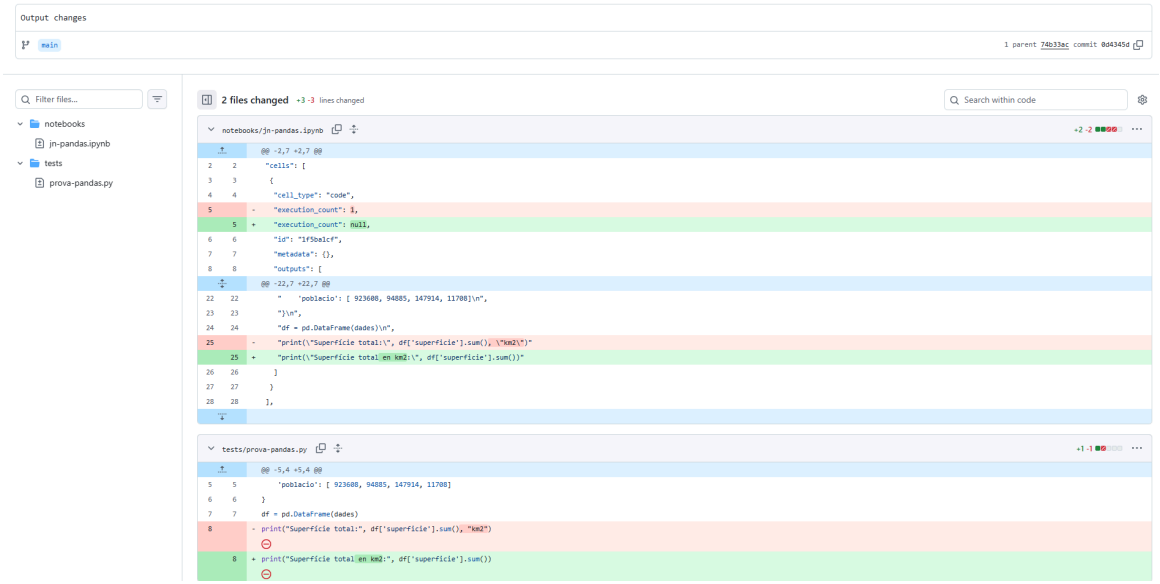
Imatge: *Fer el commit dels canvis*

Això actualitza el repositori local, però encara ens queda sincronitzar el repositori remot, el que tenim en GitHub. Per fer-ho, hem de pitjar el botó "Sync Changes" (l'equivalent a ***git push***).



Imatge: *Sincronitzar els canvis en GitHub*

Podem comprovar que els canvis s'han actualitzat a GitHub:



Imatge: *Canvis actualitzats en GitHub*

AMPLIACIÓ

GitHub és el portal més emprat pels desenvolupadors per publicar codi.

Però si volem donar més visibilitat al nostre projecte ens podem plantejar publicar-lo en una plataforma com **Hugging Face**, àmpliament utilitzada per compartir models, conjunts de dades i aplicacions de l'àmbit de la intel·ligència artificial.

7. Bibliografia

Aquests apunts estan basats parcialment en l'apartat 3.2 dels apunts de *Programación de Inteligencia Artificial* de l'*Escuela Superior de Informática* de la *Universidad de Castilla-La Mancha*.