



MÓDULO I

Introducción de Ciencia de Datos

Curso: Fundamentos de Ciencias de Datos

Sesión 02 – Repositorios, Control de Versiones, R, Python, Environment

III. Configuración de Repositorios y Control de Versiones

- ▶ Introducción a Git y GitHub
- ▶ Creación y configuración de repositorios
- ▶ Prácticas recomendadas para el control de versiones

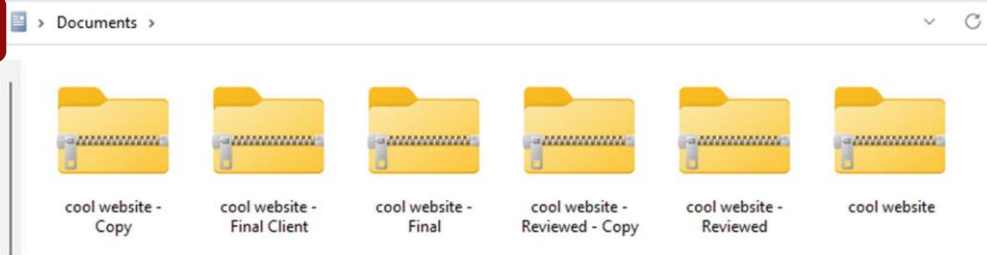
IV. Instalación y Configuración de Environments

- ▶ Instalación de Visual Studio Code para Data Science
- ▶ Instalación y configuración de Python y R
- ▶ Introducción a Jupyter Notebooks y RStudio
- ▶ Gestión de entornos virtuales (venv, conda)

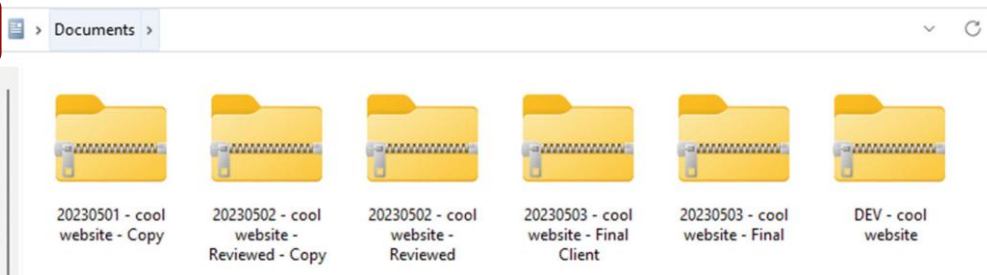
Introducción a Git



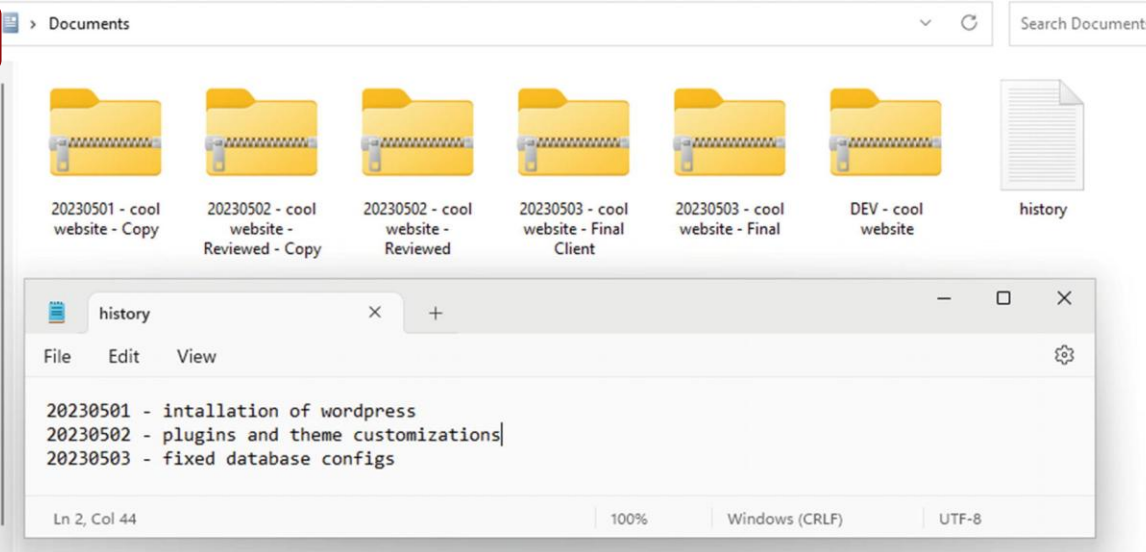
1



2



3



4

Proyecto versionado por Git



¿Qué es el control de versiones?



Definición

- ▶ El control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. En esencia, permite a los desarrolladores gestionar y rastrear las modificaciones hechas en el código fuente de un proyecto, asegurando que cualquier cambio puede ser revertido si es necesario.

Tipos de Sistemas de Control de Versiones (VCS)

- ▶ **Locales:** Almacenan todas las versiones de los archivos en una base de datos local en el mismo equipo donde se realizan los cambios. [Ejemplo: Revision Control System]
- ▶ **Centralizados (CVCS):** Almacenan todas las versiones de los archivos en un servidor central. Los desarrolladores descargan los archivos desde ese servidor, realizan cambios y luego los suben de nuevo. [Ejemplo: Apache Subversion]
- ▶ **Distribuidos (DVCS):** Cada desarrollador tiene una copia completa del repositorio, incluyendo todo el historial de cambios. Esto permite trabajar sin necesidad de estar conectado a un servidor central y facilita la colaboración. [Ejemplo: Git]

Importancia

- ▶ **Seguimiento de Cambios:** Permite ver un historial detallado de los cambios realizados, quién los hizo y cuándo.
- ▶ **Colaboración:** Facilita el trabajo en equipo, permitiendo que múltiples desarrolladores trabajen en diferentes partes de un proyecto sin interferencias. Cada contribuyente puede hacer cambios en su propia rama y luego fusionar esos cambios en el proyecto principal.
- ▶ **Reversión y Recuperación:** Proporciona la capacidad de revertir cambios a versiones anteriores del proyecto, lo cual es crucial en caso de errores o problemas imprevistos.
- ▶ **Control de Calidad:** Permite la revisión de código (code review) y la integración continua (CI), ayudando a mantener la calidad del software mediante la identificación y corrección temprana de errores.
- ▶ **Documentación y Trazabilidad:** Facilita la documentación de los cambios y decisiones tomadas durante el desarrollo, mejorando la trazabilidad y la transparencia del proyecto.

Flujo de Trabajo en Git: Working Directory, Staging Area y Git Directory



1. Working Directory (Directorio de trabajo)

- ▶ Es el área donde trabajas con los archivos de tu proyecto. Aquí es donde haces modificaciones y creas nuevos archivos.
- ▶ **Modifica:** Realizas cambios en tus archivos. No requiere comando Git.

2. Staging Area (Área de preparación)

- ▶ Es una zona intermedia donde se almacenan los cambios que serán incluidos en el próximo commit. Aquí decides qué cambios serán confirmados y cuáles no.
- ▶ **Agregar:** los preparas para el commit. Comando Git: `git add <nombre_del_archivo>` o `git add .`

3. Git directory (Directorio Git)

- ▶ Es el repositorio donde Git almacena todo el historial de cambios y las configuraciones del proyecto. Incluye todos los commits y las versiones del proyecto.
- ▶ **Confirmar (commit):** Confirma los cambios y crea un nuevo punto en el historial del proyecto. Comando Git: `git commit -m "Mensaje descriptivo de los cambios"`

Comandos básicos

- ▶ Inicializar un Repositorio: `git init`
- ▶ Clonar un Repositorio Existente: `git clone <url_del_repositorio>`
- ▶ Verificar el Estado del Repositorio: `git status`: Muestra el estado de los archivos en el working directory y el staging area.
- ▶ Comparar Cambios: `git diff`: Muestra las diferencias entre archivos en el working directory y el staging area.
- ▶ Agregar Archivos al Staging Area: `git add <nombre_del_archivo>`: Añade un archivo específico al staging area. `git add .`: Añade todos los cambios
- ▶ Confirmar Cambios: `git commit -m "Mensaje descriptivo de los cambios"`: Guarda los cambios del staging area en el repositorio con un mensaje descriptivo.
- ▶ Enviar Cambios al Servidor Remoto: `git`

`push`: Envía los commits del repositorio local al servidor remoto.

- ▶ Obtener Cambios del Servidor Remoto: `git pull`: Descarga cambios desde el repositorio remoto y los fusiona con el directorio de trabajo.

- ▶ Historial de Commits: `git log`: Muestra el historial de commits del repositorio.

- ▶ Crear y Gestionar Ramas: `git branch`: Lista todas las ramas en el repositorio. `git branch <nombre_de_la_rama>`: Crea una nueva rama. `git checkout <nombre_de_la_rama>`: Cambia a la rama especificada.

- ▶ Fusionar Ramas: `git merge <nombre_de_la_rama>`: Fusiona la rama especificada con la rama actual.

`Guardar Cambios Temporales: git stash`: Guarda temporalmente los cambios no confirmados en un stack. `git stash apply`: Recupera los cambios guardados en el stack.


Instalar Git





Link de descarga: <https://git-scm.com/downloads>

Primer ToDo: <https://www.jordandataexpert.com/blog/so2-git-task>

Downloads

 macOS

 Windows

 Linux/Unix

Older releases are available and the [Git source repository](#) is on GitHub.

GUI Clients

Git comes with built-in GUI tools ([git-gui](#), [gitk](#)), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the [web interface](#).

Git 2.41.0 Setup

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

☐ Additional icons
☐ On the Desktop
☒ Windows Explorer integration
☒ Git Bash Here
☒ Git GUI Here
☒ Git LFS (Large File Support)
☒ Associate .git* configuration files with the default text editor
☒ Associate .sh files to be run with Bash
☐ Check daily for Git for Windows updates
☒ (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 314.9 MB of disk space.

<https://gitforwindows.org/>

Back Next Cancel

Git 2.41.0 Setup

Choosing the default editor used by Git

Which editor would you like Git to use?

Use Vim (the ubiquitous text editor) as Git's default editor

The [Vim editor](#), while powerful, [can be hard to use](#). Its user interface is unintuitive and its key bindings are awkward.

Note: Vim is the default editor of Git for Windows only for historical reasons, and it is highly recommended to switch to a modern GUI editor instead.

Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.

<https://gitforwindows.org/>

Back Next Cancel

MINGW64: c:/Users/Mariot/Documents/Projects/mynewproject

```
Mariot@WINDOWS-02N95FA MINGW64 ~/Documents/Projects
$ mkdir mynewproject

Mariot@WINDOWS-02N95FA MINGW64 ~/Documents/Projects
$ cd mynewproject/

Mariot@WINDOWS-02N95FA MINGW64 ~/Documents/Projects/mynewproject
$ git init
Initialized empty Git repository in C:/Users/Mariot/Documents/Projects/mynewproject/.git/

Mariot@WINDOWS-02N95FA MINGW64 ~/Documents/Projects/mynewproject (main)
$
```

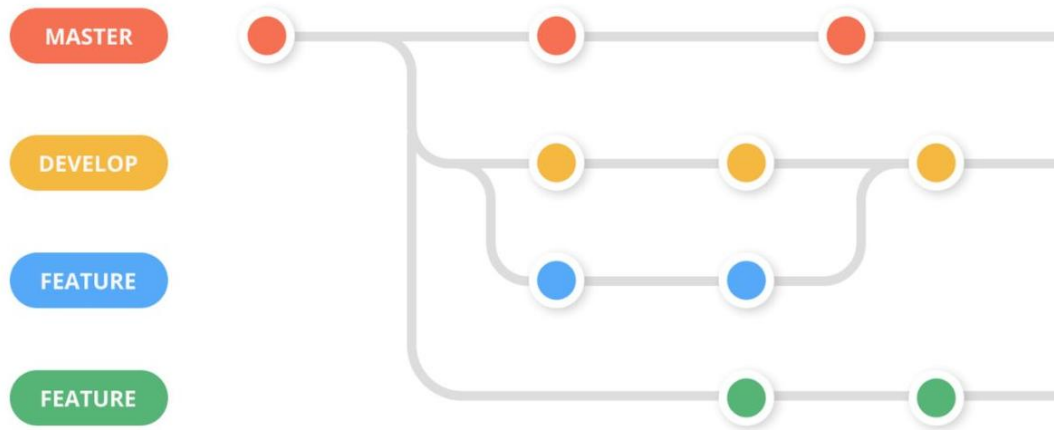
Cand Ph.D Jordan Rodriguez

I Programa de Especialización en Ciencia de Datos | 6

Mejores prácticas de Git



1. Desarrollo en equipo



2. Configura el .gitignore

```
.gitignore x
1 # Ignore all exe files
2 *.exe
3
4 #Except output.exe
5 !output.exe
6 |
```

1. Mejores Prácticas

- ▶ Mantener los mensajes de commit breves (máximo 50 caracteres).
- ▶ Comenzar con mayúscula y evitar puntos finales.
- ▶ Usar el tiempo presente.
- ▶ Asegurarse de que cada commit sea independiente y completo.
- ▶ **Ejemplos:**
 - ▶ [login] Fix typo in DB call
 - ▶ Refactor login function for reuse

2. Peores Prácticas

- ▶ Evitar mensajes vagos como "cambio CSS", "arreglo de errores".
- ▶ No usar Git como sistema de respaldo (commits diarios sin sentido).
- ▶ No abusar del comando git commit --amend.
- ▶ **Ejemplos:**
 - ▶ Fix type
 - ▶ Changing login function by moving declarations to parameters

Remote Git -

> **GitHub**,

GitLab,

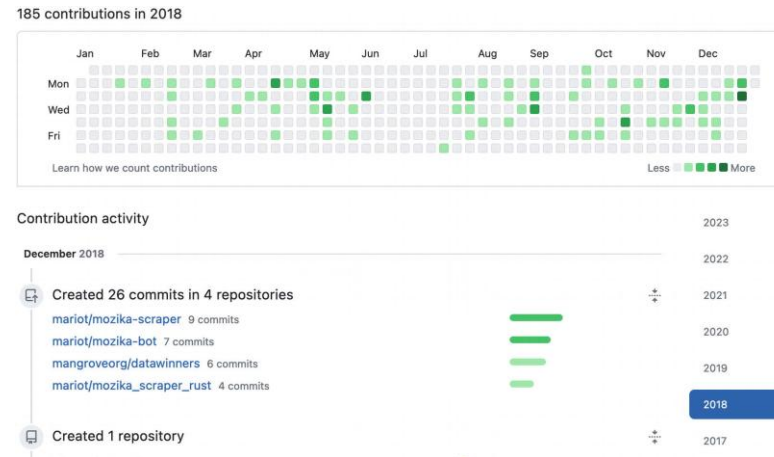
BitBucket,

CodeCloud

Link: Laboratorio

Temas pendientes:

- Issues
- Branches
- Pull requests
- Merge conflicts
- GUI Tools
- GitHub:
 - Wikis
 - Pages
 - Releases
 - Project Board



todo-list

Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.

Invite collaborators
Find people using their GitHub username or email address.

Quick setup — if you've done this kind of thing before

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# todo-list" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:link-skyloft/todo-list.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:link-skyloft/todo-list.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

<> Start writing code

Start a new repository

A repository contains all of your project's files, revision history, and collaborator discussion.

link-skyloft /

☒ **Public**
Anyone on the internet can see this repository

☐ **Private**
You choose who can see and commit to this repository

Create a new repository

Product Solutions Open Source Enterprise Pricing Search or jump to... Sign in Sign up

Let's build from here

The world's leading AI-powered developer platform.

Sign up for GitHub Start a free enterprise trial

Trusted by the world's leading organizations

3M KPMG Mercedes-Benz SAP P&G TELUS

III. Configuración de Repositorios y Control de Versiones

- ▶ Introducción a Git y GitHub
- ▶ Creación y configuración de repositorios
- ▶ Prácticas recomendadas para el control de versiones

IV. Instalación y Configuración de Environments

- ▶ Instalación de Visual Studio Code para Data Science
- ▶ Instalación y configuración de Python y R
- ▶ Configuración de Perfil de Data Science en VS Code
- ▶ Introducción a Jupyter Notebooks y RStudio
- ▶ Gestión de entornos virtuales (venv, conda)

Introducción a Visual Studio Code: es un entorno de desarrollo ligero pero potente

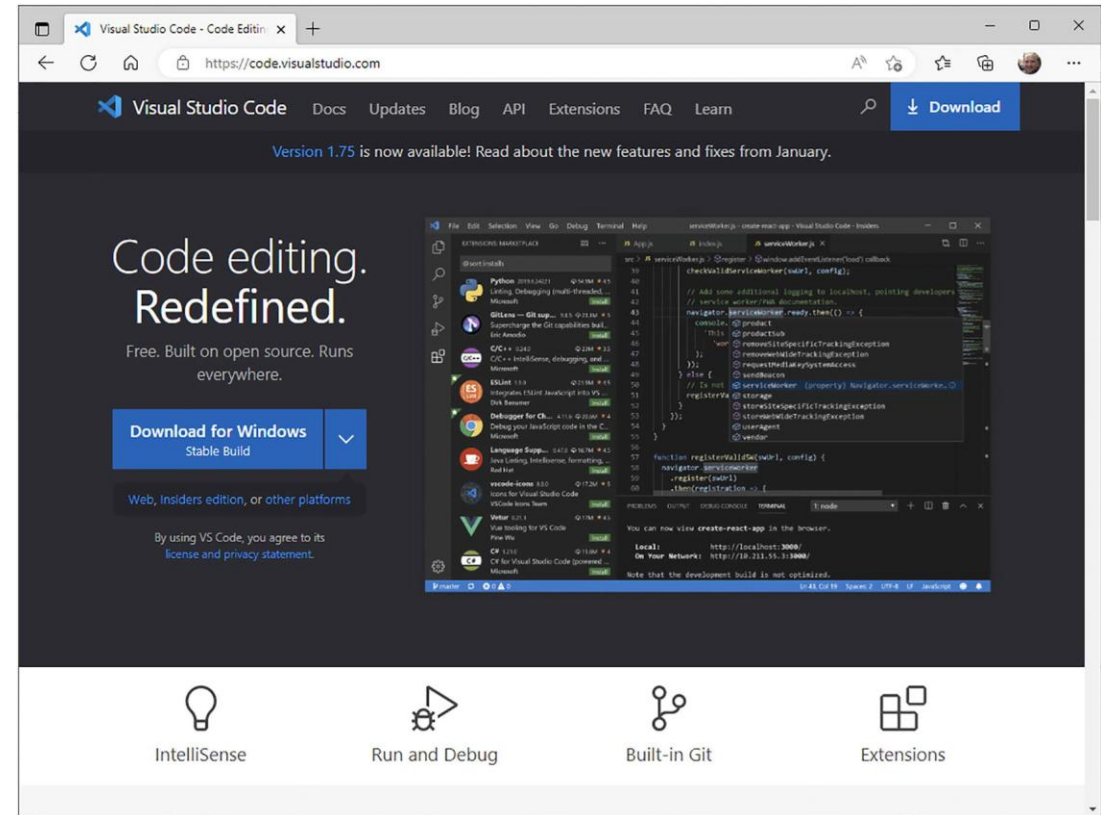


Características Destacadas

- ▶ **Cross-Platform:**
 - ▶ Funciona en Windows, Linux y macOS, lo que permite desarrollar aplicaciones en cualquier entorno.
- ▶ **Extensibilidad:**
 - ▶ IntelliSense: Autocompletado inteligente y herramientas de navegación de código.
 - ▶ Depuración: Depurador integrado para Node.js y extensiones disponibles para otros lenguajes.
- ▶ **Control de Versiones:**
 - ▶ Soporte integrado para Git, permitiendo commits, creación de ramas y gestión de versiones desde la propia interfaz.

Beneficios

- ▶ **Ligero y Rápido:** Menos pesado que los entornos de desarrollo tradicionales como Visual Studio 2022.
- ▶ **Versatilidad:** Ideal tanto para pequeños scripts como para grandes proyectos.
- ▶ **Comunidad y Extensiones:** Gran cantidad de extensiones disponibles en el marketplace de Visual Studio Code.

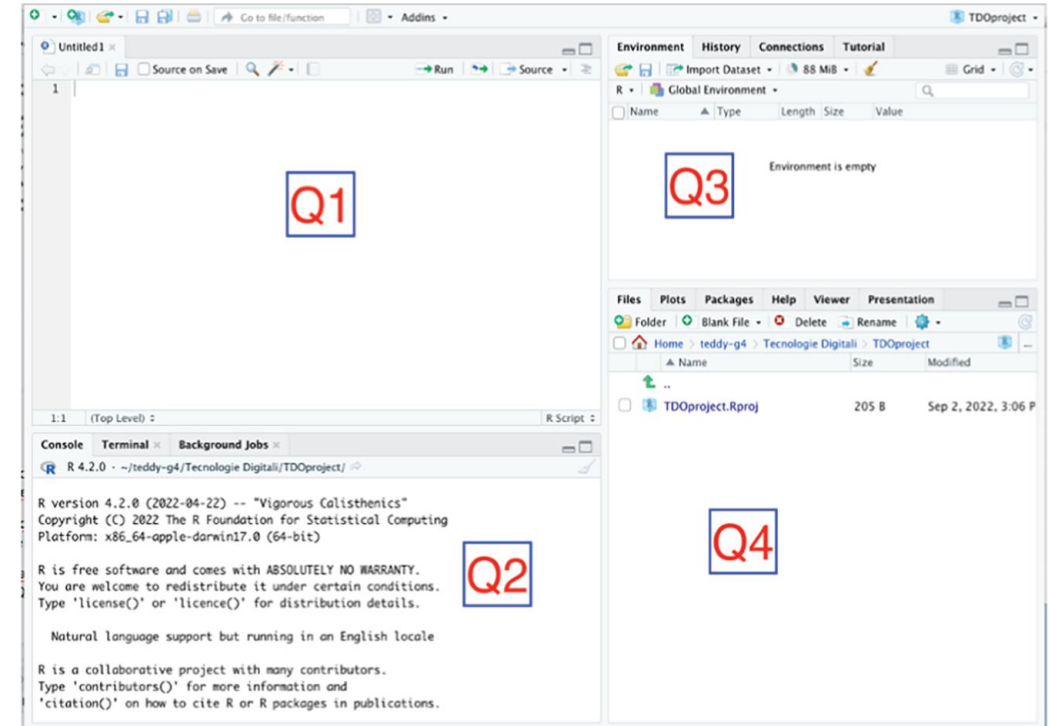


Instalando R



R Language:

- ▶ Desarrollado por la comunidad para análisis estadístico.
- ▶ CRAN (Comprehensive R Archive Network) es el archivo oficial en línea para todas las versiones de R y paquetes de software.
- ▶ RStudio IDE:
- ▶ Herramienta gráfica que ofrece una interfaz amigable para el desarrollo de proyectos en R.
- ▶ RStudio Desktop y RStudio Cloud son versiones disponibles, con la última ofrecida como servicio en la nube.
- ▶ Instalación:
 - ▶ R es compatible con Windows, macOS y Linux.
 - ▶ RStudio se instala después de instalar R, proporcionando una interfaz gráfica robusta.
- ▶ Uso: Paquetes: Instalación y manejo de paquetes adicionales a través de CRAN para extender las funcionalidades de R. Tidyverse: Paquete principal utilizado en data science que agrupa varios paquetes importantes como dplyr, ggplot2, tidyr, y más.

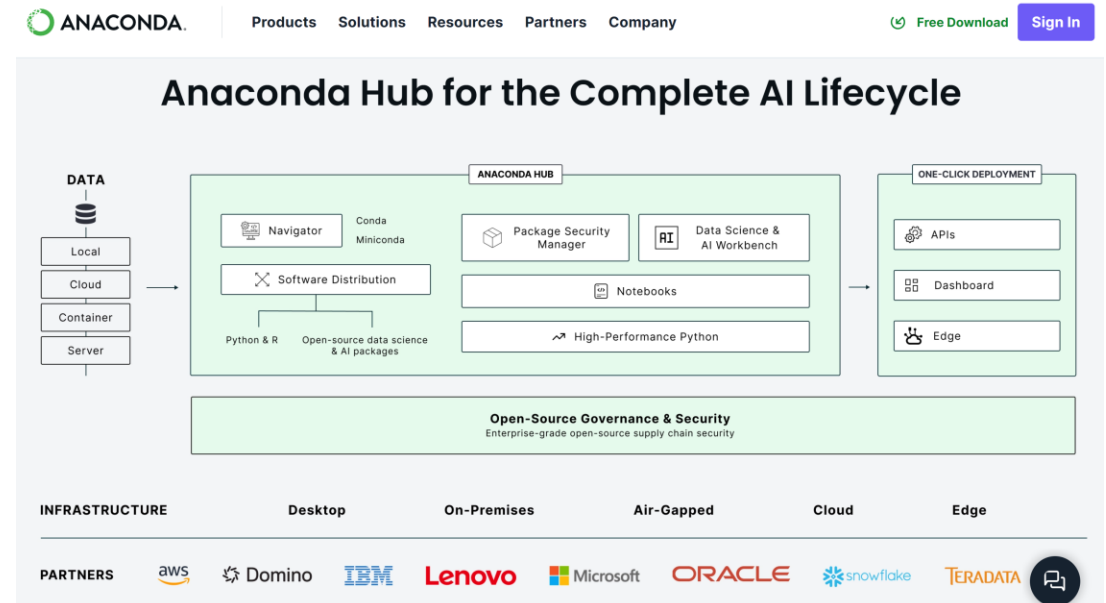


Instalando Python



Python Language:

- ▶ Python es un lenguaje de programación de propósito general ampliamente utilizado en diversas aplicaciones, incluidas la ciencia de datos y el desarrollo web.
- ▶ Lenguaje interpretado y de alto nivel, fácil de aprender y usar. Amplia comunidad de desarrolladores y gran cantidad de bibliotecas y paquetes.
- ▶ **Anaconda:**
 - ▶ Incluye Python, el gestor de paquetes conda, y herramientas esenciales como JupyterLab y Spyder. Ofrece un instalador todo en uno para simplificar la configuración del entorno de desarrollo.
- ▶ Incluye Python, el gestor de paquetes conda, y herramientas esenciales como JupyterLab y Spyder. Ofrece un instalador todo en uno para simplificar la configuración del entorno de desarrollo.



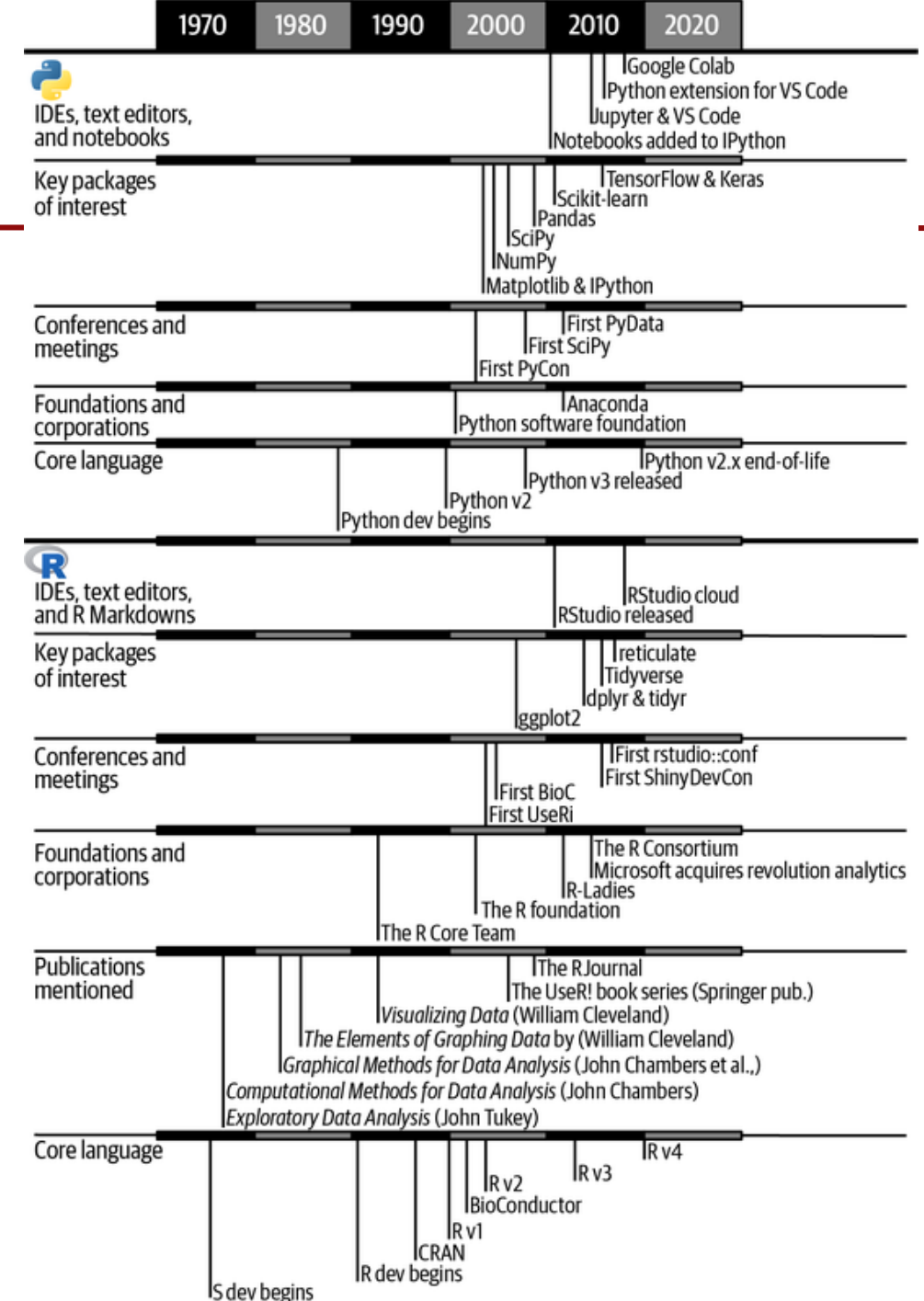
Origen de R y Python

R: herramienta específica para estadísticos

- ▶ Origen: Bell Laboratories en 1976 con el lenguaje S, desarrollado por John Chambers.
- ▶ Inspiración: Filosofía FUBU (For Us, By Us) para estadísticos.
- ▶ Evolución: En 1991, Ross Ihaka y Robert Gentleman crearon R en la Universidad de Auckland.
- ▶ Características: Enfoque en análisis de datos, visualizaciones flexibles y comunidad inclusiva.
- ▶ Lanzamiento: Primera versión estable R v1.0.0 en 2000.
- ▶ Infraestructura: CRAN para paquetes y el equipo R Core Team.

Python: Lenguaje de propósito general

- ▶ Origen: Creado por Guido van Rossum en 1991 en Países Bajos.
- ▶ Propósito: Resolver problemas comunes de programación con una sintaxis simple
- ▶ Versatilidad: Usado en desarrollo web, administración de sistemas, aplicaciones de escritorio y más.
- ▶ Ascenso en Ciencia de Datos: Paquetes como NumPy (2005), SciPy, pandas (2009) y herramientas de aprendizaje profundo como TensorFlow y Keras.
- ▶ Popularidad: Fácil adopción debido a su simplicidad y capacidad de integración en diferentes áreas.

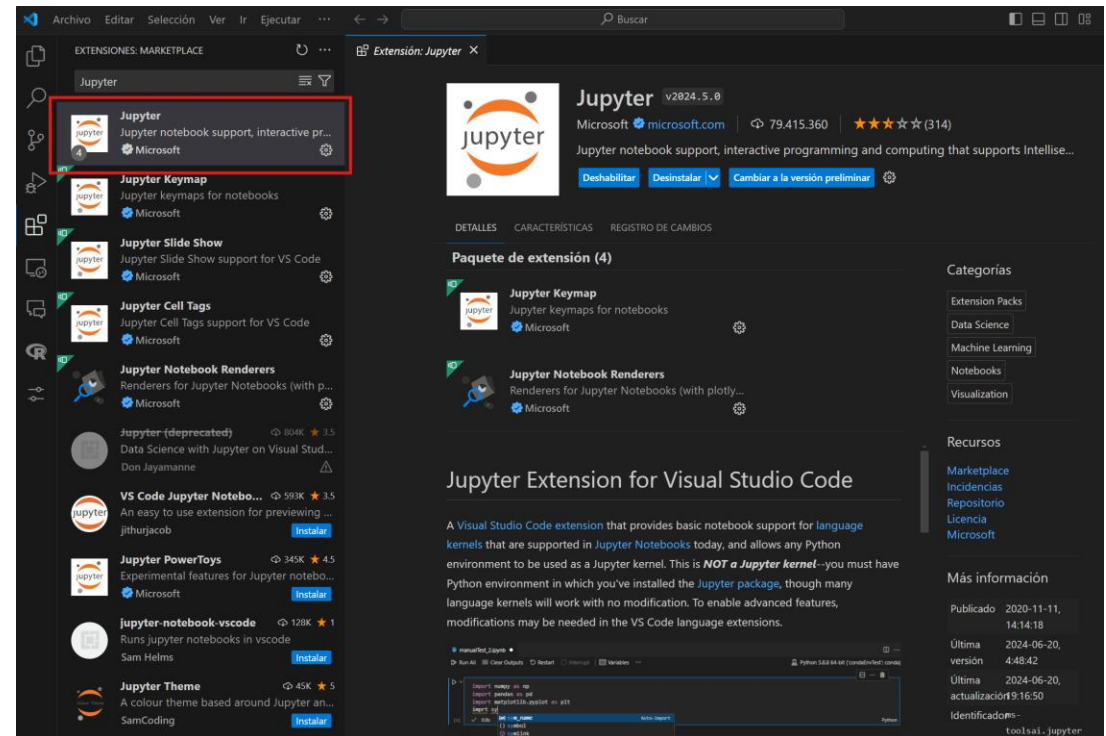
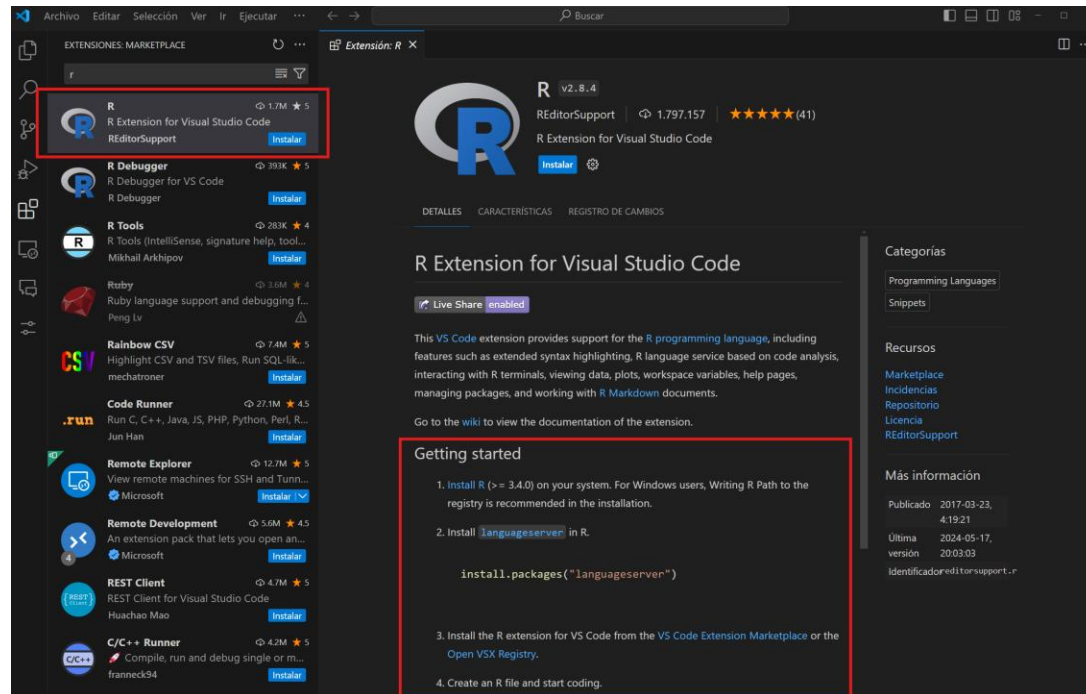


Creación de un Perfil en Visual Studio Code para Proyectos de Data Science

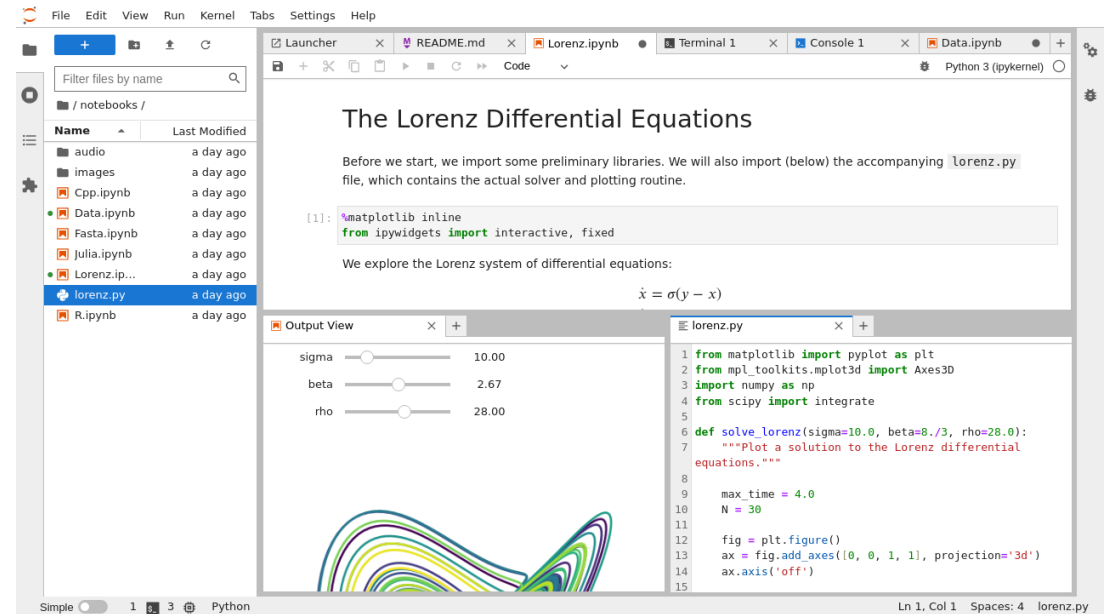
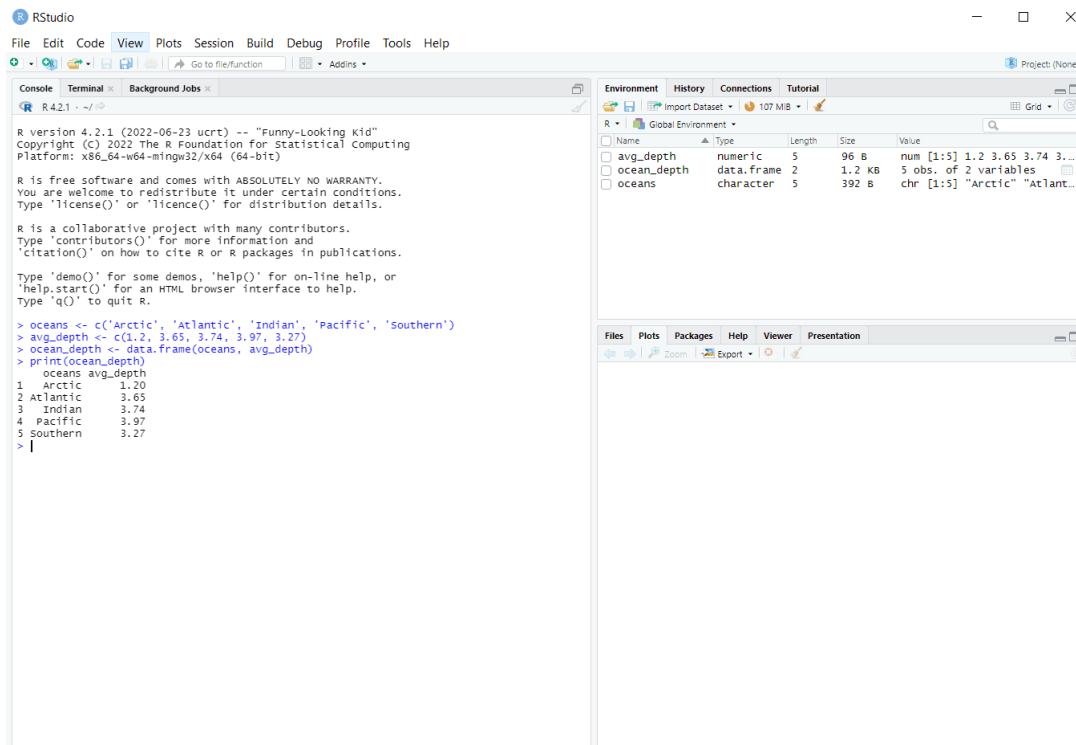


Facultad de
Ingeniería
Económica,
Estadística y
Ciencias
Sociales

Centro de
Formación
Continua



Instalación de Anaconda y RStudio





MÓDULO I

Introducción de Ciencia de Datos

Curso: Fundamentos de Ciencias de Datos

Sesión 02