



Ciencia de Datos con Python. Una introducción para usuarios de R

Daniel Ciganda, Facundo Morini, Bruno Tancredi

Por qué Python?



Figure 1: Monty Python

- **Popularidad:** Python es uno de los lenguajes de programación de más rápido crecimiento.
- **Versatilidad:** Utilizado en desarrollo web, análisis de datos, aprendizaje automático, finanzas y más.
- **Soporte de la Comunidad:** Gran ecosistema de bibliotecas y frameworks. Más de 500.000 paquetes disponibles en PyPI (Python Package Index) vs aprox. 20.000 en CRAN (a Octubre 2024).

Top programming languages on GitHub

RANKED BY COUNT OF DISTINCT USERS CONTRIBUTING TO PROJECTS OF EACH LANGUAGE.



Global Programming Language Ranking Report - Q4 2024					
Rank	Language	Popularity Rating (%)	Change (%)	Category	Sub-Categories
1	Python	21.90%	+7.08%	General Purpose	Web Development, Data Science, Machine Learning
2	C++	11.60%	+0.93%	General Purpose	System Programming, Game Development
3	Java	10.51%	+1.59%	General Purpose	Enterprise Applications, Web Development
4	C	8.38%	-3.70%	General Purpose	System Programming, Embedded Systems
5	C#	5.62%	-2.09%	General Purpose	Game Development, .NET Framework
6	JavaScript	3.54%	+0.64%	Scripting	Frontend Development, Server-Side Scripting
7	Visual Basic	2.35%	+0.22%	General Purpose	Desktop Application Development
8	Go	2.02%	+0.65%	General Purpose	Systems Programming, Cloud Native
9	Fortran	1.80%	+0.78%	General Purpose	Scientific Computing, Numerical Analysis
10	Delphi/Object Pascal	1.68%	+0.38%	General Purpose	Desktop Application Development
11	SQL	1.64%	-0.15%	Database	Relational Databases, Data Management
12	MATLAB	1.48%	+0.22%	General Purpose	Mathematics, Engineering
13	Rust	1.45%	+0.53%	General Purpose	Systems Programming, Safety-Critical
14	Scratch	1.41%	+0.05%	General Purpose	Education, Kids Programming
15	PHP	1.21%	-0.69%	General Purpose	Web Development, Content Management
16	Assembly language	1.13%	-0.51%	General Purpose	Low-Level Programming, Embedded Systems
17	R	1.09%	+0.12%	General Purpose	Statistics, Data Analysis
18	Ruby	0.99%	+0.07%	General Purpose	Scripting, Web Development
19	COBOL	0.99%	+0.23%	General Purpose	Legacy Systems, Mainframe
20	Swift	0.98%	-0.09%	General Purpose	Mobile Application Development

Ecosistema Simple (R) vs Ecosistema Complejo (Python)

- R fue diseñado por y para estadísticos, con un ámbito de aplicación bien definido y una comunidad relativamente homogénea.
- Al utilizarse para una gran variedad de aplicaciones, el ecosistema de Python es más complejo.
- Para comenzar a trabajar con R, no hay que tomar un número grande de decisiones. La mayoría de los usuarios utilizan RStudio, existe un sólo repositorio de paquetes centralizado y las dependencias se gestionan internamente desde el lenguaje mismo con `install.packages()`.
- En Python debemos tomar una cantidad de decisiones antes de poder empezar a trabajar. Desde la elección del IDE (VS Code, PyCharm, Spyder), hasta la instalación de paquetes (`pip`, `conda`), o la gestión de dependencias (`virtualenv`, `venv`, `conda env`, `pipenv`, `poetry`).





La Elección del IDE - Integrated Development Environment -

- **Visual Studio Code (VS Code)**

- Editor de código fuente desarrollado por Microsoft.
- Soporta extensiones para Python que proporcionan copiloto y debugging avanzado.
- Ligero y altamente personalizable con extensiones.
- Amplia comunidad y soporte.

- **PyCharm**

- IDE especializado en Python desarrollado por JetBrains.
- Disponible en versiones Community (gratuita) y Professional (de pago).
- Soporte para desarrollo web y ciencias de datos en un mismo editor, integración con herramientas de base de datos y control de versiones.
- Características avanzadas como depuración, testeo integrado y refactorización de código.

- **Spyder**

- IDE orientado a la ciencia de datos y a la programación científica.
- Incluye una consola interactiva, explorador de variables y editor de código.
- Interfaz similar a RStudio, facilitando la transición para usuarios de R.
- Integración con paquetes científicos como NumPy, SciPy y Matplotlib.

Gestión de Dependencias: R vs Python



- Repositorio único (CRAN)
- Control estricto de paquetes
- Instalación automática de dependencias
- Comunidad enfocada en la compatibilidad hacia atrás y la estabilidad



- PyPI es el repositorio oficial, pero existen múltiples
- Menor control de paquetes
- Comunidad enfocada en la flexibilidad
- Más conflictos de dependencias
→ “Dependency hell”

CONFLICTOS DE DEPENDENCIAS





NUEVOS PAQUETES

PYPI

TU

Gestores de Paquetes en Python: pip & conda

- **pip**

- Es el gestor de paquetes oficial de Python.
- Utilizado para instalar paquetes desde el Python Package Index (PyPI).
- Sencillo y viene preinstalado con Python desde la versión 3.4.
- Amplia disponibilidad de paquetes en PyPI.
- No gestiona dependencias de sistemas ni entornos virtuales.
- Puede presentar conflictos de versiones si no se usa junto con entornos virtuales.

- **conda**

- Gestor de paquetes y entornos que puede manejar dependencias de Python y otros lenguajes.
- Utilizado en distribuciones populares como Anaconda y Miniconda.
- Gestiona paquetes y dependencias a nivel de sistema
- Facilita la creación de entornos aislados.

pip vs. conda

En general se recomienda utilizar `conda` e instalar con `pip` únicamente los paquetes o versiones que no se consiguen en los repositorios de `conda`.

Ejemplo: Instalando SciPy

- Pip:**
- Obtiene SciPy
 - Pero necesitas instalar las bibliotecas C/Fortran por tu cuenta
 - Puede requerir múltiples pasos e instalaciones manuales

- Conda:**
- Obtiene SciPy y todas las bibliotecas C/Fortran necesarias
 - Todo se instala en un solo paso
 - Funciona inmediatamente

Entornos Virtuales en Python

- **¿Qué son los entornos virtuales?**
 - Son entornos aislados que permiten ejecutar Python con paquetes y configuraciones específicas sin afectar al sistema global.
- **¿Por qué son necesarios?**
 - **Manejo de Dependencias:** Permiten tener versiones específicas de paquetes para diferentes proyectos.
 - **Evitar Conflictos:** Previenen conflictos entre paquetes y versiones en distintos proyectos.
 - **Sin Permisos de Administrador:** Permiten instalar paquetes sin necesidad de permisos en sistemas restringidos.
 - **Portabilidad:** Facilitan la reproducción del entorno en otras máquinas o servidores.
- **Herramientas para la gestión de Entornos Virtuales**
 - `virtualenv`, `pipenv`
 - `poetry`
 - `conda env`

Buenas Prácticas en Gestión de Paquetes

- **Usar Entornos Aislados:** Siempre trabajar dentro de entornos virtuales (`conda env`, `virtualenv`).
- **Orden de Instalación:**
 1. Instalar paquetes disponibles con `conda` primero.
 2. Luego, usar `pip` para paquetes que no están en los canales de `conda`.
- **Documentar Dependencias:** Guardar los comandos de instalación y versiones específicas utilizadas.
- **Reproducibilidad:** Utilizar archivos de requisitos (`environment.yml`, `requirements.txt`) para compartir entornos.

Flujo de Trabajo

- **R**
 - Desarrollo centrado en **RStudio**, que integra editor, consola, visualización de datos y más en una sola interfaz.
 - Menor dependencia del terminal; la mayoría de las tareas se realizan dentro de RStudio.
 - Flujo de trabajo altamente interactivo, ideal para análisis exploratorio.
- **Python**
 - Mayor uso del **terminal** para ejecutar scripts, gestionar entornos y paquetes.
 - Aunque existen entornos interactivos (Spyder, Jupyter Notebooks), el flujo de trabajo tradicional es menos interactivo.
 - Requiere alternar entre el editor y el terminal en muchos casos.

¿Qué es Miniconda?

- **Miniconda** es una distribución mínima de **Conda**, un gestor de paquetes y entornos para Python.
- **Características principales:**
 - Incluye únicamente **Conda** y sus dependencias esenciales.
 - Permite crear entornos virtuales y gestionar paquetes de manera eficiente.
 - Es ligero y ocupa menos espacio que Anaconda completa.

Instalación de Miniconda y Configuración de Entorno Virtual en Spyder

1. Instalar Miniconda:

- Descarga el instalador desde
<https://docs.conda.io/en/latest/miniconda.html>
- Sigue las instrucciones del instalador para tu sistema operativo.

2. Instalar Spyder en el entorno base:

- Abre la terminal de anaconda (Anaconda prompt) y ejecuta: `conda install spyder`

3. Crear un entorno virtual:

- Crea el entorno: `conda create -n curso_env`
- Activa el entorno: `conda activate curso_env`
- Instala paquetes: `conda install pandas, scikit-learn, matplotlib, seaborn`

- **Asociar el entorno virtual en Spyder:**

- Abre Spyder.
- Ve a **Preferencias → Intérprete de Python.**
- Selecciona **Usar el siguiente intérprete de Python.**
- Navega hasta el intérprete de tu entorno virtual:
 - En Windows: C:\Ruta_a_Miniconda\envs\mi_entorno\python.exe
 - En macOS/Linux: /ruta_a_miniconda/envs/mi_entorno/bin/python
- Aplica los cambios y reinicia Spyder.

Descargar el Repositorio

Desde el directorio donde vamos a alojar el repositorio:

Click derecho → *git bash here*

Ejecutar: `git clone https://github.com/FaMori/Taller_Python_SUE24`