

Entornos Virtuales



Introducción

Bienvenidos al módulo de Entornos virtuales en Python. Los objetivos de aprendizaje estarán orientados a incorporar conocimientos básicos que nos permitirán crear entornos virtuales para versiones específicas de Python y gestionar las dependencias en ellos.

Una vez finalizado este módulo serás capaz de:

- Realizar el setup de Anaconda
- Ejecutar los primeros Notebooks
- Crear entornos virtuales con la librería “venv” en distintas versiones de Python.
- Almacenar y recuperar dependencias con un archivo requirements.txt



Tema 1. Setup de Suite Anaconda

Objetivo

El objetivo de este tema es brindar una descripción de la suite anaconda y de Jupyter Notebook.

Una vez finalizado este tema serás capaz de:

- Conocer las funcionalidades esenciales de Anaconda y Jupyter Notebook
- Realizar el setup de estas herramientas.
- Ejecutar el primer notebook

Anaconda, que es?

Anaconda es una distribución open-source de los lenguajes de Python y R.

Entre sus características se destacan:

- ☐ Ser una Aplicación gráfica que integra las herramientas y versiones de Python orientadas al desarrollo científico.
- ☐ IDE para desarrollar fácilmente aplicaciones
- ☐ Gestor de paquetes integrado (Conda).
- ☐ Integración avanzada con Jupyter Notebook.
- ☐ Soporte de entornos virtuales de Python.
- ☐ Permite utilizar Python y R en el mismo entorno.

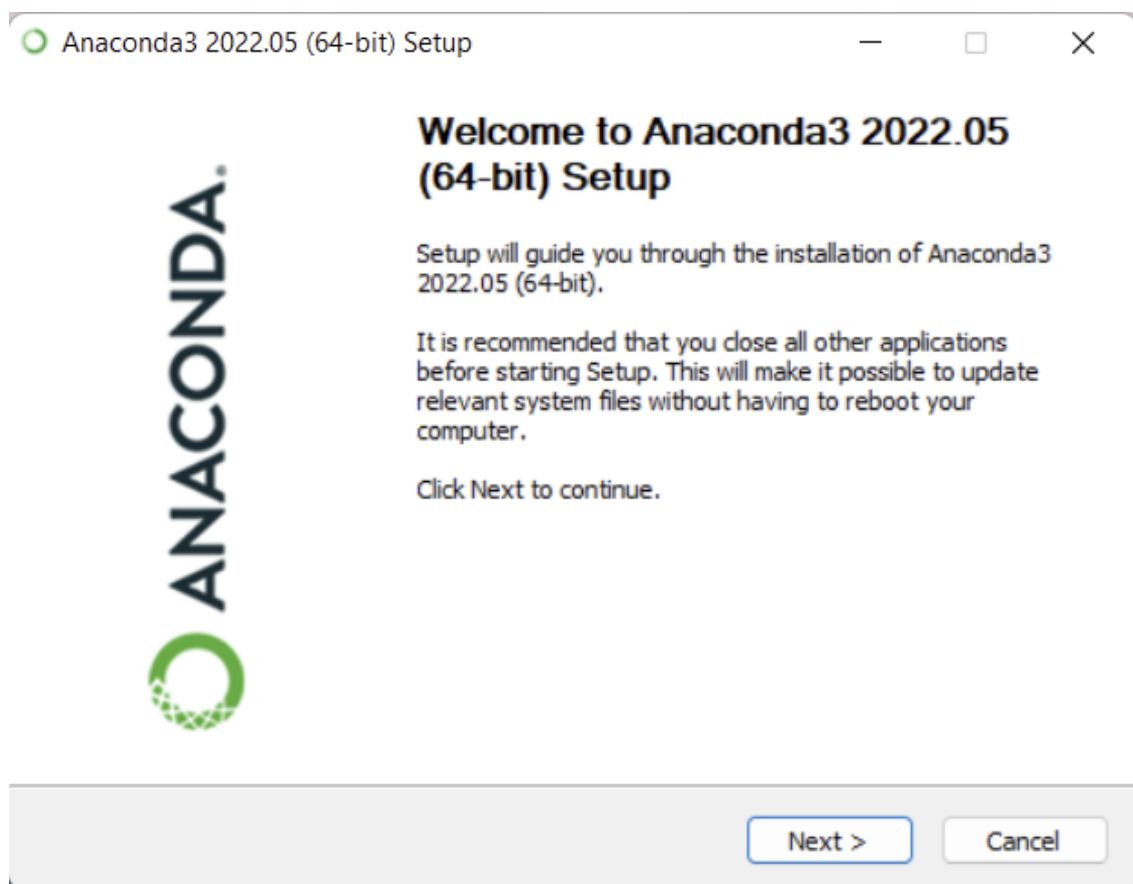
La principal ventaja de Anaconda, es la de simplificar la gestión e implementación de paquetes. Las versiones de paquetes en Anaconda son administradas por el sistema de gestión de paquetes conda.

Instalación de Anaconda

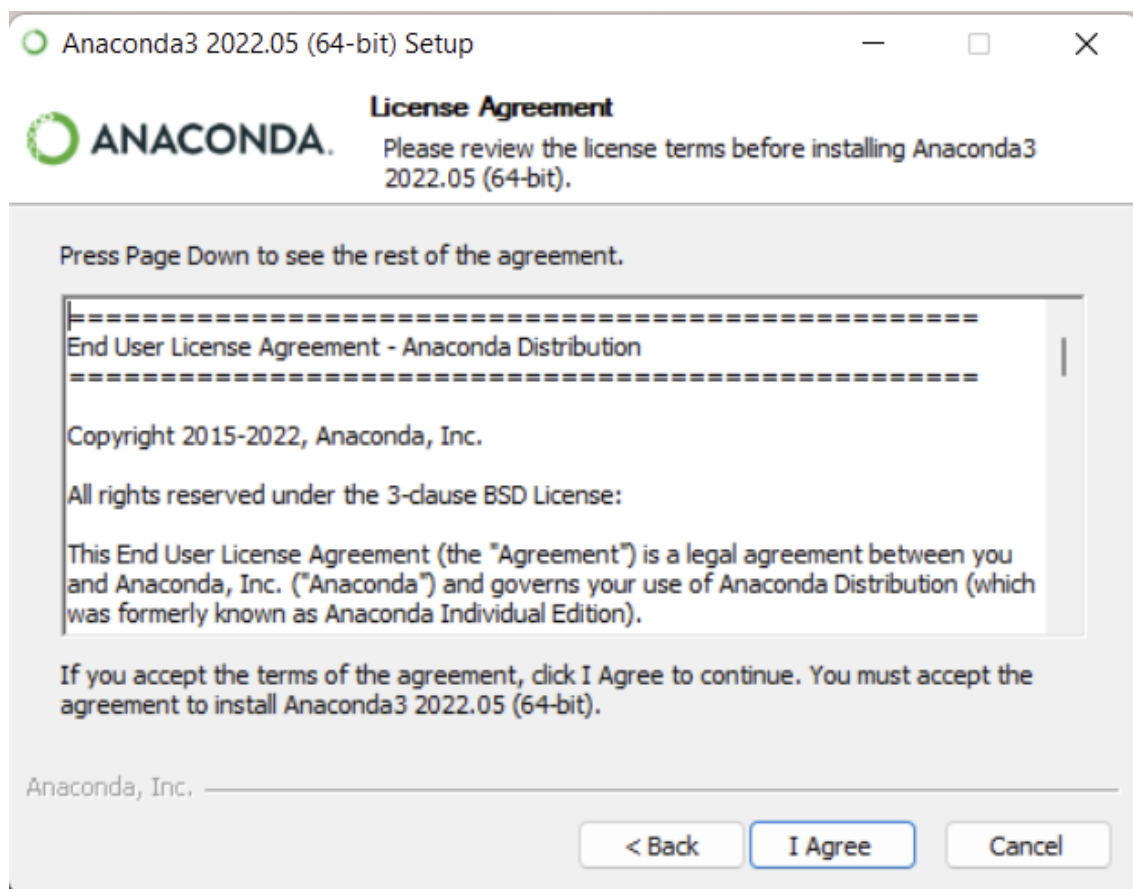
Para comenzar con la instalación, procederemos a realizar la descarga en el siguiente Link:

<https://www.anaconda.com/>

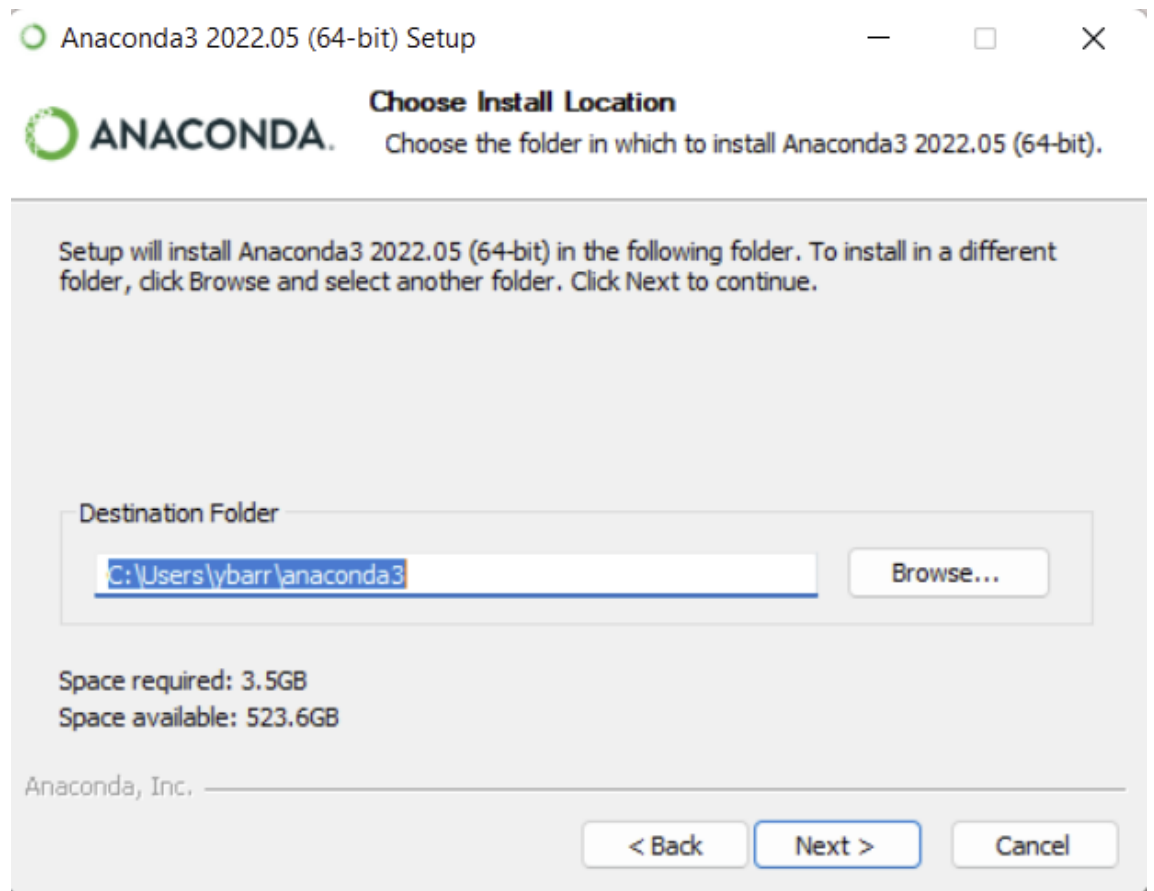
Una vez finalizada la descarga, comenzaremos con la instalación, la cual es muy simple. La pantalla inicial de la instalación, es la siguiente:



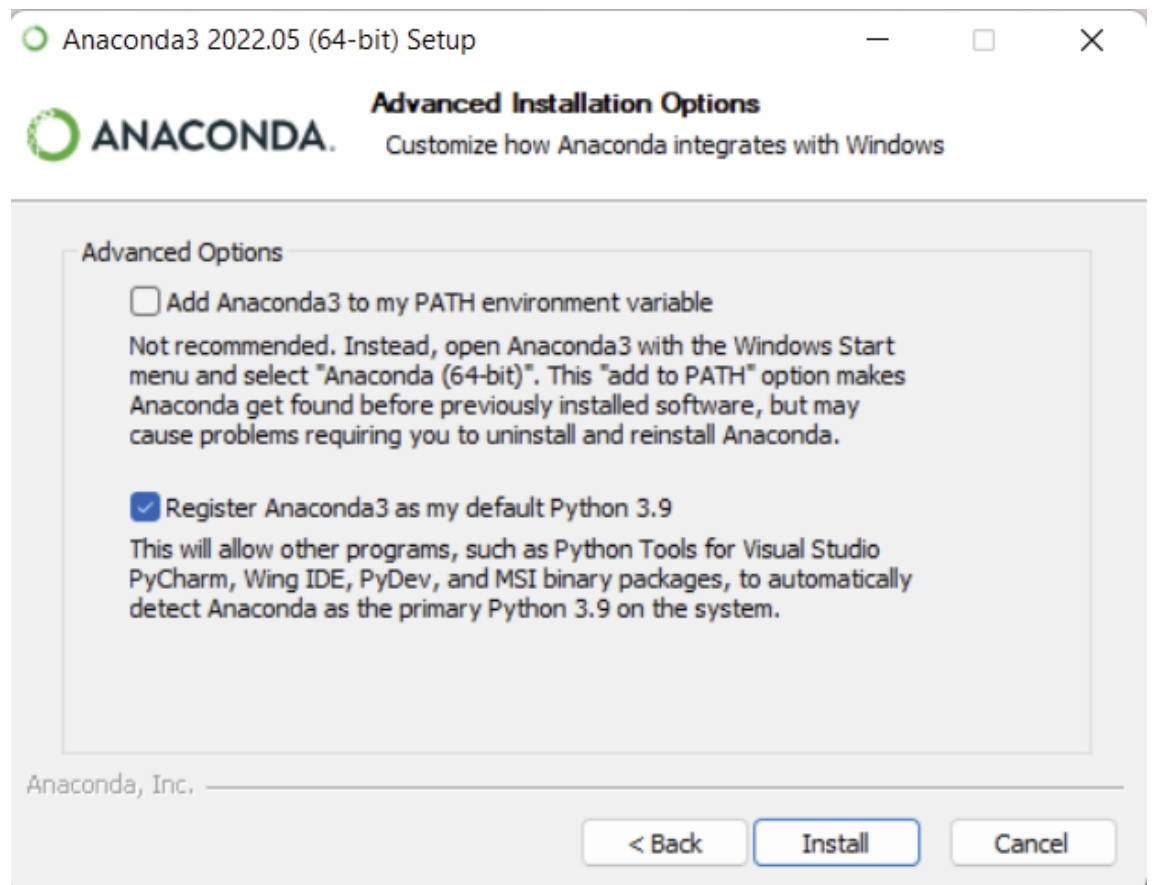
Presionamos “Next”. Luego nos aparecerá la siguiente pantalla:



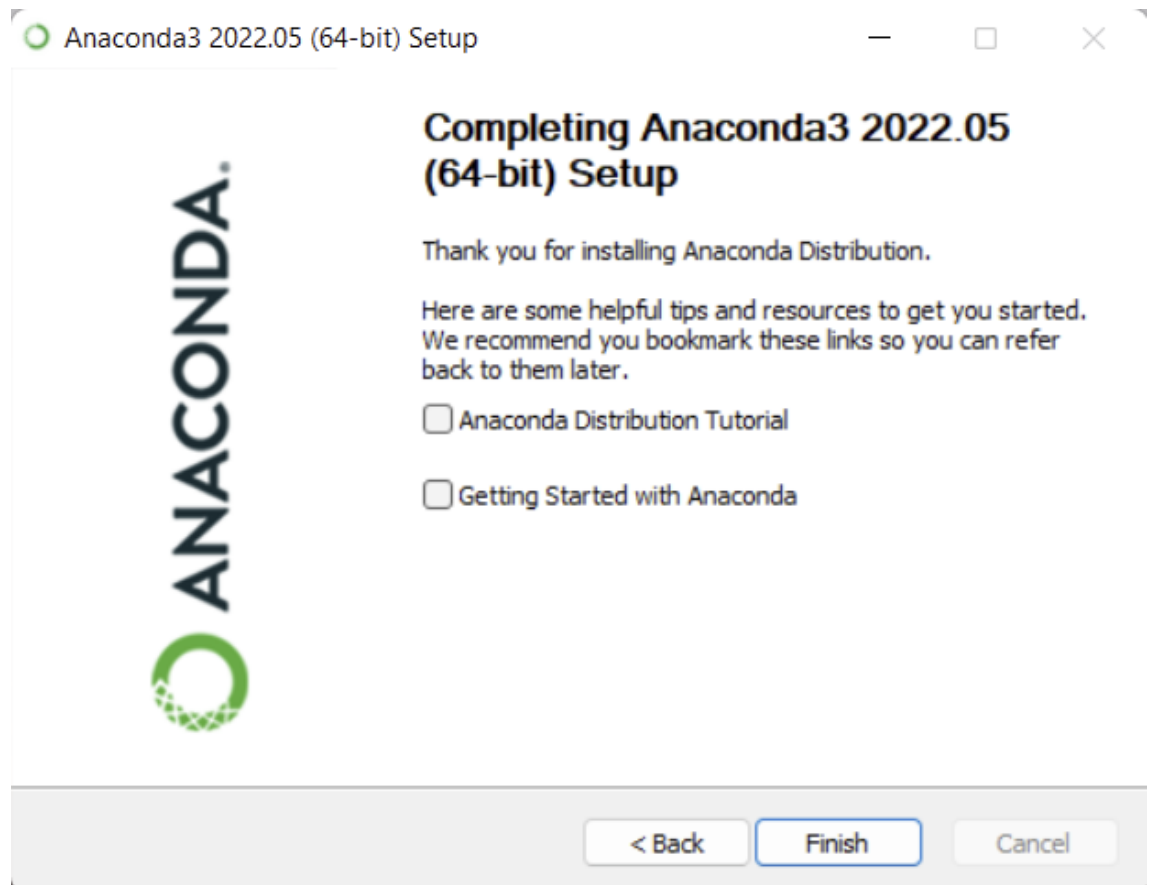
Presionamos “I Agree” para aceptar los términos de licenciamiento.



Aceptamos la carpeta de instalación (O la modificamos según nuestra necesidad) y presionamos en “Next”.

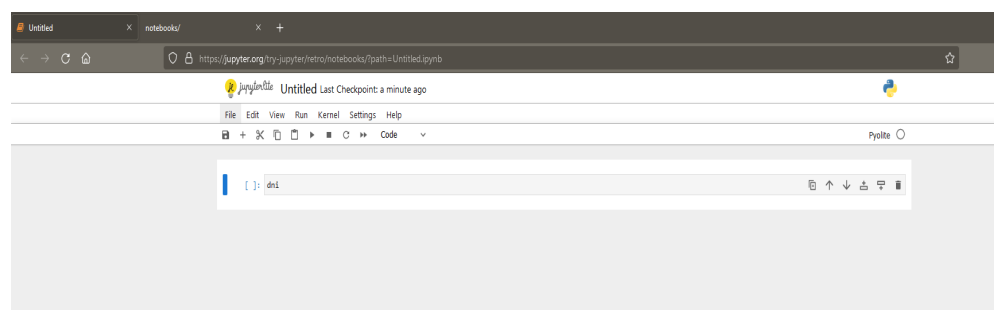


Dejamos las opciones por defecto y presionamos “Install”.



Una vez finalizada la instalación presionamos “Finish” y ya podemos acceder a trabajar con Python utilizando Jupyter Notebook.

Jupyter Notebook



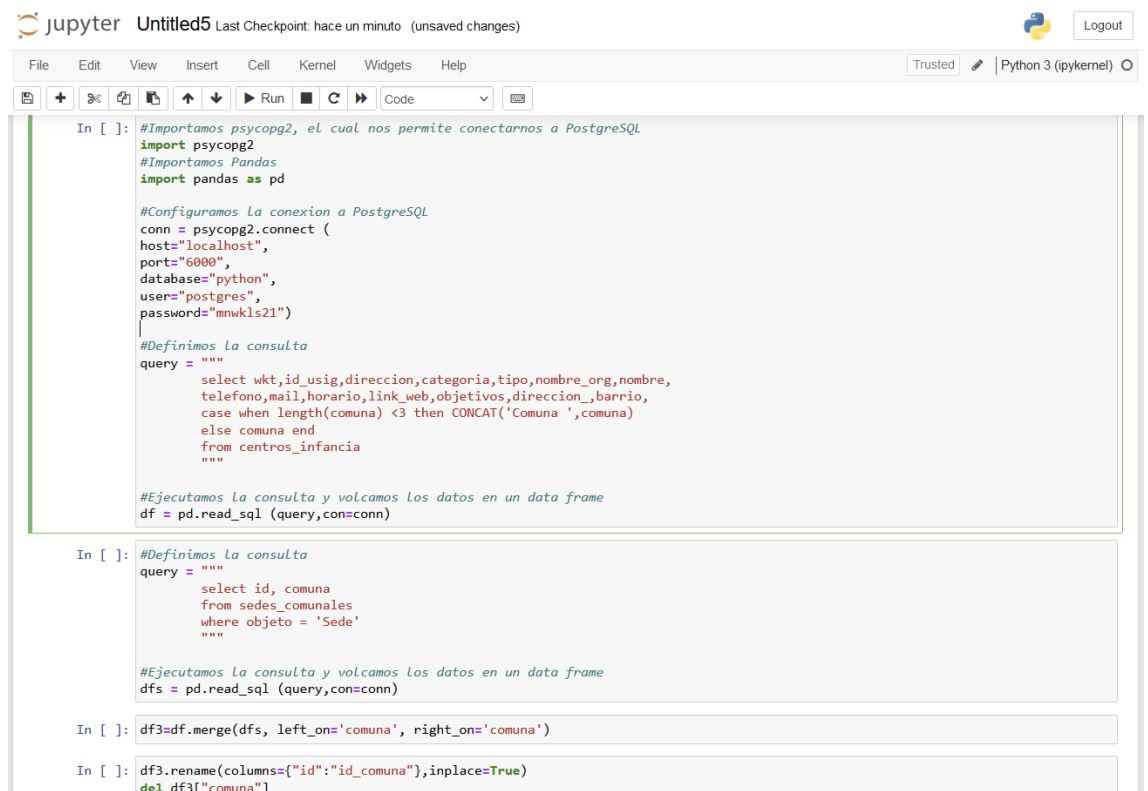
El **Proyecto Jupyter** es una organización sin ánimo de lucro creada para desarrollar software de código abierto, estándares abiertos y servicios para computación interactiva en docenas de lenguajes de programación".

Según la organización Jupyter, podemos definir a “Jupyter Notebook” como ***“una herramienta web que permite crear y compartir documentos, ofreciendo una experiencia simple y centrada en el documento.”***

En este sentido, podemos destacar las siguientes características:

- Soporta más de 40 lenguajes de desarrollo, entre los que podemos destacar ***Phyton, R, Julia y Scala.***
- Permite compartir los ***“Notebooks”*** generados a través de emails, cloud storage y GitHub.
- Los desarrollos creados pueden producir contenido interactivo, incorporando elementos como HTML, imágenes y videos.
- Facilita la ***integración con herramientas de Big Data*** como Apache Spark, accediendo desde Phyton, R o Scala. Permite acceder a los datos utilizando pandas, scikit-learn, ggplot2 y TensorFlow. Además, podemos destacar que Jupyter Notebook puede usarse de manera local (Realizando la instalación de la suite Anaconda) o en la nube.

Un documento de Jupyter Notebook es un documento que contiene una lista ordenada de celdas de entrada/salida las cuales pueden contener código, texto (usando Markdown), matemáticas, gráficos y texto enriquecidos, generalmente terminado con la extensión “.ipynb”.



```

jupyter Untitled5 Last Checkpoint: hace un minuto (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)
In [ ]: #Importamos psycopg2, el cual nos permite conectarnos a PostgreSQL
import psycopg2
#Importamos Pandas
import pandas as pd

#Configuramos la conexion a PostgreSQL
conn = psycopg2.connect (
    host="localhost",
    port="6000",
    database="python",
    user="postgres",
    password="mnwkl521")

#Definimos la consulta
query = """
    select wkt,id_usig,direccion,categoria,tipo,nombre_org,nombre,
    telefono,mail,horario,link_web,objetivos,direccion_barrio,
    case when length(comuna) <3 then CONCAT('Comuna ',comuna)
    else comuna end
    from centros_infancia
    """

#Ejecutamos la consulta y volcamos los datos en un data frame
df = pd.read_sql (query,conn)

In [ ]: #Definimos la consulta
query = """
    select id, comuna
    from sedes_comunales
    where objeto = 'Sede'
    """

#Ejecutamos la consulta y volcamos los datos en un data frame
dfs = pd.read_sql (query,conn)

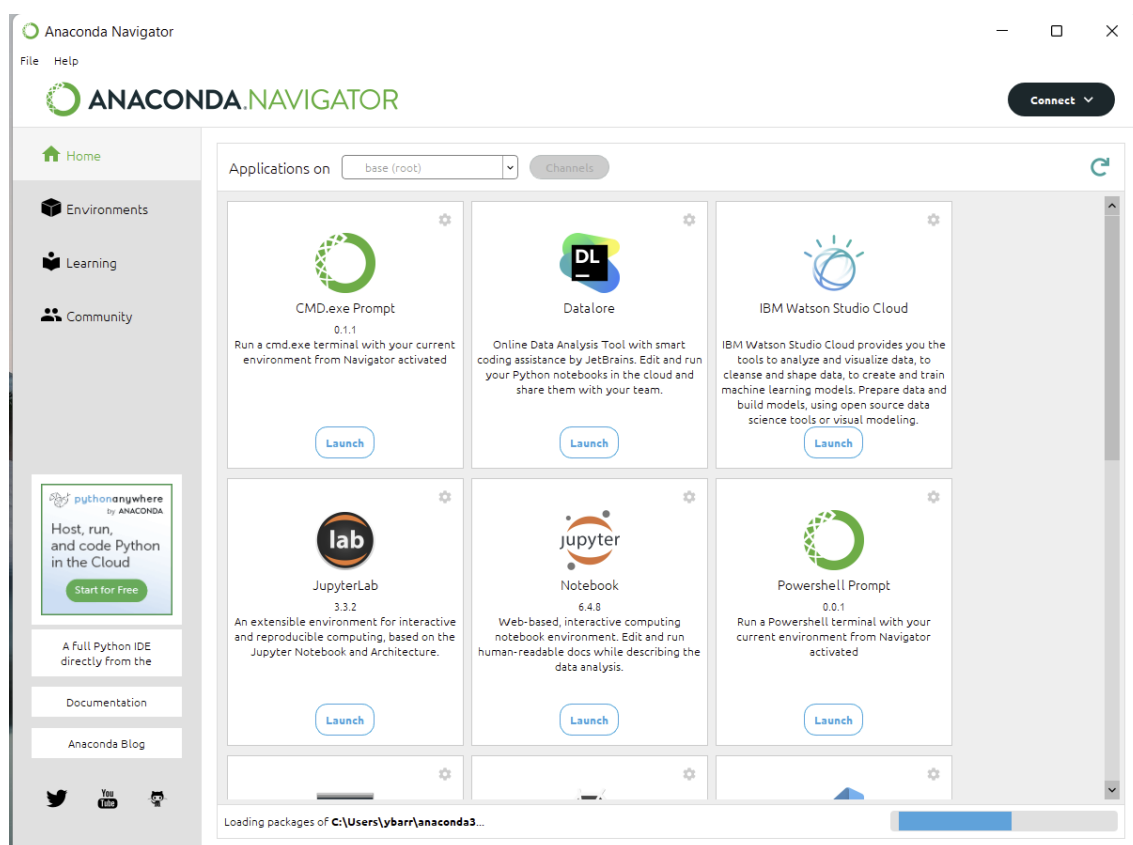
In [ ]: df3=df.merge(dfs, left_on='comuna', right_on='comuna')

In [ ]: df3.rename(columns={"id":"id_comuna"},inplace=True)
del df3["comuna"]
  
```

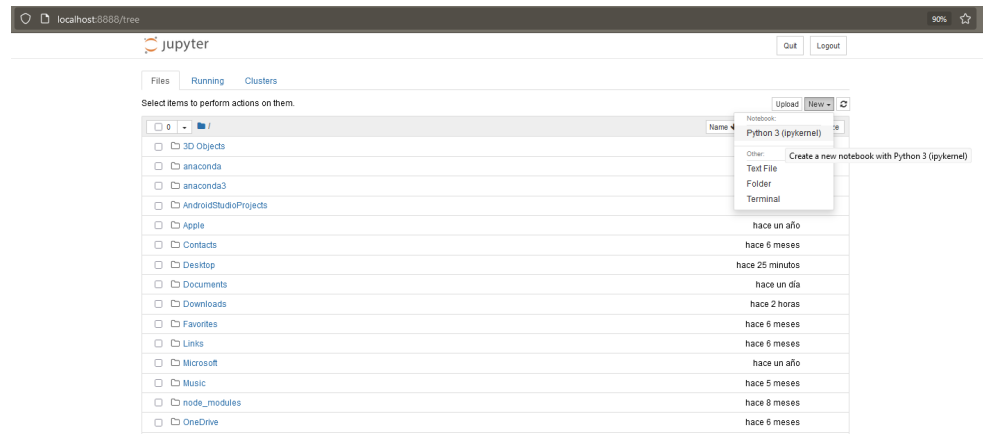
Imagen ejemplo de Notebook

Ejecutando Jupyter Notebook

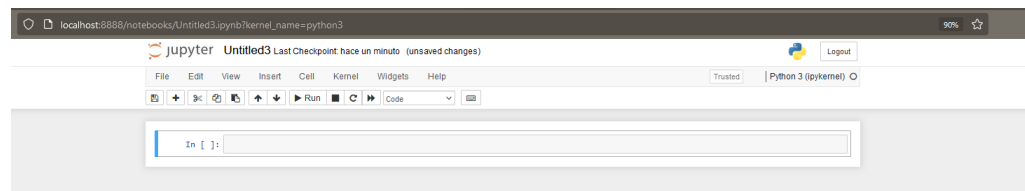
Para este módulo, desde Windows seleccionamos Anaconda Navigator. Nos aparecerá la siguiente pantalla:



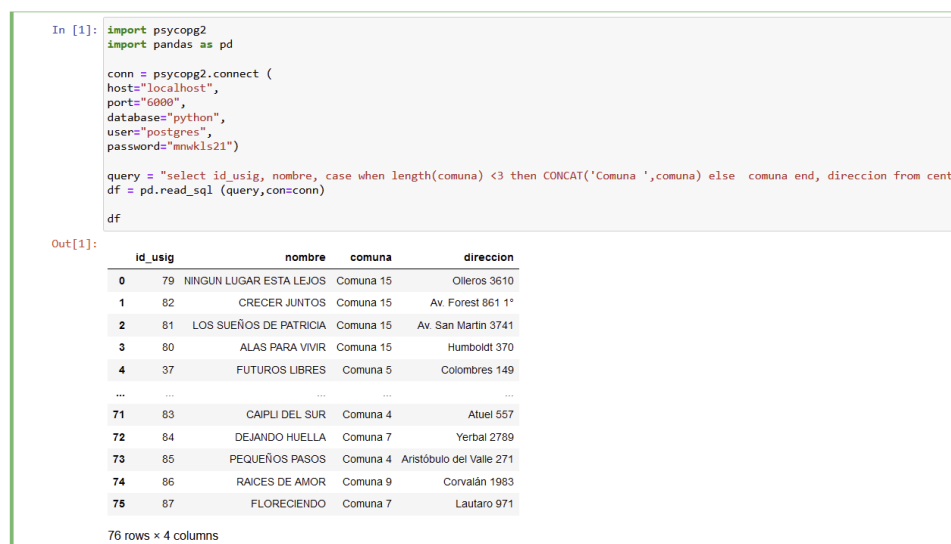
Seleccionamos Jupyter Notebook y presionamos “Launch”. Accedemos a la siguiente pantalla en nuestro navegador Web, donde presionaremos el botón “New” y seleccionaremos la opción “Python 3”.



Luego, jupyter nos mostrará la siguiente pantalla, donde podremos comenzar a trabajar con Jupyter Notebook.



En cada una de las celdas que nos mostrará la herramienta, iremos ingresando las sentencias de Python de nuestro programa en la celda “In”. Para ejecutar las mismas presionamos el botón “run”. Jupyter Notebook nos mostrará el resultado de la sentencia en la celda “Out”. Vemos en un ejemplo en la siguiente pantalla:



Una vez terminado nuestro programa, podemos guardar el mismo desde la opción “Archivo” y “Guardar Como”. Además tendremos la opción de descargar lo desarrollado, en diferentes formatos como: Python, Notebook, html, pdf, markdown, etc.

Por otro lado, presionando en la opción de menú **“help”**, aparecerán opciones que nos permitirán, por ejemplo, tomar un rápido tour interactivo sobre cómo utilizar el **notebook**. Además de ayuda sobre **Jupyter**, aparecerán links a recursos de paquetes clave.

Tema 2. Módulos y Paquetes en Python

Objetivo

El objetivo de este tema es brindar conocimientos vinculados a los conceptos de módulos, paquetes y Distribution Package en Python.

Una vez finalizado este tema serás capaz de:

- Conocer las conceptos básicos de los módulos
- Conocer las diferencias entre paquete y distribution package
- Importar paquetes en proyectos Python
- Actualizar paquetes

Módulos

Los **módulos** son archivos de Python donde se **almacenan** instrucciones, variables y/o definiciones, en archivos de extensión **.py** y en donde el nombre de este archivo se corresponde con el nombre del módulo. Los módulos son uno de los principales niveles de abstracción, ya que permiten separar el código en partes, manteniendo las funcionalidades y datos relacionados.

Una de las ventajas de trabajar con módulos es que nos permite dividir un programa extenso y con muchas líneas de código, en otros más pequeños. Otra ventaja es que, además, permite la reutilización de los mismos en otros programas o módulos. Entonces podemos decir que los módulos son la unidad básica de código reutilizable de Python.

En líneas generales, Python ofrece una serie de módulos estándar que podrán utilizarse como base para nuestros programas. Para ello, solo debemos importarlos. Veremos esto más adelante.

Paquetes en Phyton (Import Package)

De la misma forma en que agrupamos sentencias y definiciones en **Módulos**, los **Paquetes** nos permiten organizar de forma jerárquica el conjunto de módulos que componen nuestra aplicación. Una ventaja de trabajar con paquetes es que nos permiten la posibilidad de utilizar módulos con el mismo nombre sin que existan errores.

Entonces, los módulos se organizan en **Paquetes**, los cuales son folders que contienen **otros paquetes y/o un conjunto de módulos**. En este sentido, en Phyton, para que un directorio sea considerado un paquete, el mismo debe incluir un módulo llamado **_init_.py**, el cual en la mayoría de los casos, estará vacío. Algo a tener en cuenta es que el módulo **_init_.py** no siempre estará en blanco, sino que, en algunos casos, podría incluir el código necesario para inicializar el paquete.

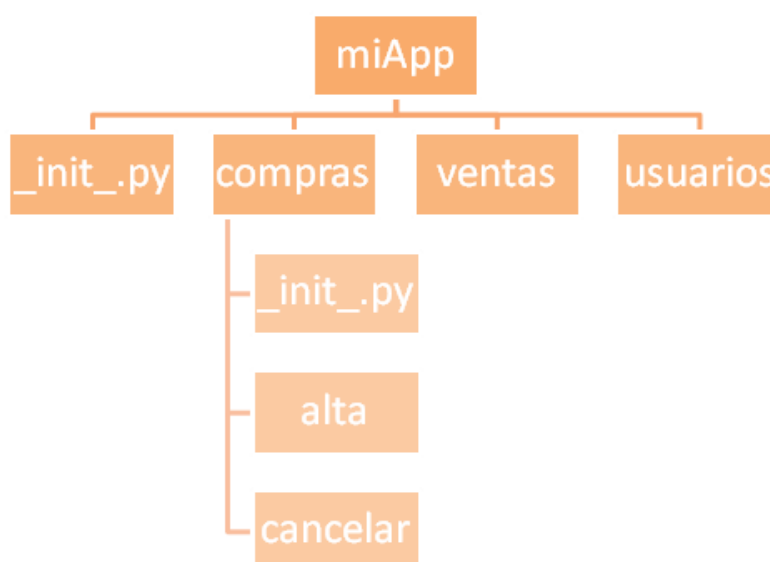


Imagen1: Ejemplo estructura jerárquica

Importar Paquetes

Para importar módulos y definiciones de los mismos que se encuentran dentro de Paquetes, en Phyton usaremos el operador **"."**. Las referencias se realizan indicando el nombre completo del módulo, detallando los paquetes para llegar al mismo, separándolos con **"."**.

De acuerdo a la "Imagen 1", si queremos importar el módulo "altas", ejecutaremos el siguiente comando en Python:

```
In [ ]: from miApp.ventas import alta
```

Incluso, se puede importar una definición de un módulo específico de la siguiente forma:

```
In [ ]: from miApp.ventas.alta import nuevaVenta
        nuevaVenta (cliente)
```

En este sentido, también podemos hacer referencia a módulos pero de forma relativa. Para esto escribimos:

```
In [ ]: from . import alta
```

Siendo que un “.” refiere al paquete actual. O bien:

```
In [ ]: from .. import cancelar
```

Distribution Package

Cabe destacar que existen otros tipos de paquetes a los cuales, en este documento llamaremos “Distribution Package”. Los mismos son archivos versionados que contienen módulos de Python, paquetes y otros recursos que son usados para ser distribuidos a través de una Release. Entendemos por Release a un Snapshot de un proyecto específico en un punto del tiempo determinado y denotado, en general, con un número de versión. Comúnmente, estos archivos son los que un usuario final descarga de internet para posteriormente instalarlos.

Finalmente, podemos mencionar algunos de los distribution packages más relevantes, como por ejemplo:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- Requests
- urllib3

Instalación de Distribution Packages

Antes de comenzar la instalación de un paquete específico, verificaremos que contamos con la herramienta para poder llevar adelante las instalaciones. Desde un Notebook ejecutaremos el siguiente comando:

```
In [1]: pip --version
pip 21.2.4 from C:\Users\ybarr\anaconda\anaconda3\lib\site-packages\pip (python 3.9)
Note: you may need to restart the kernel to use updated packages.
```

En este caso verificamos que tenemos “pip” (Instalador de Paquetes de Python) instalado.

La estructura del comando para realizar la instalación de **la última versión** de un paquete es la siguiente:

pip install nombre_paquete

Entonces para instalar un paquete específico, por ejemplo Matplotlib, ejecutaremos el siguiente comando:

```
In [2]: pip install Matplotlib
Requirement already satisfied: Matplotlib in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (3.4.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from Matplotlib) (3.0.4)
Requirement already satisfied: cycler>=0.10 in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from Matplotlib) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from Matplotlib) (8.4.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from Matplotlib) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from Matplotlib) (1.3.1)
Requirement already satisfied: numpy>=1.16 in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from Matplotlib) (1.20.3)
Requirement already satisfied: six in c:\users\ybarr\anaconda\anaconda3\lib\site-packages (from cycler>=0.10->Matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

En la pantalla anterior verificamos como Jupyter Notebook nos informa acerca de la correcta instalación del paquete Matplotlib.

Por otro lado, la estructura del comando para realizar la instalación de una versión específica de un paquete es la siguiente:

pip install nombre_paquete == numero_version

En este sentido, la estructura del comando para realizar la instalación de una versión mayor a X pero menor a Y de un paquete, es la siguiente:

pip install nombre_paquete >= numero_version_X, < numero_version_Y

La estructura del comando para realizar la instalación de una URL específica será la siguiente:

pip install --index-url <http://repositorio/ejemplo/paquete>

En cambio, La estructura del comando para realizar la instalación desde un archivo local, será la siguiente:

pip install -e ./descargas/paquete

Actualización de Paquetes

En este caso, de ser necesario una actualización de algún paquete específico anteriormente instalado y para el cual necesitamos un upgrade a la última versión, también trabajaremos con Pip.

La estructura del comando es la siguiente: ***pip install --upgrade nombre_paquete***



Tema 3. Módulo “venv”

Objetivo

El objetivo de este tema es brindar conocimientos vinculados a la herramienta Virtualenv, utilizada para crear un entorno virtual de Python, aislado de otros entornos virtuales.

Una vez finalizado este tema serás capaz de:

- Conocer las conceptos básicos del módulo “venv”
- Conocer el concepto de entorno virtual
- Crear entornos virtuales en proyectos python.
- Desactivar entornos virtuales
- Instalar paquetes y dependencias en entornos virtuales.

“venv” y entorno virtual, concepto

El módulo “venv” permite la creación de «entornos virtuales» con sus propios directorios de ubicación, aislados opcionalmente de los directorios de ubicación del sistema. Cada entorno virtual tiene su propio binario Python (que coincide con la versión del binario que se utilizó para crear este entorno) y puede tener su propio conjunto

independiente de paquetes Python instalados en sus directorios de ubicación.

La ventaja de trabajar con entornos virtuales radica en:

- ☐ Es posible tener varios entornos, con diferentes conjuntos de paquetes, sin conflictos entre ellos. De esta forma, los requerimientos de distintos proyectos se pueden satisfacer al mismo tiempo.
- ☐ Es posible lanzar fácilmente un proyecto con sus propios módulos dependientes.

Instalación

El módulo “venv” puede instalarse, desde la línea de comandos, con el siguiente comando:

```
python -m pip install virtualenv
```

Luego, su correcta instalación puede verificarse con el siguiente comando:

```
python -m virtualenv --version
```

Crear un entorno virtual

Una vez verificado que “venv” ha sido correctamente instalado, es posible crear un entorno virtual con el siguiente comando:

```
python -m venv c:\Users\ybarr\Desktop\Python\my-ambiente-1
```

Se debe reemplazar “c:\Users\ybarr\Desktop\Python” por la ruta donde se creará el ambiente.

Se debe reemplazar “my-ambiente-1” por el nombre que se desea dar al ambiente nuevo.

Nota: Es recomendable que la carpeta para el entorno virtual sea una subcarpeta del proyecto Python al que está asociado.

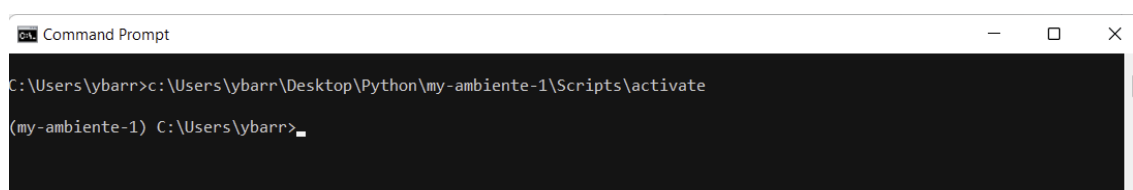
Desde el navegador de windows podemos ver la estructura de carpetas dentro del ambiente creado.

Python > my-ambiente-1				
Name	^	Date modified	Type	Size
Include		15/9/2022 20:47	File folder	
Lib		15/9/2022 20:47	File folder	
Scripts		15/9/2022 20:48	File folder	
pyenv		15/9/2022 20:47	Configuration Sou...	1 KB

Con el siguiente comando activaremos el entorno virtual:

`c:\Users\ybarr\Desktop\Python\my-ambiente-1\Scripts\activate`

Luego, la consola de windows nos mostrará lo siguiente:



```

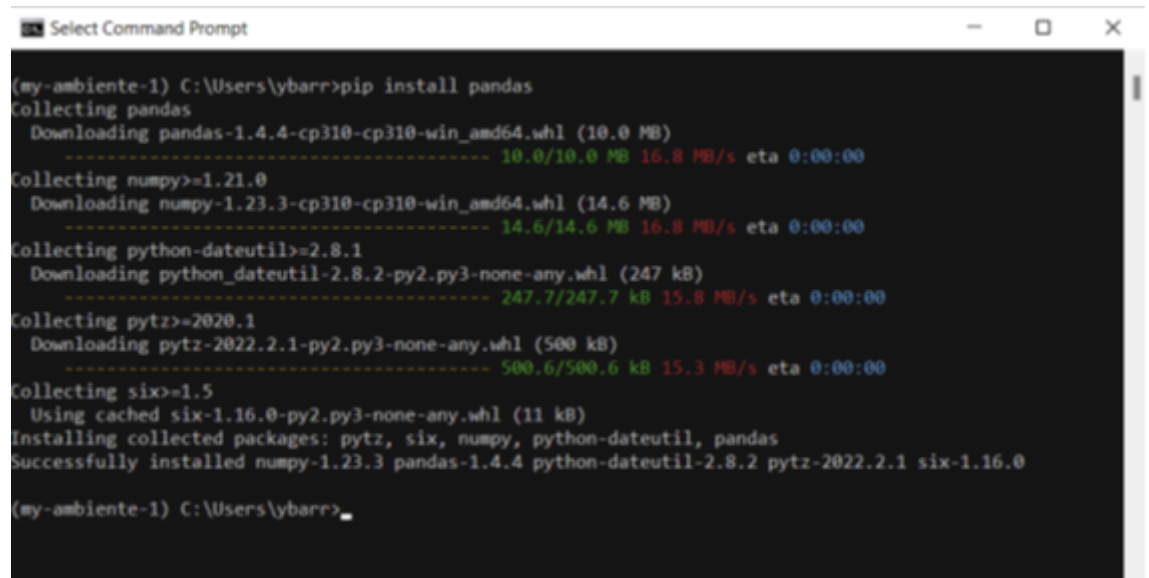
C:\Users\ybarr>c:\Users\ybarr\Desktop\Python\my-ambiente-1\Scripts\activate
(my-ambiente-1) C:\Users\ybarr>
  
```

De esta forma, ya estaremos trabajando en nuestro ambiente virtual. Nos damos cuenta porque el nombre de nuestro ambiente virtual aparece delante del prompt.

Ahora ejecutamos el siguiente comando, para instalar la última versión del paquete pandas en nuestro ambiente virtual.

`pip install pandas`

Al finalizar, visualizamos lo siguiente:



```
(my-ambiente-1) C:\Users\ybarr>pip install pandas
Collecting pandas
  Downloading pandas-1.4.4-cp310-cp310-win_amd64.whl (10.0 MB)
    ----- 10.0/10.0 MB 16.8 MB/s eta 0:00:00
Collecting numpy>=1.21.0
  Downloading numpy-1.23.3-cp310-cp310-win_amd64.whl (14.6 MB)
    ----- 14.6/14.6 MB 16.8 MB/s eta 0:00:00
Collecting python-dateutil>=2.8.1
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
    ----- 247.7/247.7 kB 15.8 MB/s eta 0:00:00
Collecting pytz>=2020.1
  Downloading pytz-2022.2.1-py2.py3-none-any.whl (500 kB)
    ----- 500.6/500.6 kB 15.3 MB/s eta 0:00:00
Collecting six>=1.5
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, six, numpy, python-dateutil, pandas
Successfully installed numpy-1.23.3 pandas-1.4.4 python-dateutil-2.8.2 pytz-2022.2.1 six-1.16.0

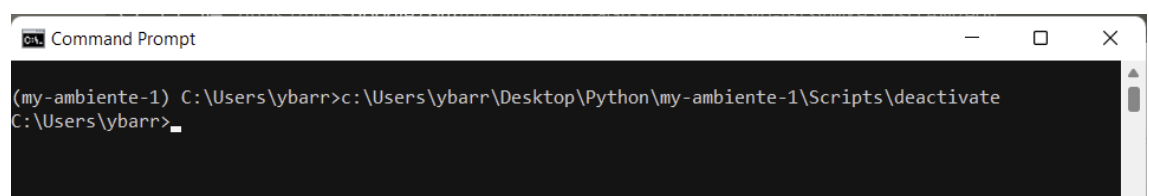
(my-ambiente-1) C:\Users\ybarr>
```

Desactivar entorno virtual

Para desactivar nuestro entorno virtual ejecutamos el siguiente comando:

c:\Users\ybarr\Desktop\Python\my-ambiente-1\Scripts\deactivate

Verificamos que el ambiente se ha desactivado y ya no estamos trabajando dentro del mismo. El prompt del SO vuelve a visualizarse normalmente.



```
(my-ambiente-1) C:\Users\ybarr>c:\Users\ybarr\Desktop\Python\my-ambiente-1\Scripts\deactivate
C:\Users\ybarr>
```

Podemos ahora crear un nuevo ambiente virtual, activarlo, y allí instalar una versión anterior a la última versión de Pandas. Ejecutamos los comandos:

python -m venv c:\Users\ybarr\Desktop\Python\my-ambiente-2

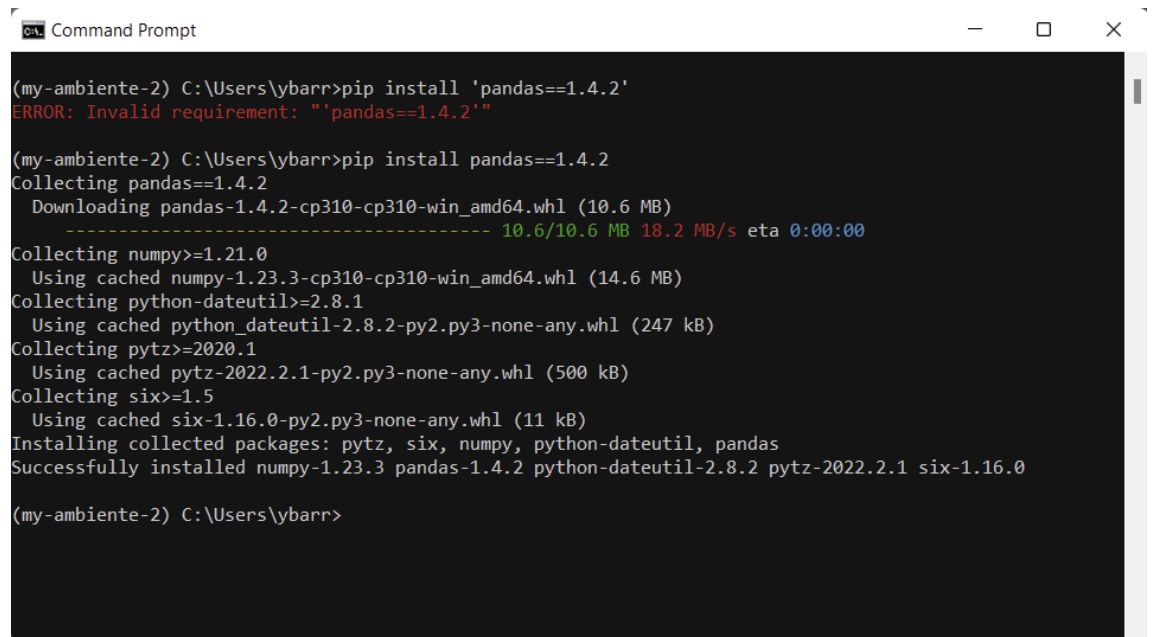
y luego:

c:\Users\ybarr\Desktop\Python\my-ambiente-2\Scripts\activate

Desde nuestro nuevo ambiente ejecutamos:

(my-ambiente-2) C:\Users\ybarr>pip install 'pandas==1.4.2'

Se procederá a la instalación de la versión 1.4.2 de Pandas. El sistema mostrará lo siguiente:

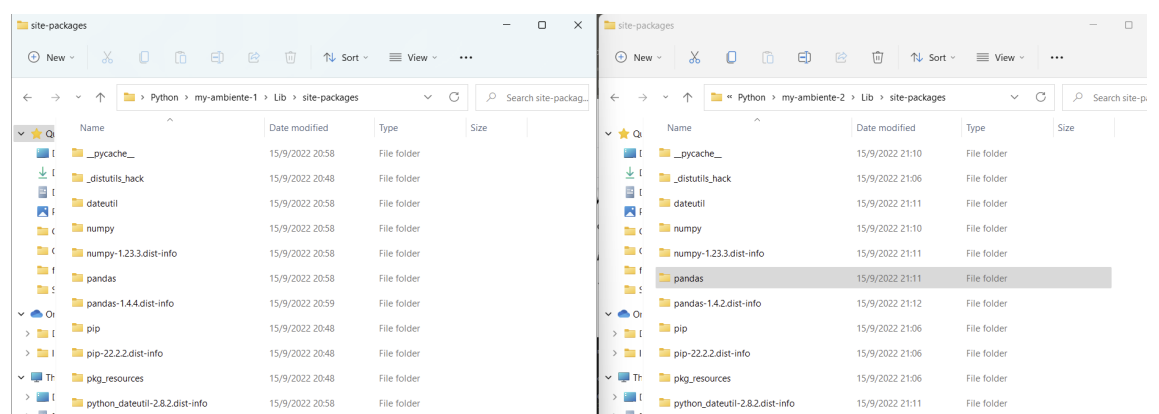


```
(my-ambiente-2) C:\Users\ybarr>pip install 'pandas==1.4.2'
ERROR: Invalid requirement: "'pandas==1.4.2'"

(my-ambiente-2) C:\Users\ybarr>pip install pandas==1.4.2
Collecting pandas==1.4.2
  Downloading pandas-1.4.2-cp310-cp310-win_amd64.whl (10.6 MB)
----- 10.6/10.6 MB 18.2 MB/s eta 0:00:00
Collecting numpy>=1.21.0
  Using cached numpy-1.23.3-cp310-cp310-win_amd64.whl (14.6 MB)
Collecting python-dateutil>=2.8.1
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting pytz>=2020.1
  Using cached pytz-2022.2.1-py2.py3-none-any.whl (500 kB)
Collecting six>=1.5
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, six, numpy, python-dateutil, pandas
Successfully installed numpy-1.23.3 pandas-1.4.2 python-dateutil-2.8.2 pytz-2022.2.1 six-1.16.0

(my-ambiente-2) C:\Users\ybarr>
```

Luego de la ejecución de este comando, verificamos en el explorador de windows como ha quedado la estructura de carpetas y archivo en ambos ambientes en el explorador de windows:



De esta forma podemos verificar que podemos mantener versiones diferentes de paquetes y dependencias en cada entorno virtual, de forma aislada uno del otro.



Tema 4. Archivo “requirements.txt”

Objetivo

El objetivo de este tema es brindar conocimientos vinculados a la utilización del archivo “requirements.txt” .

Una vez finalizado este tema serás capaz de:

- Conocer las conceptos básicos de la función del archivo
- Crear el archivo “Requirements.txt”
- Utilizar buenas prácticas sobre el mismo

Archivo “Requirements.txt”

En Python, “Requirement.txt” es un tipo de archivo (De texto) que generalmente almacena información sobre todas las bibliotecas, módulos y paquetes que se utilizan al desarrollar un proyecto en particular. También almacena todos los archivos y paquetes de los que ese proyecto depende o requiere para ejecutarse.

En general, la estrategia a la hora de desarrollar una aplicación es la de dividir la misma en componentes pequeños, funcionales y reutilizables. Como hemos explicado anteriormente, podemos estructurar nuestro código en módulos y paquetes, de forma jerárquica.

En este sentido, la estructura de directorios y archivos que debieran crearse en una carpeta de proyecto inicial y vacío, serán los siguientes:

1- LICENSE: Este archivo define el tipo de licencia del proyecto. Por lo tanto, contendrá información legal vinculada a las pautas de uso de la aplicación y los detalles de distribución de la misma. Puede, también, no definirse un tipo de licenciamiento.

2- README.md: Este archivo es meramente descriptivo, conteniendo información vinculada al detalle de la aplicación, sus archivos, como configurar la misma y su ejecución. Además se puede profundizar en pautas de contribución así como también información respecto a si la aplicación es o no de código abierto.

Se sitúa normalmente en el directorio raíz de la aplicación. Además, se puede escribir en casi cualquier formato de texto, pero en desarrollo normalmente se usa Markdown.

3- `setup.py` o `run.py`: Este archivo es el script de configuración, permitiendo la gestión de paquetes y librerías. Puede definirse como el punto de entrada a la aplicación. De este modo, siempre que queramos ejecutar un aplicación python con una estructura de proyecto similar, debemos ejecutar este archivo.

4- **requirements.txt**: Como hemos mencionado anteriormente, en este archivo se define toda la información vinculada a las dependencias de la aplicación. Detalla, además, las versiones específicas de cada dependencia.

Este archivo, dentro de nuestro proyecto, podemos generarlo con el siguiente comando:

pip freeze > requirements.txt

Una vez generado el mismo, con toda la lista de dependencias, desde una nueva PC o entorno virtual, podremos ejecutar:

pip install -r requirements.txt

Este comando le indicará a Pip que debe instalar las versiones de los paquetes especificados en el archivo `requirements.txt`. Siempre asumiendo que estamos trabajando en el directorio donde hemos copiado nuestro archivo `requirements.txt`

5- `/Paquetes y/o módulos`: En esta carpeta contendrá nuestra aplicación propiamente dicha, con sus módulos y paquetes.

6- `/docs`: En esta carpeta se almacenará la documentación propiamente dicha de la aplicación.

7- `/tests`: En esta carpeta estarán alojados los archivos necesarios para llevar adelante los tests de la aplicación.

8- `/static`: En esta carpeta estarán almacenados todos los recursos estáticos de la aplicación, como audios, videos, imágenes, etc

9- `database.db`: Este archivo contiene el código para realizar la conexión a una base de datos (De utilizar alguna).

10- `templates`: En esta carpeta contiene las plantillas HTML del proyecto.

Se debe tener en cuenta que, dependiendo de las particularidades del proyecto, quizás alguno de estos archivos/directorios no sea necesario. Por

ejemplo: Si no se va a trabajar con ninguna base de datos, no será necesario el archivo `database.db`.

Buenas Prácticas

Existen algunas buenas prácticas recomendadas que se deben seguir al usar el archivo `requirements.txt` de Python:

- Utilizar siempre el comando `"pip freeze"` para generar una lista de módulos y paquetes de Python instalados en el entorno virtual del proyecto. Esto asegura que la lista esté actualizada y sea precisa.
- Enumerar solamente los módulos y paquetes de Python que se necesitan. No se deben incluir módulos o paquetes innecesarios, ya que esto hace que el archivo crezca innecesariamente y sea difícil de leer. También es un desperdicio de recursos.
- Guardar el archivo `"requirements.txt"` en el repositorio del código fuente del proyecto para que otros desarrolladores puedan instalar fácilmente todos los módulos y paquetes de Python cuando clonen el proyecto.
- Usar el comando `"pip install -r requirements.txt"` para instalar todos los módulos y paquetes de Python enumerados en el archivo `requirements.txt`. Esto ahorra tiempo y esfuerzo.
- Mantener el archivo `requirements.txt` actualizados. Esto garantiza que el proyecto siempre use las últimas versiones de los módulos y paquetes de Python.



Referencias

<https://jupyter.org/>

<https://elpythonista.com/anaconda>

https://es.wikipedia.org/wiki/Proyecto_Jupyter

<https://docs.python.org/3/tutorial/modules.html>

<https://j2logo.com/python/tutorial/espacios-de-nombres-modulos-y-paquetes/>

<https://packaging.python.org/en/latest/tutorials/installing-packages/>

<https://www.delftstack.com/es/howto/python/python-project-structure/>

<https://docs.python.org/es/3/library/venv.html#>

<https://www.atlassian.com/es/git/tutorials/what-is-version-control>

<https://www.atlassian.com/es/git/tutorials/using-branches>

tiiiiiit by 