# Pyspark

<https://www.linkedin.com/learning/domina-python-pyspark/instalacion-de-pyspark>

<https://spark.apache.org/docs/latest/api/python/getting_started/index.html>

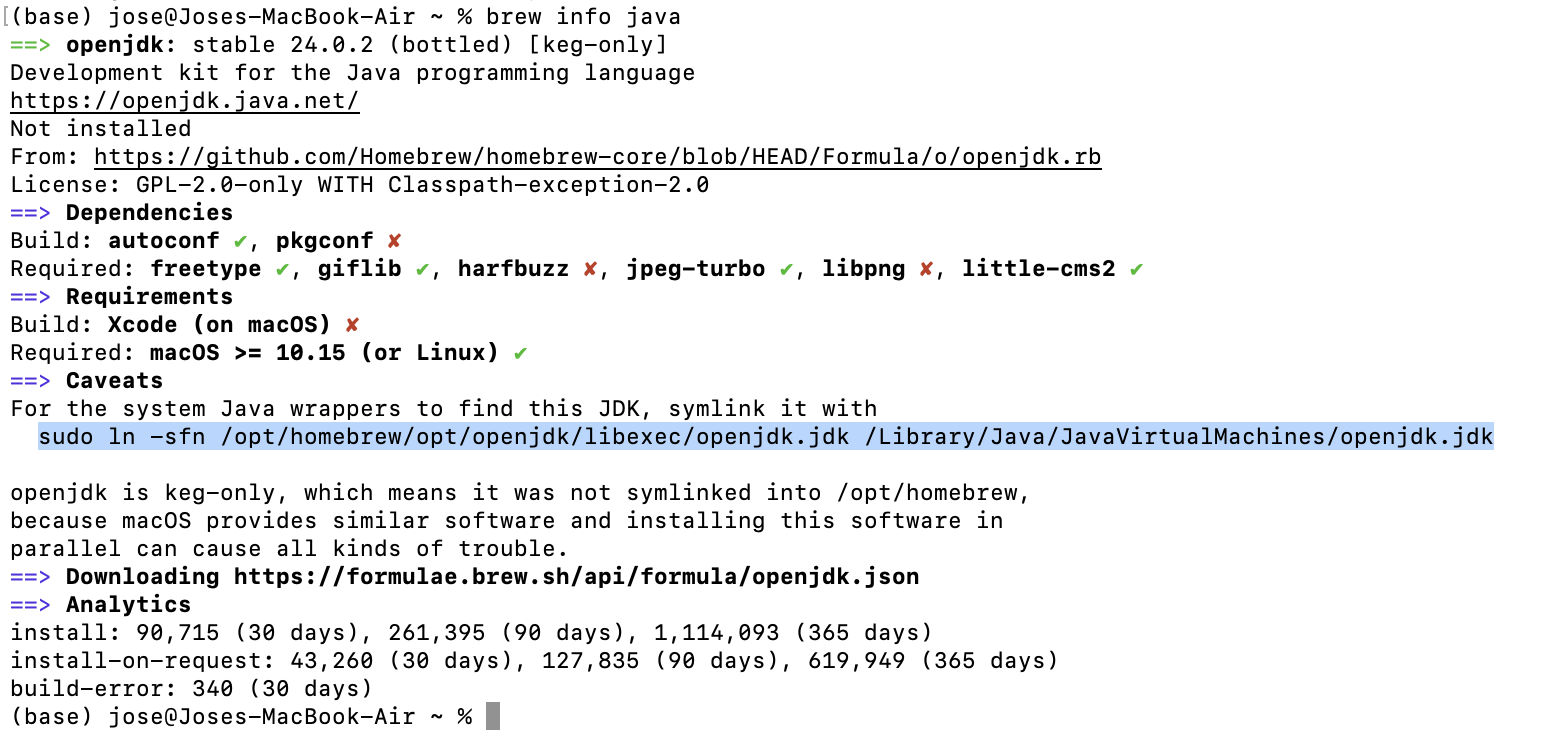
<https://github.com/Josesosa0777/pyspark_scipy>

## Instalación de Pyspark

Instalar JDK de Java, ejemplo: JDK 17

*brew install openjdk@17*

*brew info java*



Ejecutar el enlace que da:  
*sudo ln -sfn /opt/homebrew/opt/openjdk/libexec/openjdk.jdk /Library/Java/JavaVirtualMachines/openjdk.jdk*

Para ver la ruta donde se instaló JDK:

*brew --prefix openjdk*

abir el file .zshrc, yo lo hice con:

*nano ~/.zshrc*

agregar:

*export JAVA\_HOME=/opt/homebrew/opt/openjdk@17*

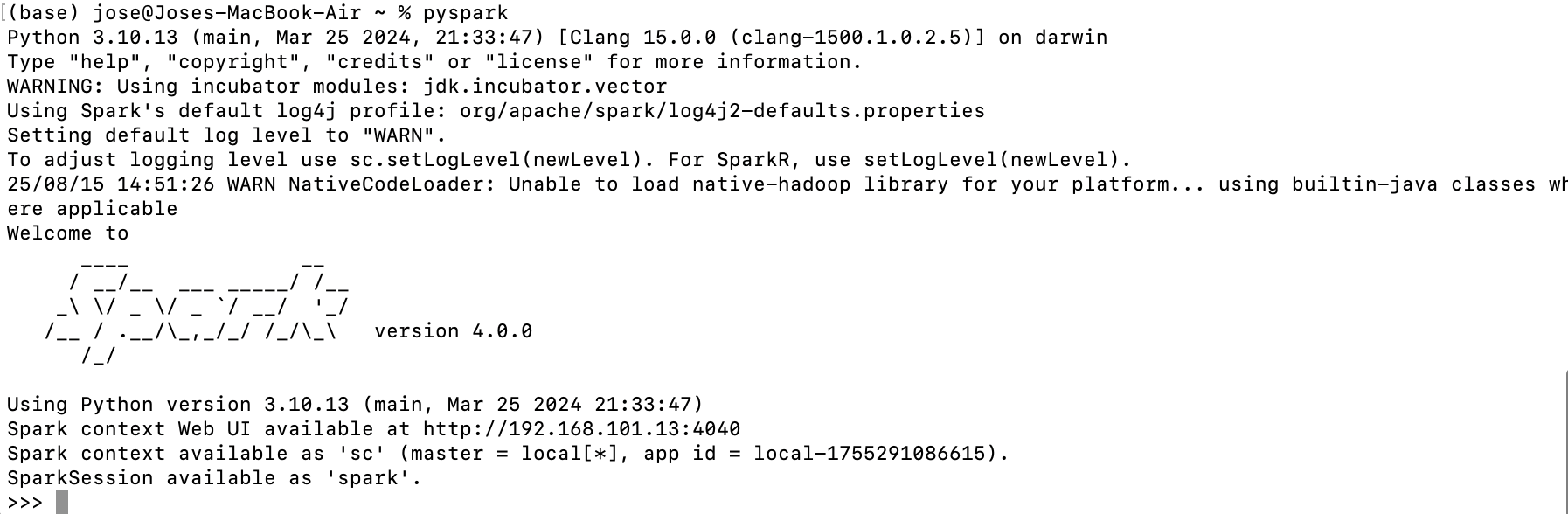
*export PATH=$JAVA\_HOME/bin:$PATH*

Reiniciar terminal, y ejecutar

*java -version*

*brew install apache-spark*

Una vez finalizado, si ejecuto *pyspark*, veré que se instaló. Para salir, ejecutar *exit()*



## Crear un sparkSession de pySpark

Inicio creando un folder con su virtual environment

*python3 -m venv venv*

*source venv/bin/activate*

*pip install pyspark*

crear app.py:

*from pyspark.sql import SparkSession*

*# Crear SparkSession*

*spark = SparkSession.builder \*

*.appName("MiAplicacionSpark") \*

*.getOrCreate()*

*print("SparkSession creada con éxito")*

.getOrCreate() es para que Si ya existe una SparkSession con la misma configuracion, la reutiliza en lugar de crear una nueva.

## [Crear un RDD (Resilient Distributed Datasets) a partir de una lista en PySpark](https://www.linkedin.com/learning/domina-python-pyspark/crear-un-rdd-resilient-distributed-datasets-a-partir-de-una-lista-en-pyspark?autoplay=true&resume=false)

La RDD son una estructura de datos básica en Apache Spark que nos permite manejar grandes conjuntos de datos.

spark\_context, en este caso, es el punto de entrada para trabajar con RDD en PySpark.

El método parallelize, es un método de spark\_context que toma una colección de datos en Python, en este caso, la lista de datos que tenemos, y la distribuye como un RDD en un clúster de Spark, y, seguidamente, vamos a hacer uso del método collect, que es una acción que recupera todos los elementos del RDD, desde los nodos distribuidos, y los devuelve como una lista.

Nota que el método collect debería usarse con precaución, ya que si el RDD es muy grande, puede sobrecargar la memoria.

El método stop, es un paso final esencial en cualquier aplicación o script de Spark, dado que nos garantiza que todos los recursos utilizados por la sesión se liberen correctamente.

Example:

from pyspark.sql import SparkSession

# Crear SparkSession

spark = SparkSession.builder \

.appName("MiAplicacionSpark") \

.getOrCreate()

print("SparkSession creada con éxito")

spark\_context = spark.sparkContext

datos = [1, 2, 3, 4, 5, 6]

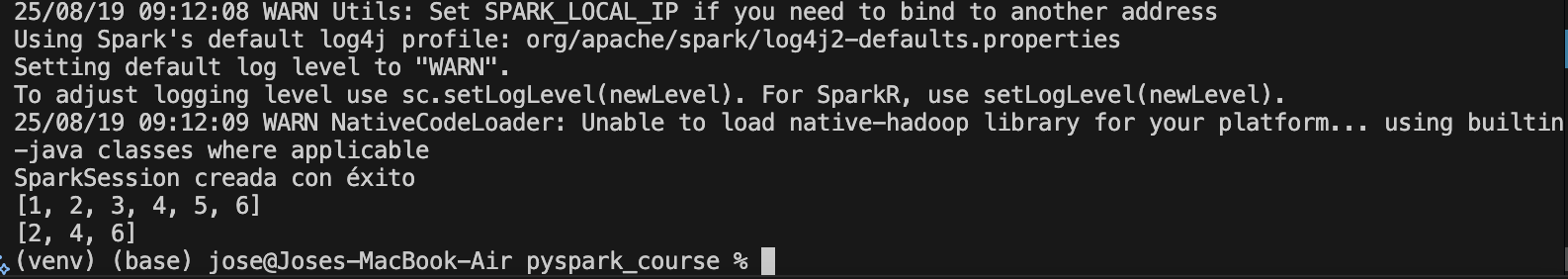
rdd = spark\_context.parallelize(datos)

print(rdd.collect())

num\_pares = rdd.filter(lambda x: x % 2 == 0)

print(num\_pares.collect())

spark.stop()



## Cargar un archivo en un RDD en PySpark

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

ruta\_archivo = "datos.csv"

rdd = spark\_context.textFile(ruta\_archivo)

if rdd.isEmpty():

print("El RDD está vacío. Verifica el contenido del archivo.")

else:

fila\_encabezado = rdd.first()

datos\_sin\_encabezado = rdd.filter(lambda fila: fila != fila\_encabezado)

datos\_sin\_encabezado = datos\_sin\_encabezado.map(lambda fila: fila.split(","))

print("Primeros 5 registros:")

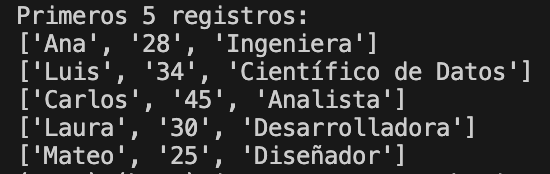
# ⚠️ Cuidado con .collect(). Si estás trabajando con archivos muy grandes (miles o millones de filas),

# puede consumir toda tu RAM y hacer que tu programa se bloquee.

for registro in datos\_sin\_encabezado.take(5): # .collect() trae todos los datos del RDD a la memoria local (tu computadora).

print(registro)

spark.stop()



# Transformaciones y acciones en pyspark

## ⚙️ Operaciones comunes

| **Operación** | **Qué hace** |
| --- | --- |
| rdd.map(func) | Aplica una función a cada fila |
| rdd.filter(func) | Filtra filas según una condición |
| rdd.first() | Devuelve la primera fila |
| rdd.take(5) | Devuelve las primeras 5 filas |
| rdd.collect() | Devuelve **todas** las filas como una lista (¡cuidado con archivos grandes!) |

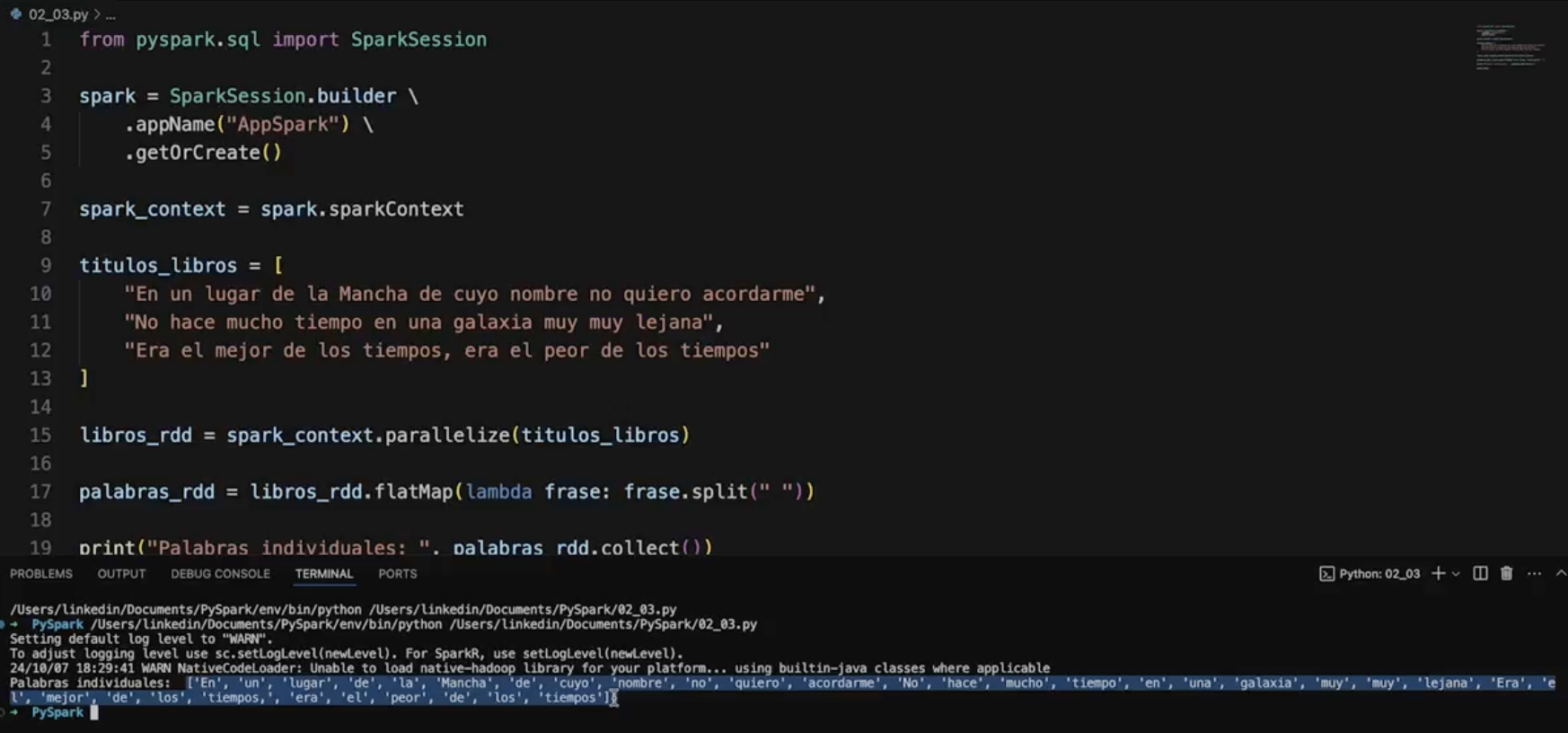
parallelize convierte **una lista de Python** (u otra colección) en un **RDD distribuido**.

Es una forma rápida de crear un RDD a partir de datos que ya tienes en memoria (como una lista), para procesarlos en paralelo con Spark.

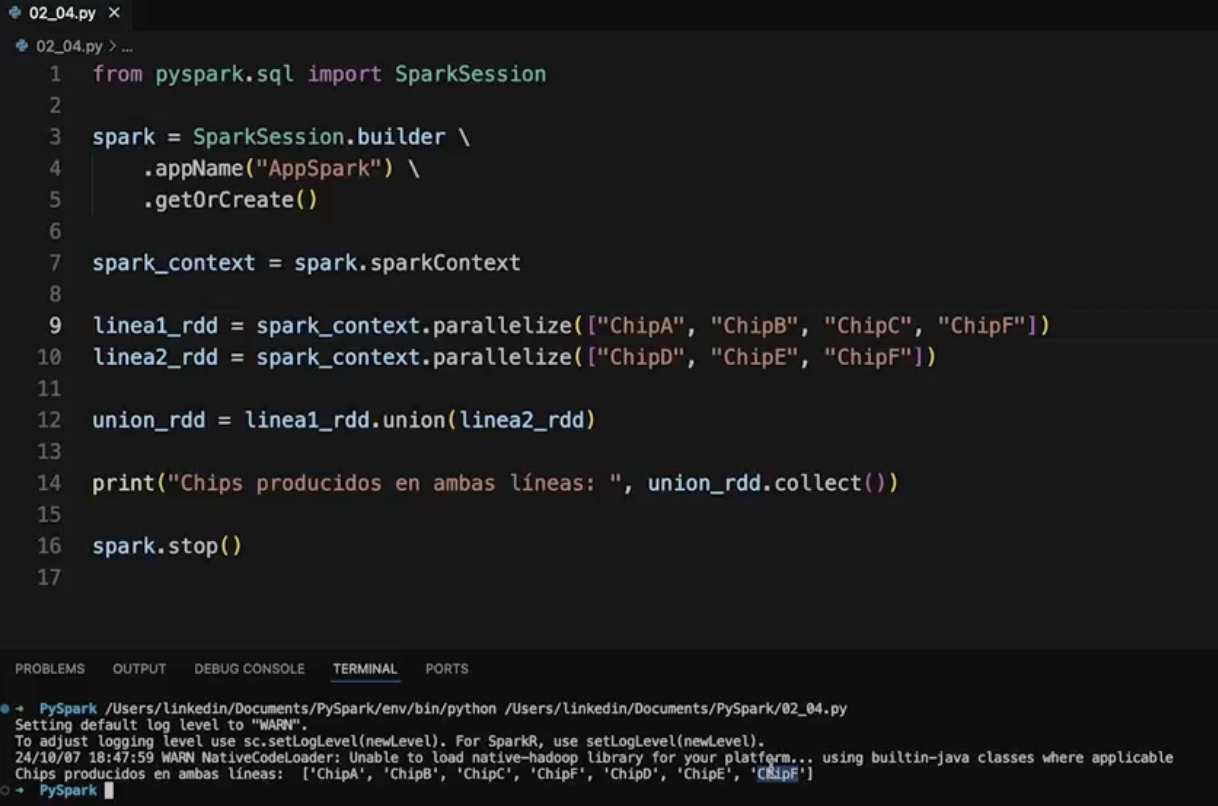
Puedes especificar cuántas particiones quieres:

rdd = spark.sparkContext.parallelize(datos, numSlices=3)

## **Transformaciones a un RDD de PySpark usando la función flatMap**



## **Transformaciones a un RDD de PySpark usando union**



Union une los rdd sin eliminar duplicados.

## **Transformaciones a un RDD de PySpark usando reduceByKey**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

transacciones\_rdd = spark\_context.parallelize([

("cliente\_01", 100),

("cliente\_02", 200),

("cliente\_01", 150),

("cliente\_03", 300),

("cliente\_02", 250),

("cliente\_03", 100)

])

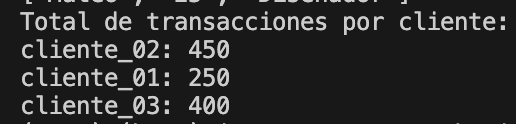
total\_por\_cliente\_rdd = transacciones\_rdd.reduceByKey(lambda x, y: x + y)

print("Total por cliente:")

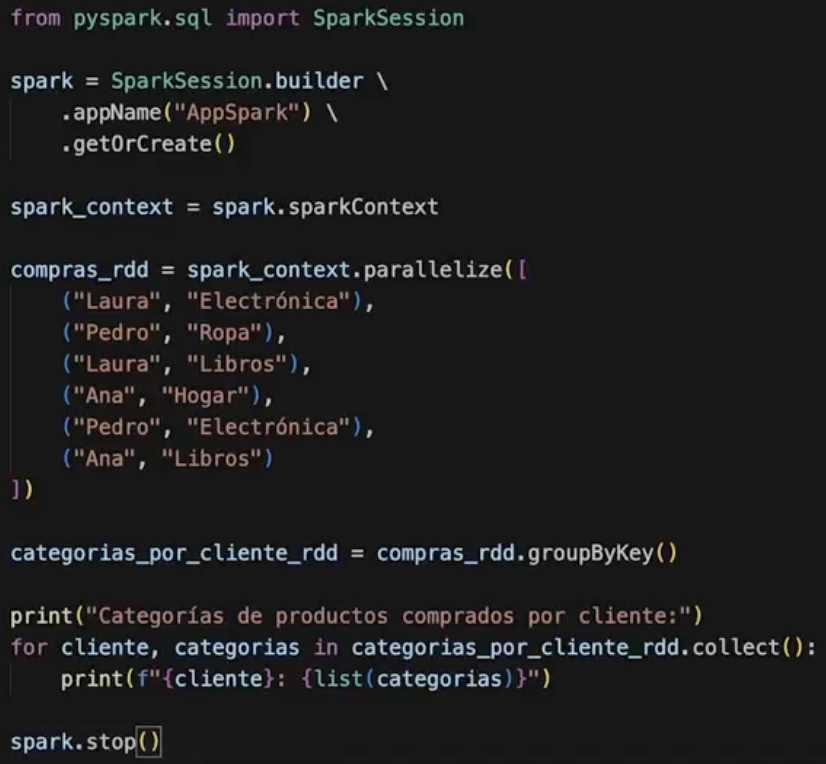
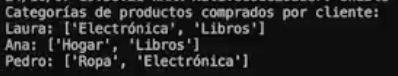
for cliente, total in total\_por\_cliente\_rdd.collect():

print(f"{cliente}: {total}")

spark.stop()



## **Transformaciones a un RDD de PySpark usando groupByKey**

## **Transformaciones a un RDD de PySpark usando sortByKey**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

envios\_rdd = spark\_context.parallelize([

("ZX123", "Envío a Cartago"),

("AB456", "Envío a Heredia"),

("MN789", "Envío a Alajuela"),

("CD234", "Envío a San José"),

("XY567", "Envío a Puntarenas")

])

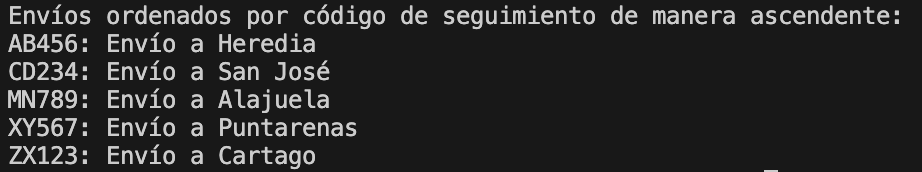
envios\_ordenados\_rdd = envios\_rdd.sortByKey()

print("Envíos ordenados por código de seguimiento de manera ascendente:")

for codigo, detalle in envios\_ordenados\_rdd.collect():

print(f"{codigo}: {detalle}")

spark.stop()



## **Transformaciones a un RDD de PySpark usando join**

Combina 2 conjuntos de datos basados en la clave compartida. Si alguno no tiene la clave en ambos, no lo considera.

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

clientes\_rdd = spark\_context.parallelize([

("C001", "Ana"),

("C002", "Carlos"),

("C003", "Lucía"),

("C004", "Pedro")

])

contratos\_rdd = spark\_context.parallelize([

("C001", "Contrato Luz"),

("C002", "Contrato Televisión"),

("C003", "Contrato Internet"),

("C005", "Contrato Telefonía")

])

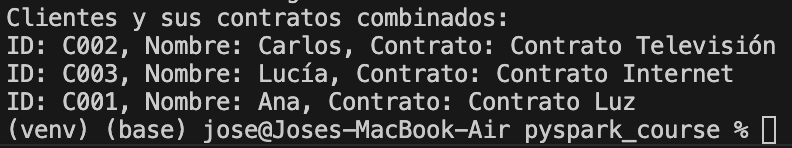
clientes\_contratos\_rdd = clientes\_rdd.join(contratos\_rdd)

print("Clientes y sus contratos combinados:")

for cliente\_id, (nombre, contrato) in clientes\_contratos\_rdd.collect():

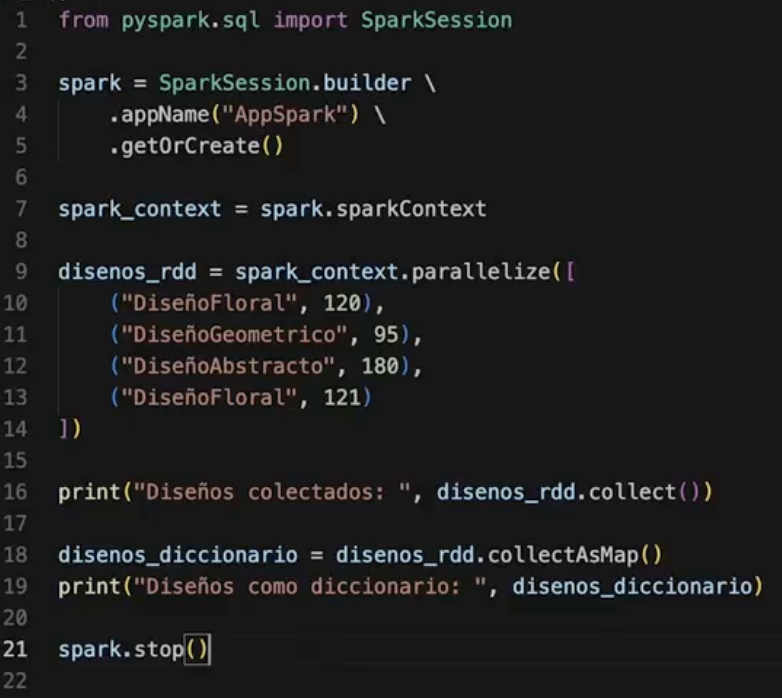
print(f"ID: {cliente\_id}, Nombre: {nombre}, Contrato: {contrato}")

spark.stop()



## **Acciones en RDD de PySpark usando collect, o collectAsMap**

collectAsMap() devuelve un rdd de pares clave-valor a manera de un dict

## **Acciones en RDD de PySpark usando first, count, countByKey**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

observaciones\_rdd = spark\_context. parallelize([

("Eclipse", 90),

("Cometa", 70),

("Eclipse", 85),

("Supernova", 95),

("Cometa", 75)

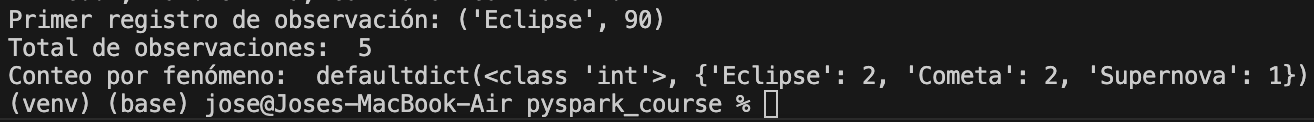
])

print("Primer registro de observación:", observaciones\_rdd.first())

print("Total de observaciones: ", observaciones\_rdd.count())

print("Conteo por fenómeno: ", observaciones\_rdd.countByKey())

spark.stop()



## **Acciones en RDD de PySpark usando reduce**

Imagina que quieres sumar el total de las transacciones de una lista de un rdd, se puede hacer con reduce:

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

transacciones\_rdd = spark\_context.parallelize([100, 200, 300, 400, 500])

total\_transacciones = transacciones\_rdd.reduce(lambda x, y: x + y)

print("Total de transacciones del día:", total\_transacciones)

spark.stop()



# **Manipulación de datos con PySpark**

## **Crear un DataFrame en PySpark**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

spark\_context = spark.sparkContext

datos = [("Juan", 28, "Ingeniero"),

("Ana", 23, "Diseñadora"),

("Luis", 35, "Arquitecto"),

("Carlos", 45, "Analista"),

("Laura", 30, "Desarrolladora"),

("Mateo", 25, "Diseñador"),

("Sofía", 38, "Gerente de Proyecto"),

("Pedro", 50, "Director"),

("Elena", 27, "Marketing"),

("José" ,33, "Consultor"),

("Lucía", 29, "Administradora")]

nombre\_columnas = ["Nombre", "Edad", "Profesión"]

df = spark. createDataFrame(datos, nombre\_columnas)

# df = spark.read.csv("datos.csv", header=True, inferSchema=True) # from a csv

df.show()

# Filtrar filas donde la edad es mayor a 30

df\_filtrado = df.filter(df["Edad"] > 30)

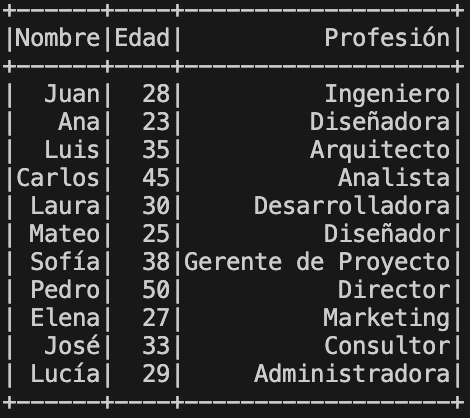
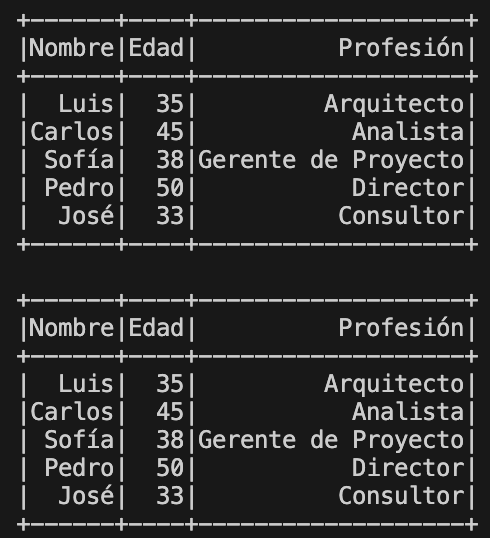
df\_filtrado.show()

# Same result as above but using where

df\_filtrado\_where = df.where(df["Edad"] > 30)

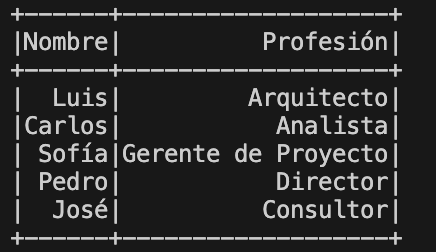
df\_filtrado\_where.show()

spark.stop()

## **Seleccionar columnas de un DataFrame en PySpark usando df.select()**

df\_filtrado\_where.select("Nombre", "Profesión").show()



## **Realizar transformaciones en una columna de un DataFrame en PySpark**

Ejemplo poner nombres en mayúscula, y agregar nueva columna con concatenación de tipo y valor

from pyspark.sql import SparkSession

from pyspark.sql. functions import lower, upper, concat\_ws

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

datos = [("Gen A", "Muestra", 10),

("gen b", "Muestra", 20),

("gen C", "Muestra", 30)]

nombre\_columnas = ["Nombre\_Gen", "Tipo", "Valor"]

df = spark. createDataFrame(datos, nombre\_columnas)

df\_transformado = df.withColumn("Nombre\_Gen", upper(df["Nombre\_Gen"]))

df\_transformado = df\_transformado.withColumn("Muestra\_ID", concat\_ws("\_", lower(df["Tipo"]), df["Valor"]))

df\_transformado.show()

spark.stop()



## **Aplicar métodos de agregación a un DataFrame en PySpark**

from pyspark.sql import SparkSession

from pyspark.sql. functions import sum, avg, max, min

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

datos = [("Enero", 200, "Jose"),

("Enero", 450, "Maria"),

("Febrero", 300, "Jose"),

("Febrero", 700, "Maria"),

("Marzo", 500, "Juan"),]

nombre\_columnas = ["Mes", "Ventas", "Vendedor"]

df = spark.createDataFrame(datos, nombre\_columnas)

df\_agg = df.groupBy("Mes").agg(

sum("Ventas").alias("TotalVentas"),

avg("Ventas").alias("PromedioVentas"),

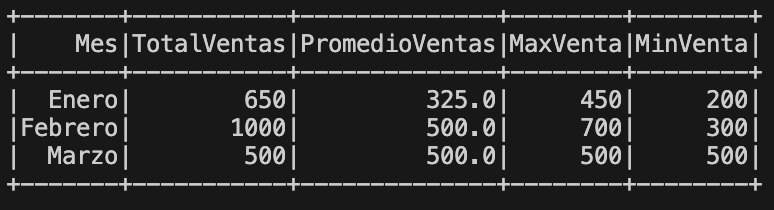
max("Ventas").alias("MaxVenta"),

min("Ventas").alias("MinVenta")

)

df\_agg.show()

spark.stop()



## **Ordenar elementos de un DataFrame en PySpark**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

datos = [("Juan", 28, "Ingeniero"),

("Ana", 23, "Diseñadora"),

("Luis", 35, "Arquitecto"),

("Carlos", 45, "Analista"),

("Laura", 30, "Desarrolladora"),

("Mateo", 25, "Diseñador"),

("Sofía", 38, "Gerente de Proyecto"),

("Pedro", 50, "Director"),

("Elena", 27, "Marketing"),

("José", 33, "Consultor"),

("Lucía", 29, "Administradora")]

columnas = ["Nombre", "Edad", "Profesión"]

df = spark.createDataFrame(datos, columnas)

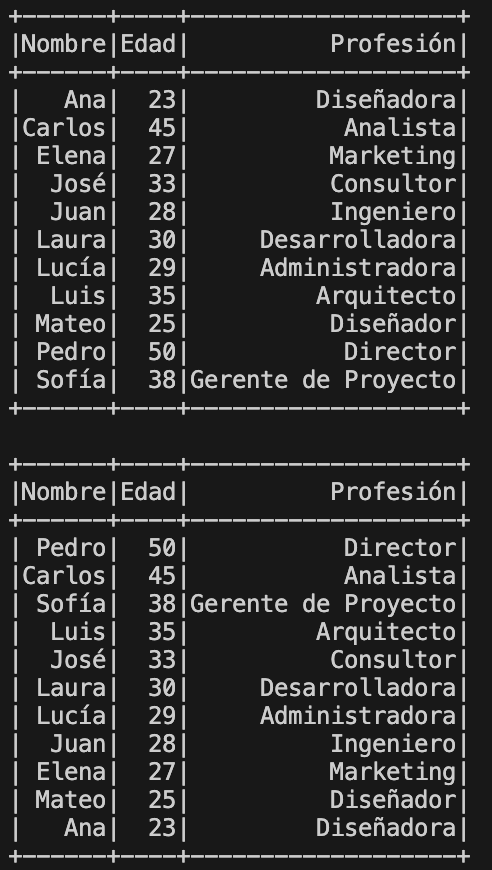
df\_ordenado = df.sort("Nombre")

df\_ordenado.show()

df\_ordenado\_desc = df.orderBy(df["Edad"].desc())

df\_ordenado\_desc.show()

spark.stop()



## **Operaciones condicionales en un DataFrame en PySpark (when, ortherwise)**

from pyspark.sql.functions import when

spark = SparkSession.builder \

.appName ("AppSpark") \

.getOrCreate()

datos = [("Juan", 28),

("Ana", 23),

("Luis", 35),

("Carlos" ,45),

("Laura", 30),

("Mateo", 25),

("Sofía", 38),

("Pedro" ,50),

("Elena", 27),

("José", 33),

("Lucía", 29)]

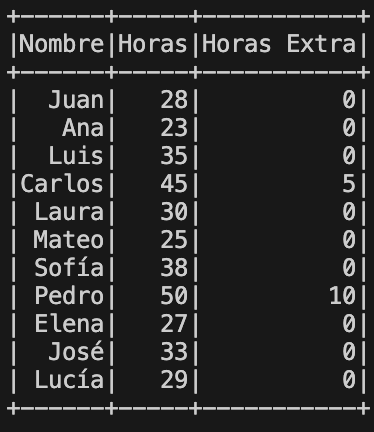
nombre\_columnas = ["Nombre", "Horas"]

df = spark.createDataFrame(datos, nombre\_columnas)

df\_condicional = df.withColumn("Horas Extra", when(df["Horas"] > 40, df["Horas"] - 40).otherwise(0))

df\_condicional.show()

spark.stop()



## **Operaciones Join en un DataFrame en PySpark**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("AppSpark") \

.getOrCreate()

clientes = [("1", "Juan", "España"),

("2", "Ana", "México"),

("3", "Luis", "Colombia")]

compras = [("1", "Teléfono", 1000),

("2", "Computadora", 1500),

("1", "Monitor", 550),

("4", "Audífonos", 360)]

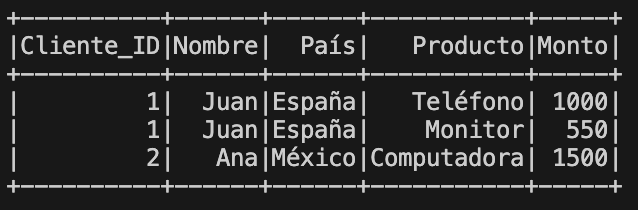
df\_clientes = spark.createDataFrame(clientes, ["Cliente\_ID", "Nombre", "País"])

df\_compras = spark.createDataFrame(compras, ["Cliente\_ID", "Producto", "Monto"])

df\_join = df\_clientes.join(df\_compras, on="Cliente\_ID", how="inner")

df\_join.show()

spark.stop()



Nota:

Existen diferentes tipos de join. En este caso, el inner sólo devolverá las filas que tienen coincidencias en ambos dataframes. El left join, mantiene las filas del dataframe izquierdo y las coincidentes del derecho, si no hay coincidencias en el derecho, se rellena con valores nulos. También está el right join, que es parecido al de left porque mantiene las filas del lado derecho y las filas coincidentes del lado izquierdo y, si no hay coincidencias, se rellenan con null, y por último el outer, donde mantiene todas las filas de ambos dataframes, donde no haya coincidencias se rellenan con null.

# **SciPy**

## **Ejercicios nivel básico con SciPy**

### **Crear una matriz CSR (Compressed Sparse Row) en SciPy**

Una matriz dispersa es una matriz que contiene **muchos ceros**. En lugar de guardar todos los ceros (lo que sería ineficiente en memoria), usamos estructuras como csr\_matrix para guardar **sólo los elementos no cero**, junto con su posición.

import numpy as np

from scipy.sparse import csr\_matrix

calificaciones\_peliculas = np.array([

[5, 0, 0, 0, 3],

[0, 4, 0, 2, 0],

[0, 0, 0, 0, 1],

[1, 0, 3, 0, 0],

[0, 2, 0, 0, 0]

])

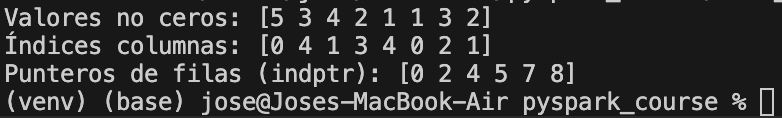
# Convertir a matriz dispersa CSR

matriz\_csr = csr\_matrix(calificaciones\_peliculas)

print("Valores no ceros:", matriz\_csr.data)

print("Índices columnas:", matriz\_csr.indices)

print("Punteros de filas (indptr):", matriz\_csr.indptr)



### **Suma y multiplicación de matrices CSR con SciPy**

import numpy as np

from scipy.sparse import csr\_matrix

datos\_usuarios = [1, 2, 3, 4]

indices\_columna\_usuarios = [0, 2, 2, 0]

indptr\_usuarios = [0, 2, 3, 4]

matriz\_usuarios = csr\_matrix((datos\_usuarios, indices\_columna\_usuarios, indptr\_usuarios), shape=(3, 3) )

datos\_productos = [5, 6, 7]

indices\_columna\_productos = [1, 0, 2]

indptr\_productos = [0, 1, 2, 3]

matriz\_productos = csr\_matrix((datos\_productos, indices\_columna\_productos, indptr\_productos), shape=(3, 3))

sum\_matriz = matriz\_usuarios + matriz\_productos

mult\_matriz = matriz\_usuarios.dot(matriz\_productos)

print("Matriz de usuarios:\n", matriz\_usuarios.toarray())

print("Matriz de productos:\n", matriz\_productos.toarray())

print("Suma de matrices CSR:")

print(sum\_matriz.toarray())

print("Multiplicación de matrices CSR:")

# La celda [i][j] del resultado es el producto punto entre la fila i de matriz\_usuarios y la columna j de matriz\_productos.

print(mult\_matriz.toarray())

# Fila 0 de A: [1, 0, 2]

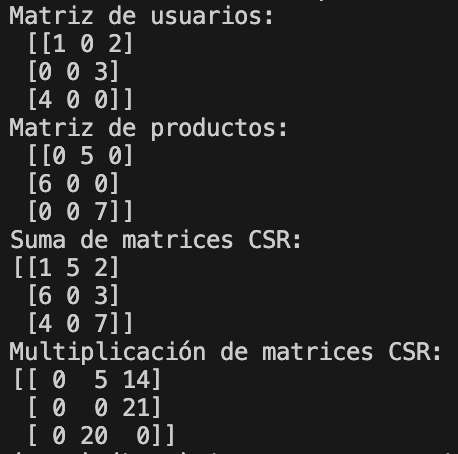
# Columna 0 de B: [0, 6, 0] → 1x0 + 0x6 + 2x0 = 0

# Fila 0 de A: [1, 0, 2]

# Columna 1 de B: [5, 0, 0] → 1x5 + 0x0 + 2x0 = 5

# Fila 0 de A: [1, 0, 2]

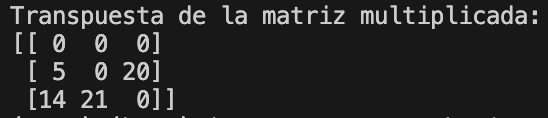
# Columna 2 de B: [0, 0, 7] → 1x0 + 0x0 + 2x7 = 14



### **Transposición de una matriz CSR con SciPy**

print("Transpuesta de la matriz multiplicada:")

print(mult\_matriz.transpose().toarray())



### **Contar y eliminar valores cero en una matriz CSR con SciPy**

import numpy as np

from scipy.sparse import csr\_matrix

datos\_originales = np.array([

[0, 0, 3],

[4, 0, 0],

[0, 1, 0],

[3, 0, 5]

])

matriz\_csr = csr\_matrix(datos\_originales)

print("Matriz original:\n", datos\_originales)

print("Matriz CSR:", matriz\_csr)

print("Número de elementos no cero:", matriz\_csr.count\_nonzero())

matriz\_csr[1, 1] = 0

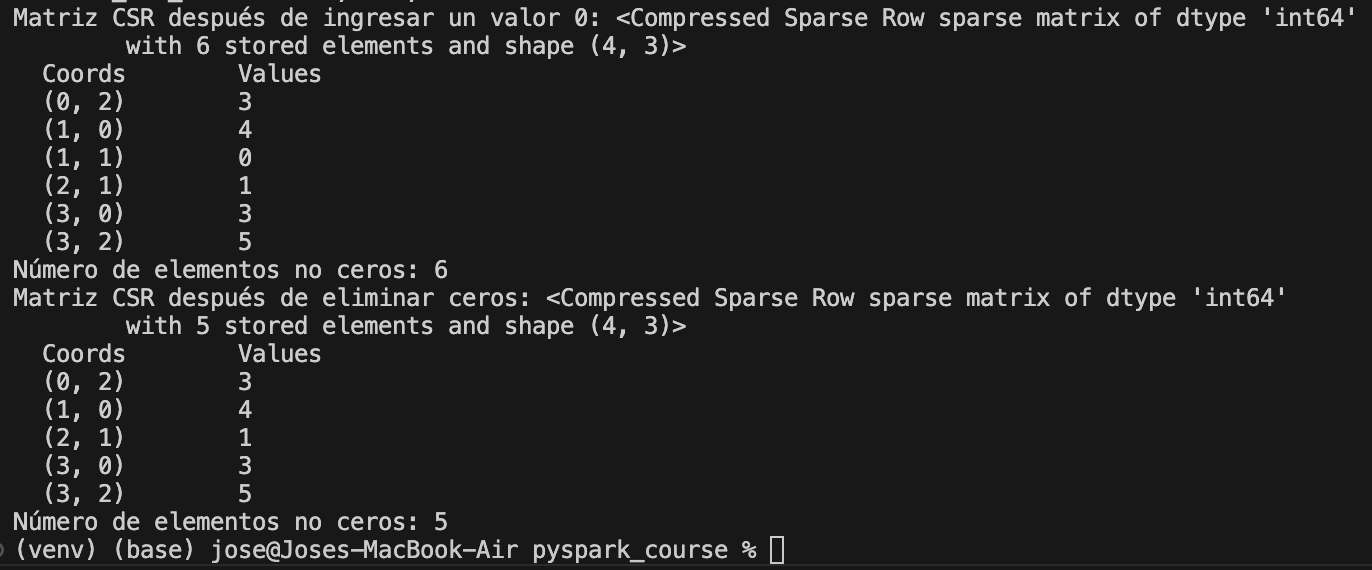
print("Matriz CSR después de ingresar un valor 0:", matriz\_csr)

print("Número de elementos no ceros:", matriz\_csr.nnz)

matriz\_csr.eliminate\_zeros()

print("Matriz CSR después de eliminar ceros:", matriz\_csr)

print("Número de elementos no ceros:", matriz\_csr.nnz)



### **Encontrar el determinante de una matriz cuadrada con SciPy**

import numpy as np

from scipy. linalg import det

matrix = np.array([

[4, 2, 3, 1],

[2, 3, 1, 4],

[1, 1, 2, 2],

[2, 3, 2, 4]

])

determinante = det(matrix)

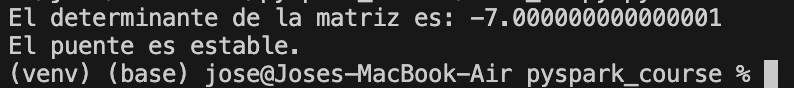
print("El determinante de la matriz es:", determinante)

if np.isclose(determinante, 0): # Tiene una tolerancia.

print("No tiene una solución única y el sistema podría ser inestable.")

else:

print("El puente es estable.")



### **Calcular los valores propios (eigenvalues) y los vectores propios (eigenvectors) con SciPy**

Debe calcular los valores propios o eigenvalues que representa la frecuencia de las vibras naturales del edificio y los vectores propios o eigenvectors que describen las direcciones en las que ocurren esas vibraciones.

import numpy as np

from scipy. linalg import eig

rigidez\_edificio = np.array ([

[6, -2, 1],

[-2, 5, -2],

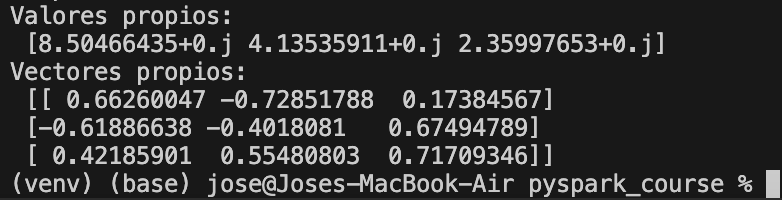
[1, -2, 4]

])

eigenvalues, eigenvectors = eig(rigidez\_edificio)

print("Valores propios:\n", eigenvalues)

print("Vectores propios:\n", eigenvectors)



### **La distancia euclidiana entre puntos dados con SciPy**

from scipy.spatial import distance

ubicacion\_maria = [2, 3]

ubicacion\_ana = [5, 1]

cafe\_1 = [3, 2]

cafe\_2 = [6, 3]

distancia\_maria\_cafe1 = distance.euclidean(ubicacion\_maria, cafe\_1)

print ("Distancia entre María y el Café 1:", distancia\_maria\_cafe1)

distancia\_ana\_cafe1 = distance.euclidean (ubicacion\_ana, cafe\_1)

print("Distancia entre Ana y el Café 1:", distancia\_ana\_cafe1)

distancia\_maria\_cafe2 = distance.euclidean(ubicacion\_maria, cafe\_2)

print("Distancia entre María y el Café 2:", distancia\_maria\_cafe2)

distancia\_ana\_cafe2 = distance.euclidean(ubicacion\_ana, cafe\_2)

print("Distancia entre Ana y el Café 2:", distancia\_ana\_cafe2)

distancia\_total\_cafe1 = distancia\_maria\_cafe1 + distancia\_ana\_cafe1

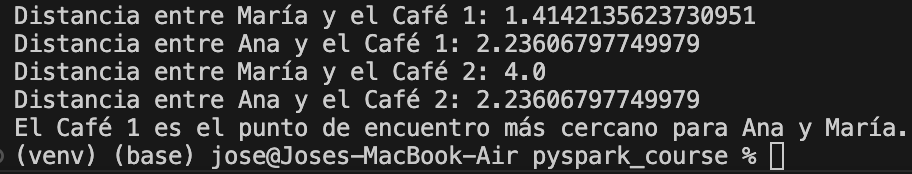
distancia\_total\_cafe2 = distancia\_maria\_cafe2 + distancia\_ana\_cafe2

if distancia\_total\_cafe1 < distancia\_total\_cafe2:

print("El Café 1 es el punto de encuentro más cercano para Ana y María.")

else:

print("El Café 2 es el punto de encuentro más cercano para Ana y María.")



### **Calcular la integral simple con SciPy**

Calcular cuánta agua pasa por una tubería en los primeros 5 segundos, a partir de una función de velocidad del agua. Hay que integrarla entre 0 y 5.

# Calcular el volumen total de agua en un tanque

# usando la integral definida de la función de velocidad

from scipy.integrate import quad

#f(t) = 2t^2 + 3t + 1

def velocidad(t):

return 2\*t\*\*2 + 3\*t + 1

resultado, error = quad(velocidad, 0, 5)

print(f"El volumen total de agua: {resultado} unidades cúbicas")

print(f"Error estimado: {error}")

## **2. Ejercicios de nivel intermedio con SciPy**

### **Resolver integrales dobles con SciPy**

from scipy.integrate import dblquad

# Definir la función que representa la superficie del terreno

# f(x, y) = x^2 + y^2

def funcion\_terreno(x, y):

return x\*\*2 + y\*\*2

# Calcular la integral doble de la función sobre el área:

# x va de 0 a 2, y va de 0 a 1 para cada x

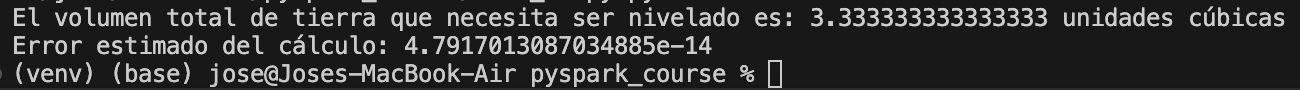
# dblquad(func, x\_min, x\_max, y\_min\_func, y\_max\_func)

resultado, error = dblquad(funcion\_terreno, 0, 2, lambda x: 0, lambda x: 1)

# Mostrar el resultado y el error estimado

print(f"El volumen total de tierra que necesita ser nivelado es: {resultado} unidades cúbicas")

print(f"Error estimado del cálculo: {error}")



### **La prueba t y P-Valor con SciPy**

import numpy as np

from scipy import stats

grupo\_vitamina = [1.1, 2.3, 1.8, 2.5, 2.2, 3.0, 2.9, 3.1, 2.8, 3.3]

grupo\_agua = [0.5, 0.6, 0.4, 0.8, 0.5, 0.7, 0.9, 0.6, 0.8, 0.5]

# t\_prueba: el valor del estadístico t de la prueba.

# p\_valor: la probabilidad de obtener una diferencia igual o mayor por azar si no hubiera diferencia real.

t\_prueba, p\_valor = stats.ttest\_ind(grupo\_vitamina, grupo\_agua)

print(f"Estadístico t: {t\_prueba}")

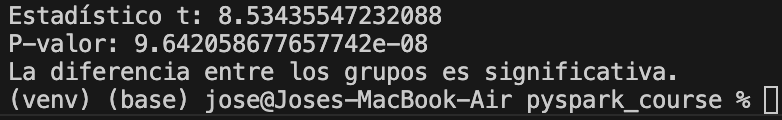
print(f"P-valor: {p\_valor}")

if p\_valor < 0.05: # Nivel de significancia del 5%. p < 0.05. Improbable que la diferencia sea por azar

print("La diferencia entre los grupos es significativa. Por lo tanto, puedes tener confianza razonable en que la vitamina realmente tiene un efecto diferente que el agua (en el contexto de tu experimento).")

else:

print("No hay una diferencia significativa entre los grupos.")



### **Análisis de regresión lineal con Scipy**

Tendencia de horas de estudio con las notas obtenidas usando la función*linregress*de SciPy**:**

Esta función devuelve varios valores importantes.

* **Pendiente:** Indica cuánto aumentan las notas por cada hora extra de estudio.
* **Intersección (ordenada al origen):** Es la nota aproximada que se espera si un estudiante no estudia nada.
* **Coeficiente de correlación (r):** Mide la fuerza y dirección de la relación entre horas de estudio y notas. El valor que se suele usar para evaluar es r2 (r al cuadrado), que indica qué porcentaje de la variabilidad en las notas se explica por las horas de estudio. Un valor cercano a 1 indica una relación fuerte.
* **P-valor:** Si es menor a 0.05, la relación entre horas de estudio y notas es estadísticamente significativa, es decir, muy poco probable que haya ocurrido por azar. Esto sugiere que realmente existe una asociación entre las horas de estudio y las notas.
* **Error estándar de la pendiente:** Mide la precisión de la estimación de la pendiente; un valor pequeño indica una estimación más confiable.

from scipy import stats

horas\_estudio = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

notas\_examen = [50, 55, 60, 65, 70, 75, 78, 82, 85, 88]

pendiente\_linea\_regresion, interseccion, r\_valor, p\_valor, std\_err = stats.linregress(horas\_estudio, notas\_examen)

print(f"Pendiente: {pendiente\_linea\_regresion}")

print(f"Intersección: {interseccion}")

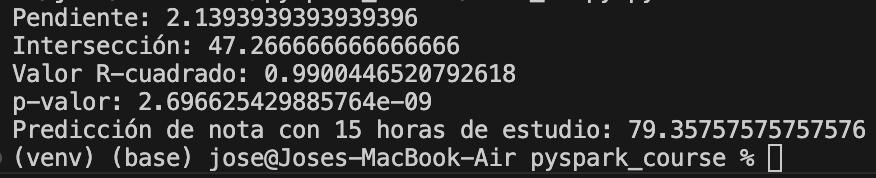
print(f"Valor R-cuadrado: {r\_valor\*\*2}")

print(f"p-valor: {p\_valor}")

horas\_futuras = 15

nota\_predicha = pendiente\_linea\_regresion \* horas\_futuras + interseccion

print(f"Predicción de nota con {horas\_futuras} horas de estudio: {nota\_predicha}")



## **3. Ejercicios de nivel avanzado con SciPy**

### **Creación de clúster con SciPy**

La función *linkage* crea una forma de agrupar a los usuarios según sus características, y el método *ward* es una forma de hacer el agrupamiento para asegurarnos que los usuarios que se agrupen estén muy cerca, o sea, que sean muy parecidos. El método *fcluster* recibe la matriz Z y escribe cómo se agrupan los usuarios, luego se le indica que vamos a hacer 3 cluster, y en criterion, *maxclust* indica que queremos un máximo de 3 clusters.

import numpy as np

from scipy.cluster.hierarchy import fcluster, linkage

import matplotlib.pyplot as plt

usuarios = np.array ([

[13, 10],

[14, 12],

[5, 9],

[15, 3],

[14, 4],

[16, 5],

[25, 1],

[24, 2],

[26, 1]

]) # [horas viendo películas, horas viendo series

Z = linkage(usuarios, method="ward")

clustering = fcluster(Z, 3, criterion='maxclust')

print("Clústeres asignados a los usuarios:", clustering)

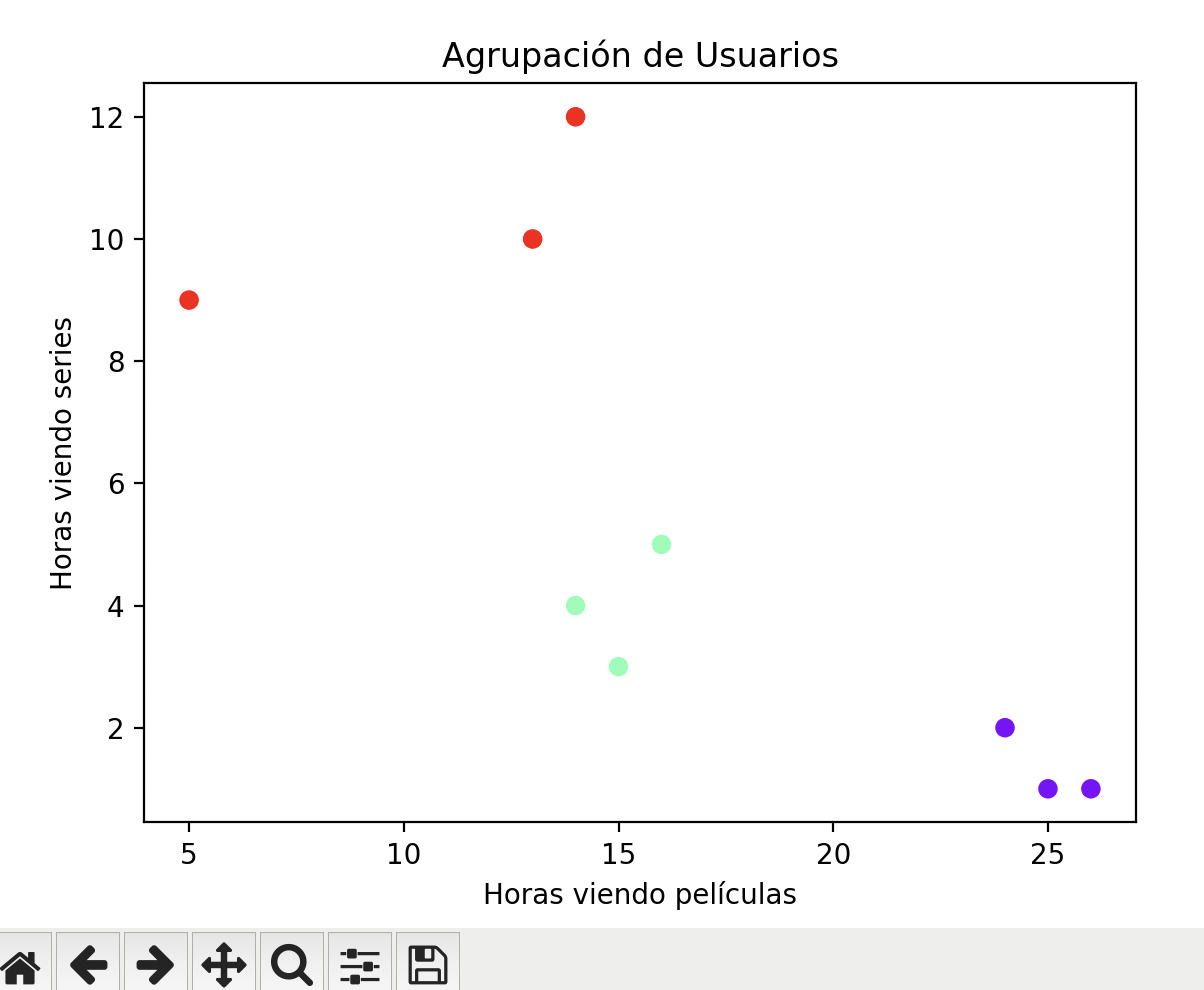
plt.scatter(usuarios[:,0], usuarios[:,1], c=clustering, cmap='rainbow')

plt.title("Agrupación de Usuarios")

plt.xlabel("Horas viendo películas")

plt.ylabel("Horas viendo series")

plt.show()





### **Transformada Rápida de Fourier con SciPy**

La Transformada de Fourier (FT) toma una señal en el tiempo (por ejemplo, un sonido, una vibración, una imagen) y la descompone en sus frecuencias.

Fourier dice:

"Voy a analizar esta señal y decirte qué frecuencias están presentes, y con cuánta intensidad."

Cuando escuchas una canción, realmente, lo que estás escuchando es una mezcla de varios sonidos. Por ejemplo, el sonido del bajo, la guitarra, la batería o cualquier otro instrumento que esté en esa canción. Lo que hace la transformación rápida de Fourier es ayudarnos a separar esos sonidos. Es como si dijéramos: «Mira, aquí está el sonido de la guitarra, aquí está el sonido del bajo, etc.

fft: para convertir sonido en frecuencias.

fftfreq: para saber qué frecuencia corresponde a cada punto.

import numpy as np

from scipy.fft import fft, fftfreq

import matplotlib.pyplot as plt

tasa\_muestreo = 1000 # 1000 muestras por segundo (Hz)

tiempo\_total = 1.0 # duración total de la señal: 1 segundo

t = np.linspace(0.0, tiempo\_total, int(tasa\_muestreo \* tiempo\_total), endpoint=False)

frecuencia\_guitarra = 150

frecuencia\_bajo = 60

sonido = np.sin(frecuencia\_guitarra \* 2.0 \* np.pi \* t) + 0.5 \* np.sin(frecuencia\_bajo \* 2.0 \* np.pi \* t)

yf = fft(sonido)

# fftfreq(n, d) devuelve las frecuencias correspondientes a los resultados de fft.

# n: número total de puntos (la longitud de la señal).

# d: el intervalo entre muestras en segundos (también llamado el período de muestreo).

xf = fftfreq(int(tasa\_muestreo \* tiempo\_total), 1/tasa\_muestreo)

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(t, sonido)

plt.title("Sonido de la Guitarra y el Bajo en el Tiempo")

plt.xlabel("Tiempo (segundos)")

plt.ylabel ("Amplitud")

plt.subplot(2, 1, 2)

plt.plot(xf[:tasa\_muestreo//2], np.abs(yf[:tasa\_muestreo//2]))

plt.title("Transformada Rápida de Fourier (Frecuencias)")

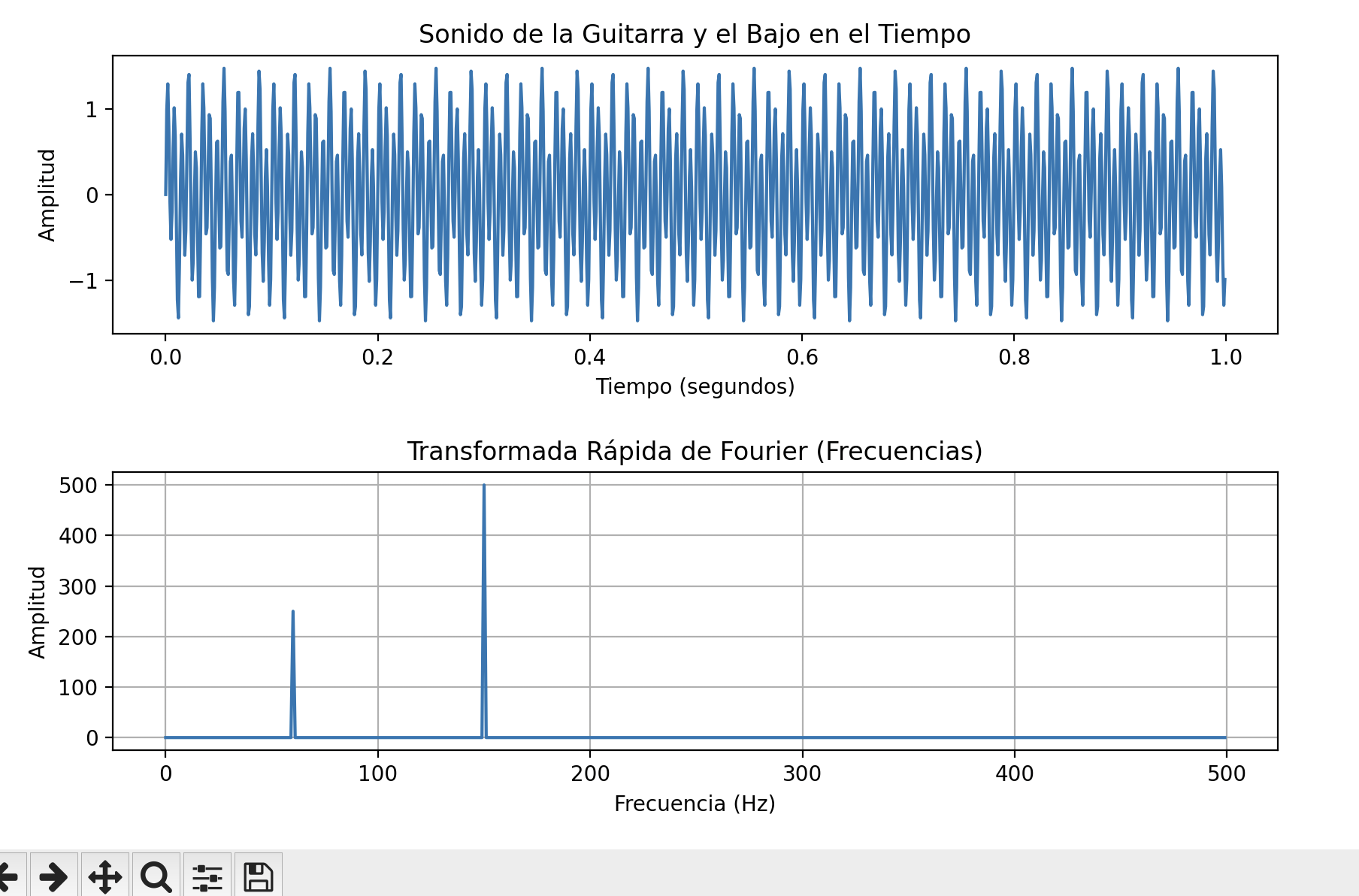
plt.xlabel("Frecuencia (Hz)")

plt.ylabel("Amplitud")

plt.grid()

plt.subplots\_adjust(hspace=0.5)

plt.show()



### **Transformada de coseno discreta con SciPy**

**Compresión de imágenes** usandola **Transformada Discreta del Coseno (DCT)**

import numpy as np

from scipy.fftpack import dct, idct

import matplotlib.pyplot as plt

from PIL import Image

imagen\_real = Image.open("planta.png")

imagen\_real = imagen\_real.convert("L") # Convierte a escala de grises ("L" = luminancia)

imagen\_np = np.array(imagen\_real)

print(imagen\_np)

dct\_imagen = dct(dct(imagen\_np.T, norm='ortho').T, norm='ortho') # Aplicar DCT

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Transformada de Coseno Discreta")

plt.imshow(dct\_imagen, cmap='gray')

plt.colorbar()

# Comprimir (eliminar detalles)

dct\_imagen\_comprimida = dct\_imagen.copy()

dct\_imagen\_comprimida[50:, 50:] = 0

# La IDCT es la operación inversa, y reconstruye una imagen desde los coeficientes comprimidos.

imagen\_comprimida = idct(idct(dct\_imagen\_comprimida.T, norm='ortho').T, norm='ortho')

plt.subplot (1, 2, 2)

plt.title("Imagen Comprimida" )

plt.imshow(imagen\_comprimida, cmap='gray')

plt.colorbar()

plt.show()



### **Procesamiento de imágenes multidimensionales con SciPy**

import numpy as np

import matplotlib.pyplot as plt

from scipy import ndimage

from PIL import Image

imagen\_color = Image.open("planta.png")

imagen\_np = np.array(imagen\_color)

print(imagen\_np)

imagen\_desenfocada = ndimage.gaussian\_filter(imagen\_np, sigma=1)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Imagen Original")

plt.imshow (imagen\_np)

plt.subplot(1, 2, 2)

plt.title("Imagen Desenfocada")

plt.imshow(imagen\_desenfocada)

plt.show()



### **Datos espaciales de SciPy**

import numpy as np

from scipy.spatial import distance

import matplotlib.pyplot as plt

escuelas = np.array([

[37.77, -122.42],

[37.78, -122.43],

[37.76, -122.45],

[37.79, -122.40]

])

hospitales = np.array([

[37.80, -122.41],

[37.74, -122.44],

])

distancias\_escuela\_hospital\_1 = distance.cdist(escuelas, [hospitales [0]], 'euclidean')

print("Distancias entre las escuelas y el Hospital 1:")

print(distancias\_escuela\_hospital\_1)

plt.figure(figsize=(8, 6))

plt.scatter(escuelas[:,1], escuelas[:,0], color='blue', label='Escuelas', marker='o')

plt.scatter(hospitales[:,1], hospitales[:,0], color='red', label='Hospitales', marker='x')

plt.title("Distribución de Escuelas y Hospitales")

plt.xlabel("Longitud")

plt.ylabel("Latitud")

plt.legend()

plt.show()

