Comenzando con Numpy y Pandas

Andrés F. Hidalgo Romero

Universidad San Pablo CEU
Escuela Politécnica Superior
June 26, 2024

Comenzando con Numpy y Pandas

Andrés F. Hidalgo Romero

June 26, 2024

Contents

1 Introducción

Este tutorial cubre las funciones principales de las bibliotecas ‘pandas‘, ‘numpy‘ y ‘matplotlib‘, esenciales para el análisis de datos básicos. A lo largo del documento, se proporcionarán ejemplos de código en Python, visualización de DataFrames y gráficos generados.

2 Numpy

‘numpy‘ es una biblioteca fundamental para la computación numérica en Python. Proporciona soporte para matrices y operaciones matemáticas de alto rendimiento.

2.1 Arrays

Los arrays de ‘numpy‘ son más eficientes que las listas de Python para operaciones matemáticas. Veamos cómo funciona:

```
import numpy as np

# Crear un array
arr = np.array([1, 2, 3, 4, 5])
print(arr)

# Operaciones con arrays
arr2 = arr * 2
print(arr2)
```

2.2 Operaciones Matemáticas

‘numpy‘ incluye una amplia gama de funciones matemáticas. Aquí puedes ver algunas funciones básicas:

```
# Suma, resta, multiplicación, y división
arr = np.array([1, 2, 3, 4])
print(np.sum(arr))
print(np.mean(arr))
print(np.std(arr))
```

3 Pandas

‘pandas‘ es una biblioteca poderosa para la manipulación y análisis de datos, construida sobre ‘numpy‘. Si numpy es el motor que impulsa el análisis numérico, pandas es la herramienta que proporciona la estructura y comodidad necesarias para manejar y manipular datos con eficiencia y elegancia.

3.1 DataFrames

Un DataFrame es una estructura de datos similar a una tabla en Excel. Veamos cómo crear uno:

```
import pandas as pd

# Crear un DataFrame
data = {
    'Nombre': ['Ana', 'Luis', 'Carlos'],
    'Edad': [23, 45, 34],
    'Ciudad': ['Madrid', 'Barcelona', 'Valencia']
}
df = pd.DataFrame(data)
print(df)
```

El DataFrame creado es el siguiente:

Nombre	Edad	Ciudad
Ana	23	Madrid
Luis	45	Barcelona
Carlos	34	Valencia

3.2 Lectura y Escritura de Archivos

'pandas' facilita la lectura y escritura de archivos CSV, proporcionando todas las herramientas necesarias para gestionar datos de manera efectiva:

```
# Leer un archivo CSV
df = pd.read_csv('archivo.csv')

# Escribir un archivo CSV
df.to_csv('archivo_salida.csv', index=False)
```

3.3 Manipulación de Datos

Filtrar, agregar y transformar datos es una tarea sencilla con pandas. A continuación, se presentan algunos ejemplos prácticos:

```
# Filtrar datos
df_filtrado = df[df['Edad'] > 30]

# Agregar datos
df['NuevaColumna'] = df['Edad'] * 2
print(df)
```

El DataFrame después de agregar una nueva columna es el siguiente:

Nombre	Edad	Ciudad	NuevaColumna
Ana	23	Madrid	46
Luis	45	Barcelona	90
Carlos	34	Valencia	68

3.4 Tablas Dinámicas (Pivot Tables)

Las tablas dinámicas permiten resumir y reorganizar datos de manera eficiente. Procedamos a su implementación:

```
# Crear una tabla dinámica
data = {
    'Ciudad': ['Madrid', 'Barcelona', 'Madrid', 'Valencia', 'Barcelona', 'Madrid'],
    'Ventas': [100, 200, 150, 50, 300, 200],
    'Año': [2020, 2020, 2021, 2021, 2021, 2020]
}
df = pd.DataFrame(data)

pivot_table = df.pivot_table(values='Ventas', index='Ciudad', columns='Año', aggfunc='sum')
print(pivot_table)
```

El resultado de la tabla dinámica es el siguiente:

Ciudad	2020	2021
Barcelona	200	300
Madrid	300	150
Valencia	0	50

3.5 Combinar DataFrames

'pandas' permite combinar DataFrames de diversas maneras. A continuación, se presenta un ejemplo sencillo pero eficaz:

```
# Crear DataFrames de ejemplo
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Nombre': ['Ana', 'Luis', 'Carlos']
})

df2 = pd.DataFrame({
    'ID': [2, 3, 4],
    'Salario': [50000, 60000, 70000]
})

# Merge
merged_df = pd.merge(df1, df2, on='ID', how='inner')
print(merged_df)
```

El resultado del merge es el siguiente:

ID	Nombre	Salario
2	Luis	50000
3	Carlos	60000

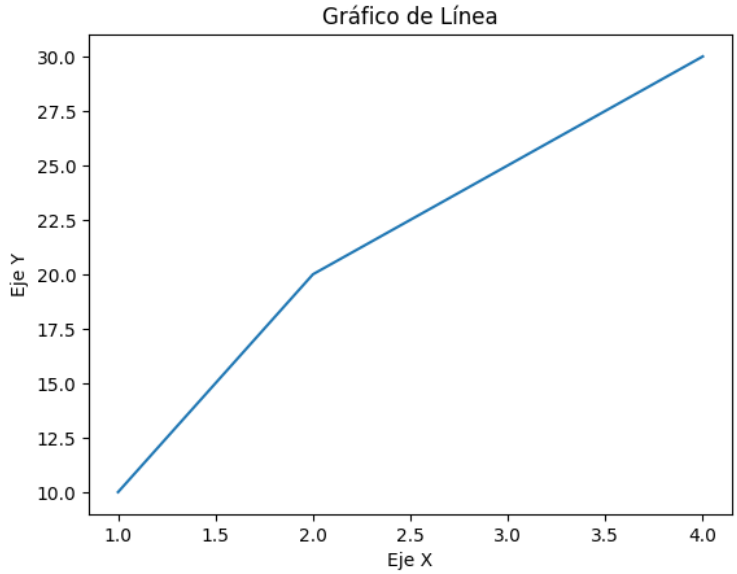
4 Matplotlib

'matplotlib' es una biblioteca utilizada para la creación de gráficos en Python. Con esta herramienta, podemos visualizar datos de manera efectiva con unos pocos pasos

4.1 Gráficos Básicos

Crear gráficos sencillos con 'matplotlib' es directo. Aquí tienes tu primer gráfico:

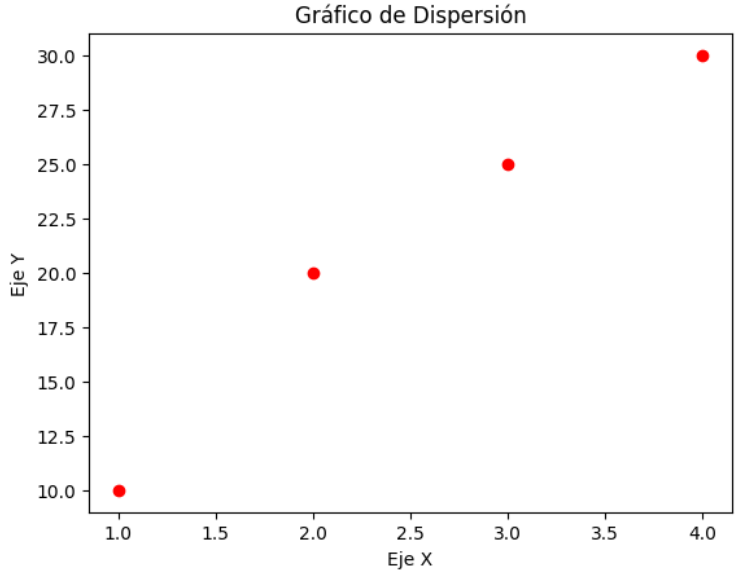
```
import matplotlib.pyplot as plt

# Datos
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

# Crear un gráfico
plt.plot(x, y)
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de Línea')
plt.savefig('grafico_linea.png')
plt.show()
```



4.2 Gráficos de Dispersión

Los gráficos de dispersión son útiles para visualizar la relación entre dos variables. Procedamos a crear un gráfico de dispersión.

```
# Crear un gráfico de dispersión
plt.scatter(x, y, color='red')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de Dispersión')
plt.savefig('grafico_dispersion.png')
plt.show()
```

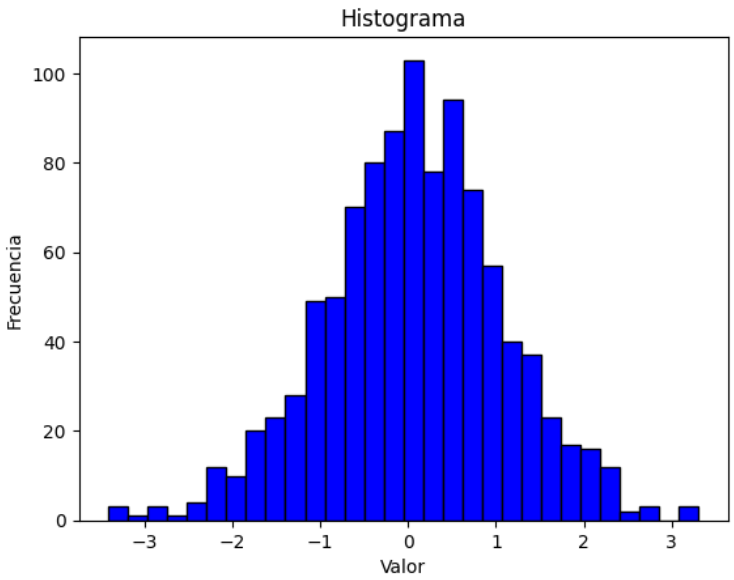


4.3 Histogramas

Los histogramas muestran la distribución de una variable. Veamos cómo se hace:

```
# Datos
data = np.random.randn(1000)

# Crear un histograma
```

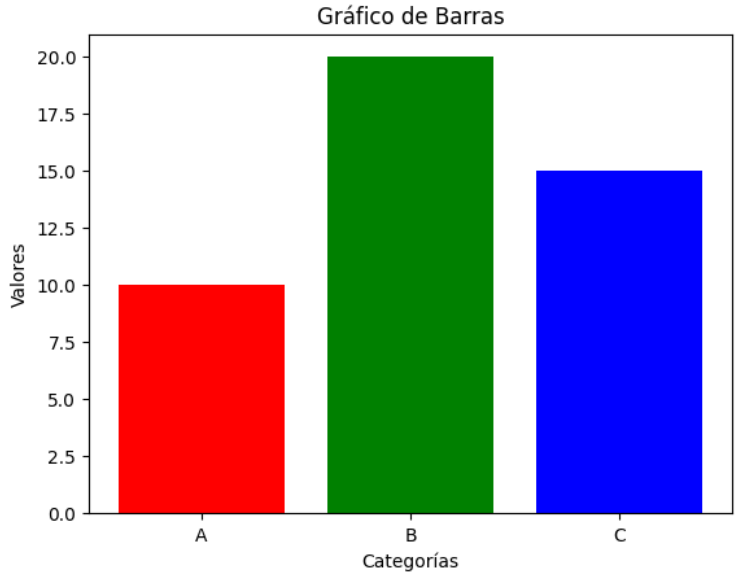
```
plt.hist(data, bins=30, color='blue', edgecolor='black')
plt.xlabel('Valor')
plt.ylabel('Frecuencia')
plt.title('Histograma')
plt.savefig('histograma.png')
plt.show()
```



4.4 Gráficos de Barras

Los gráficos de barras son ideales para comparar cantidades entre diferentes grupos. A continuación, se presentará un ejemplo de su uso

```
# Datos
categories = ['A', 'B', 'C']
values = [10, 20, 15]

# Crear un gráfico de barras
plt.bar(categories, values, color=['red', 'green', 'blue'])
plt.xlabel('Categorías')
plt.ylabel('Valores')
plt.title('Gráfico de Barras')
plt.savefig('grafico_barras.png')
plt.show()
```



5 Proyecto de Ejemplo

Vamos a crear un proyecto sencillo que combine 'numpy', 'pandas' y 'matplotlib'. Analizaremos un conjunto de datos, realizaremos operaciones básicas y visualizaremos los resultados.

5.1 Preparación de Datos

Primero, generamos datos sintéticos y los cargamos en un DataFrame. Este DataFrame será una herramienta fundamental en nuestro análisis:

```
import numpy as np
import pandas as pd

# Generar datos sintéticos
np.random.seed(0)
data = {
    'Ciudad': np.random.choice(['Madrid', 'Barcelona', 'Valencia'], 100),
    'Edad': np.random.randint(18, 70, 100),
    'Salario': np.random.randint(20000, 80000, 100)
}

df = pd.DataFrame(data)
print(df.head())
```

El DataFrame generado es el siguiente:

	Ciudad	Edad	Salario
0	Madrid	63	62488
1	Barcelona	18	71684
2	Madrid	52	39414
3	Valencia	28	26565
4	Valencia	60	27538

5.2 Análisis de Datos

Realizamos un análisis básico de los datos. El conocimiento detallado de los datos es fundamental para el éxito del análisis:

```
# Descripción estadística
print(df.describe())
```

```
# Tabla dinámica de salarios medios por ciudad
pivot_table = df.pivot_table(values='Salario', index='Ciudad', aggfunc=np.mean)
print(pivot_table)
```

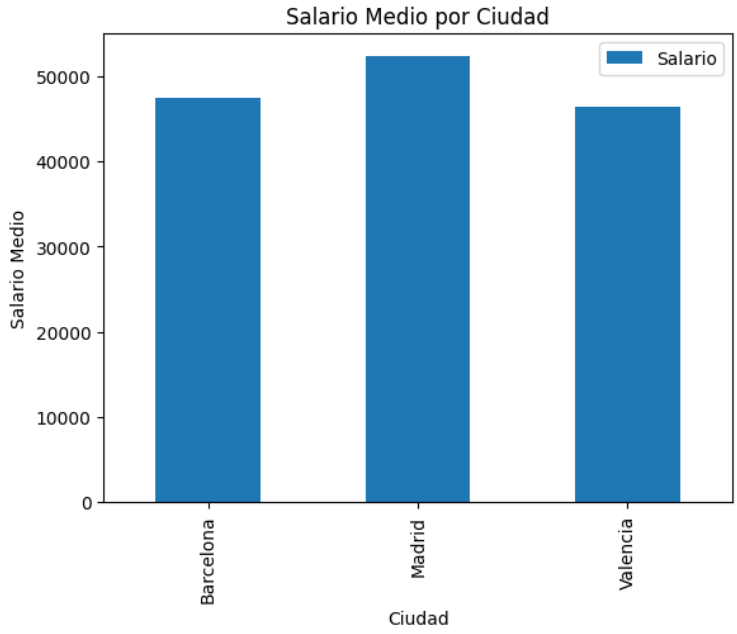
El resultado de la tabla dinámica es el siguiente:

Ciudad	Salario
Barcelona	48534.2
Madrid	49082.0
Valencia	50441.1

5.3 Visualización de Datos

Visualizamos los datos utilizando 'matplotlib', ya que una representación gráfica adecuada facilita una comprensión más profunda de los datos:

```
import matplotlib.pyplot as plt

# Gráfico de barras de salarios medios por ciudad
pivot_table.plot(kind='bar')
plt.xlabel('Ciudad')
plt.ylabel('Salario Medio')
plt.title('Salario Medio por Ciudad')
plt.savefig('salario_medio_ciudad.png')
plt.show()
```



6 Ejemplo de Problema: Gasto por Departamento y Tarea

Supongamos que tienes dos DataFrames con información de empleados y tareas. El objetivo es calcular el gasto de los departamentos por tarea. Este ejemplo muestra el uso de la unión con diferentes claves y el resumen de gastos.

6.1 Crear DataFrames

Creamos los DataFrames con información de empleados y tareas.

```
# DataFrame con información de empleados
empleados = pd.DataFrame({
    'ID_Empleado': [10, 20, 30, 60],
    'Nombre': ['Carlos', 'Marta', 'Rafael', 'Jorge'],
    'Departamento': ['Ventas', 'Marketing', 'IT', 'IT'],
    'Gastos': [1000, 120, 250, 500]
})

# DataFrame con información de tareas
tareas = pd.DataFrame({
    'Empleado_ID': [20, 30, 10, 50, 60],
    'Tarea': ['Informe', 'Desarrollo', 'Presentación', 'Limpieza', 'Insumo']
})
```

Los DataFrames creados son los siguientes:
Empleados:

ID_Empleado	Nombre	Departamento	Gastos
10	Carlos	Ventas	1000
20	Marta	Marketing	120
30	Rafael	IT	250
60	Jorge	IT	500

Tareas:

Empleado_ID	Tarea
20	Informe
30	Desarrollo
10	Presentación
50	Limpieza
60	Insumo

6.2 Unir DataFrames

Usamos ‘merge‘ para unir los DataFrames basándonos en las claves correspondientes.

```
# Unir los DataFrames
df_merged = pd.merge(empleados, tareas, left_on='ID_Empleado',
right_on='Empleado_ID', how='inner')
print(df_merged)
```

El resultado de la unión es el siguiente:

ID_Empleado	Nombre	Departamento	Gastos	Empleado_ID	Tarea
10	Carlos	Ventas	1000	10	Presentación
20	Marta	Marketing	120	20	Informe
30	Rafael	IT	250	30	Desarrollo
60	Jorge	IT	500	60	Insumo

6.3 Resumen de Gastos por Departamento y Tarea

Creamos una tabla dinámica para resumir los gastos por departamento y tarea.

```
# Crear tabla dinámica
pivot_table_gastos = df_merged.pivot_table(values='Gastos', index='Departamento',
columns='Tarea', aggfunc='sum', fill_value=0)
print(pivot_table_gastos)
```

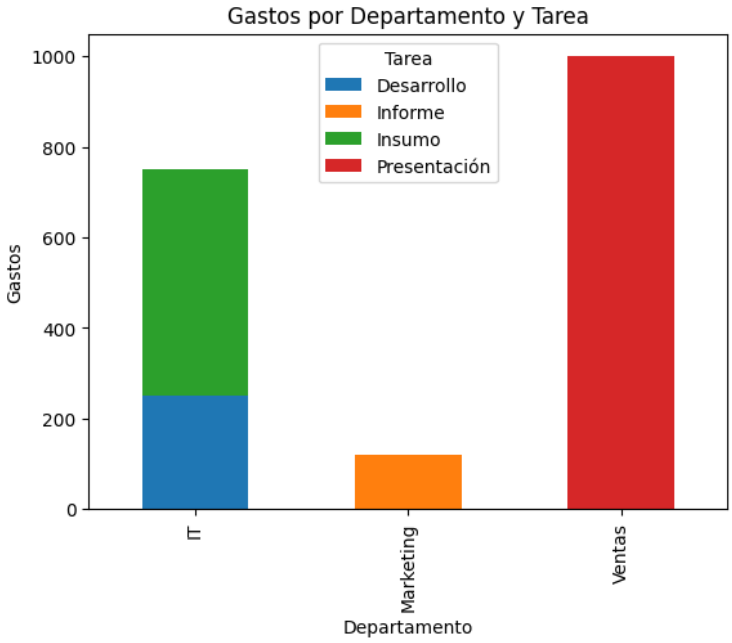
El resultado de la tabla dinámica es el siguiente:

Departamento	Desarrollo	Informe	Insumo	Presentación
IT	250	0	500	0
Marketing	0	120	0	0
Ventas	0	0	0	1000

6.4 Visualización de los Resultados

Visualizamos los resultados con ‘matplotlib‘.

```
# Gráfico de barras apiladas de gastos por departamento y tarea
pivot_table_gastos.plot(kind='bar', stacked=True)
plt.xlabel('Departamento')
plt.ylabel('Gastos')
plt.title('Gastos por Departamento y Tarea')
plt.savefig('gastos_departamento_tarea.png')
plt.show()
```



7 Conclusión

Este tutorial ha abordado las funciones fundamentales de ’numpy’, ’pandas’ y ’matplotlib’, estableciendo una base robusta para el análisis de datos en Python. A través de ejemplos prácticos y escenarios reales, se ha demostrado el potencial y la utilidad de la combinación de estas bibliotecas para resolver problemas en diversas áreas de la ingeniería.