Introducción a los Lenguajes de Programación en Ciencias
PYTHON EN CIENCIA DE DATOS

Henry R. Moncada

Universidad Nacional del Callao
Facultad de Ciencias Naturales y Matemáticas

24 de agosto de 2025

Contenido

1. Introducción a Python en Ciencia de Datos
2. Bibliotecas especializadas
3. Análisis de Datos con Pandas
4. Introducción a DataFrames
   - Importar Archivo
   - Limpiar Datos
   - Ordenar y Agregar Datos
   - Exportar Datos
   - Ejemplos de Análisis de Datos con Pandas
   - Las ventajas de usar Pandas?
5. Visualización de Datos
   - Introducción a Matplotlib
   - Introducción a Seaborn
6. Machine Learning
   - Machine Learning con Scikit-learn
   - Visualización de Datos con Matplotlib
   - Evaluación de Modelos con Scikit-learn
7. Consideraciones y Recomendaciones
8. Conclusión

¿Por qué Python?

Python es uno de los **lenguajes más populares en ciencia de datos** debido a:

- Su **sintaxis** clara y legible.
- Una amplia gama de **bibliotecas especializadas** (NumPy, Scipy, Pandas, Matplotlib, Scikit-learn).
- **Comunidad activa** y soporte continuo.
- **Versatilidad en aplicaciones** (análisis de datos, aprendizaje automático, visualización, etc.).

Hola mundo en Python

- Python 2.7 fue lanzado en 2010 como última versión mayor de la serie 2.x.
- Python 3 introdujo cambios incompatibles con 2.x.
- Python 2 llegó al fin de su vida útil en enero de 2020.

```
(base) lola:~$ vi 0.Hello_World.py
```

- **print** es una función en Python 3.
- **Division**: 3/2 es distinto en Python 2 que en 3.
- Unicode por defecto en Python 3.
- Cambios en funciones como `range()`, `input()`, `dict.keys()`.

```python
## 1.Print to the terminal
# python Hello_World.py

message = "Hello World!!!"
print(message)

print("Buenas Noches")

# Perform addition operation 1+1 and print the result
result = 1 + 1
print(result)
print("suma = %d"% result)
```

Python programa: Hola Mundo

Executar programa

```
(base) lola:~$ python 0.Hello_World.py
Hello World!!!
Buenas Noches
2
suma = 2

2  103    Jana       Prague   33      81.0
```

Python como Calculadora

- Abrir un terminal `CTRL+ALT+T`
- Escribir **python** despues de `prompt`

```
(base) lola:~$ python
Python 3.11.7 (main, Dec 15 2023, 18:12:31) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
(base) lola:~$ python
Python 3.11.7 (main, Dec 15 2023, 18:12:31) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> b=3
>>> c=a+b
>>> c
5
>>> c**2
25
```

- Importar librerias

```
(base) lola:~$ python
Python 3.11.7 (main, Dec 15 2023, 18:12:31) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> np.sqrt(25)
5.0
```

1. NumPy

- Soporte para arrays multidimensionales.
- Operaciones vectorizadas de alto rendimiento.
- Base para muchas otras bibliotecas.

g Example

```
In [1]: import numpy as np

In [2]: x = np.arange(12)

In [3]: x = x.reshape(4, 3)

In [4]: x
Out[4]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])

In [6]: x = x - np.mean(x, axis=0)

In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [ 1.5,  1.5,  1.5],
       [ 4.5,  4.5,  4.5]])
```

2. Pandas

- Estructuras de datos: `DataFrame` y `Series`.
- Limpieza, transformación y análisis de datos.
- Ideal para datos tabulares.

3. Matplotlib

- Biblioteca de visualización 2D.
- Gráficos de líneas, barras, histogramas, dispersión, etc.
- Altamente personalizable.

4. Seaborn

- Basada en Matplotlib, pero con una interfaz de alto nivel.
- Visualizaciones estadísticas elegantes.
- Gráficos como mapas de calor, diagramas de violín, etc.

5. Plotly

- Visualización interactiva en web.
- Gráficos 3D, dashboards y mapas.
- Compatible con Jupyter Notebooks.

- Extiende NumPy con módulos para:
  - Álgebra lineal
  - Optimización
  - Estadística
  - Señales y más

7. Scikit-learn

- Biblioteca de Machine Learning.
- Modelos de regresión, clasificación, clustering y reducción de dimensionalidad.
- Fácil de usar, ideal para prototipos rápidos.

8. Statsmodels

- Modelos estadísticos tradicionales: OLS, ARIMA, etc.
- Análisis exploratorio de datos y pruebas estadísticas.
- Complemento perfecto para Scikit-learn.

- Framework de Deep Learning de Google.
- Computación numérica eficiente con GPUs/TPUs.
- Para modelos complejos y redes neuronales.

- Interfaz de alto nivel para TensorFlow.
- Permite crear modelos de deep learning fácilmente.
- Ideal para principiantes y prototipado rápido.

11. PyTorch

- Framework de deep learning de Facebook.
- Computación dinámica con fuerte integración a Python.
- Usado extensivamente en investigación y producción.

12. XGBoost

- Implementación optimizada de árboles de decisión.
- Muy eficiente y preciso para tareas de clasificación y regresión.
- Comúnmente utilizado en competencias de Kaggle.

13. Anaconda

- Anaconda es una distribución gratuita y de código abierto de Python y R.
- Diseñada específicamente para ciencia de datos, análisis numérico, aprendizaje automático y computación científica.
- Incluye más de 1,500 paquetes preinstalados, así como herramientas como Jupyter Notebook, Spyder y Conda.
- Facilita la gestión de entornos virtuales y dependencias.

14.Spyder

- IDE (Entorno de Desarrollo Integrado) orientado a científicos e ingenieros.
- Integra consola interactiva, explorador de variables y editor avanzado.
- Perfecto para análisis numérico, visualización y programación estructurada en Python.
- Similar en estilo a MATLAB, lo que facilita la transición.

15.Jupyter

- Entorno interactivo para escribir y ejecutar código Python.
- Permite combinar código, texto (Markdown), ecuaciones y visualizaciones.
- Muy usado en ciencia de datos, enseñanza y documentación de proyectos.
- Facilita la exploración de datos en tiempo real.

¿Qué es Pandas?

Pandas es una biblioteca de Python para el análisis y manipulación de datos estructurados. Ofrece herramientas poderosas como:

- DataFrames para manejar datos tabulares.
- Funciones de filtrado, agrupación y agregación.
- Soporte para datos faltantes y operaciones avanzadas.

Es ampliamente utilizado en ciencia de datos debido a su flexibilidad y eficiencia.

In [2]:

```python
# Load our necessary libraries
import pandas as pd
import numpy as np
```

In [3]:

```python
# Create data into CSV (that we'll import later on)
# Let's say the data is for a veterinarian keeping tabs on his clients.

raw_data = {'pet_name': ['Woof', 'Chester', 'Rex', 'Mystery', 'Pumpkin'],
        'pet_last_name': ['Smith', 'Kim', "", 'Taylor', ""],
        'good_pet_score': [96, 34, 89, 92, 79],
        'type': ['dog', 'cat', 'mini-dinosaur', "unknown", "bird"],
        'amount_owed': ["5000", "9,000", 570, 622, 190]}
df = pd.DataFrame(raw_data, columns = ['pet_name', 'pet_last_name', 'good_pet_score', 'type', 'amount_owed'])
df
```

Out[3]:

	pet_name	pet_last_name	good_pet_score	type	amount_owed
0	Woof	Smith	96	dog	5000
1	Chester	Kim	34	cat	9,000
2	Rex		89	mini-dinosaur	570
3	Mystery	Taylor	92	unknown	622
4	Pumpkin		79	bird	190

Data

Column labels

	name	city	age	py-score
101	Xavier	Mexico City	41	88.0
102	Ann	Toronto	28	79.0
103	Jana	Prague	33	81.0
104	Yi	Shanghai	34	80.0
105	Robin	Manchester	38	68.0
106	Amal	Cairo	31	61.0
107	Nori	Osaka	37	84.0

Row labels Data

```python
data = {
      'name': ['Xavier', 'Ann', 'Jana', 'Yi', 'Robin', 'Amal', 'Nori'],
      'city': ['Mexico City', 'Toronto', 'Prague', 'Shanghai', 'Manchester', 'Cairo', 'Osaka'],
      'age': [41, 28, 33, 34, 38, 31, 37],
      'py-score': [88.0, 79.0, 81.0, 80.0, 68.0, 61.0, 84.0]
 }

row_labels = [101, 102, 103, 104, 105, 106, 107]
```

¿Qué es un DataFrame?

Un **DataFrame** es una estructura de datos bidimensional en Pandas que permite almacenar y manipular datos tabulares. Es ampliamente utilizado en ciencia de datos debido a:

- Su capacidad para manejar datos estructurados.
- Funciones integradas para limpiar, transformar y analizar datos.
- Soporte para múltiples formatos de entrada/salida (CSV, Excel, texto, etc.).
- Los DataFrames de Pandas son herramientas poderosas para manipular datos estructurados.
- Permite importar, limpiar, ordenar, agregar y exportar datos de manera eficiente.
- Es ideal para proyectos de ciencia de datos que involucran análisis y visualización.

Dataframe

```python
import pandas as pd
# Data
data = {
      'name': ['Xavier', 'Ann', 'Jana', 'Yi', 'Robin', 'Amal', 'Nori'],
      'city': ['Mexico City', 'Toronto', 'Prague', 'Shanghai', 'Manchester', 'Cairo', 'Osaka'],
      'age': [41, 28, 33, 34, 38, 31, 37],
      'py-score': [88.0, 79.0, 81.0, 80.0, 68.0, 61.0, 84.0]
 }
# etiquetas filas
row_labels = [101, 102, 103, 104, 105, 106, 107]
# Dataframe
df = pd.DataFrame(data=data, index=row_labels)
# imprimir dataframe
print(df)
print("\n")
# imprimir las dos primeras filas
print(df.head(n=2))
print("\n")
# imprimir las ultima dos fils
print(df.tail(n=2))
```

```
$ python example_12_pandas_dataframe.py
       name         city  age  py-score
101  Xavier  Mexico City   41      88.0
102     Ann      Toronto   28      79.0
103    Jana       Prague   33      81.0
104      Yi     Shanghai   34      80.0
105   Robin   Manchester   38      68.0
106    Amal        Cairo   31      61.0
107    Nori        Osaka   37      84.0


       name         city  age  py-score
101  Xavier  Mexico City   41      88.0
102     Ann      Toronto   28      79.0


     name   city  age  py-score
106  Amal  Cairo   31      61.0
107  Nori  Osaka   37      84.0
```

Importar Archivo

- **Importar Archivo CSV**

```python
import pandas as pd

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

# Mostrar las primeras filas
print(df.head(3))
```

```
$ python example_13_pandas_open_data_csv.py
               name           city    age    py-score
0     101    Xavier    Mexico City     41          88
1     102       Ann        Toronto     28          79
2     103      Jana         Prague     33          81
```

Importar archivo CSV

- **Importar Archivo de Texto**

```python
import pandas as pd

# Leer archivo de texto separado
      por una tabulacion
df = pd.read_csv('data.txt',
      delimiter = '\t')

# Mostrar las primeras filas
print(df.head(3))
```

```
$ python example_13_pandas_open_data_txt.py
          name         city  age  py-score
0  101  Xavier  Mexico City   41      88.0
1  102     Ann      Toronto   28      79.0
2  103    Jana       Prague   33      81.0
```

Importar archivo de texto

Agregar Filas

- **Importar Archivo Excel**

```python
import pandas as pd

# Leer archivo Excel
df = pd.read_excel('data.xlsx',
      sheet_name='data')

# Mostrar las primeras filas
print(df.head(3))
```

```
$ python example_13_pandas_open_data_xlsx.py
                name           city    age    py-score
0     101    Xavier    Mexico City     41          88
1     102       Ann        Toronto     28          79
2     103      Jana         Prague     33          81
```

Importar archivo Excel

- **Agregar Filas**

```python
import pandas as pd

# Crear DataFrame
data = {'Nombre': ['Ana', 'Juan'],
        'Edad': [25, 30]}
df = pd.DataFrame(data)

# Agregar nueva fila
new_row = {'Nombre': 'Maria', 'Edad': 22}
df = df.append(new_row, ignore_index=True)

print(df)
```

```
$ python example_14_pandas_append_row.py
  Nombre  Edad
0    Ana    25
1   Juan    30
  Nombre  Edad
0    Ana    25
1   Juan    30
2  Maria    22
```

Agregar filas

Eliminar Filas y Rellenar Valores Faltantes

- **Eliminar Filas con Valores Faltantes**

```python
import pandas as pd

# Crear DataFrame con valores faltantes
data = {'Nombre': ['Ana', 'Juan', None],
        'Edad': [25, None, 30]}
df = pd.DataFrame(data)

# Eliminar filas con valores faltantes
df_clean = df.dropna()

print(df_clean)
```

```
$ python example_14_pandas_clean_row.py
  Nombre  Edad
0    Ana  25.0
```

Eliminar filas con valores faltantes

- **Rellenar Valores Faltantes**

```python
import pandas as pd

# Crear DataFrame con valores faltantes
data = {'Nombre': ['Ana', 'Juan', None],
        'Edad': [25, None, 30]}
df = pd.DataFrame(data)

# Rellenar valores faltantes
df_filled = df.fillna({'Nombre': '
     Desconocido', 'Edad': 0})

print(df_filled)
```

```
$ python example_14_pandas_fill_missing_values.py
       Nombre  Edad
0         Ana  25.0
1        Juan   0.0
2 Desconocido  30.0
```

Rellenar valores faltantes

Ordenar y Agregar Datos

- **Ordenar Datos por Columna**

```python
import pandas as pd

# Crear DataFrame
data = {'Nombre': ['Ana', 'Juan', 'Maria'],
        'Edad': [25, 30, 22]}
df = pd.DataFrame(data)

# Ordenar por edad
df_sorted = df.sort_values(by='Edad', ascending=True)

print(df_sorted)
```

```
$ python example_14_pandas_sort_values.py
  Nombre  Edad
2  Maria    22
0    Ana    25
1   Juan    30
```

Ordenar datos por columna

- **Agregar Nuevas Columnas**

```python
import pandas as pd

# Crear DataFrame
data = {'Nombre': ['Ana', 'Juan', 'Maria'],
        'Edad': [25, 30, 22]}
df = pd.DataFrame(data)

# Agregar nueva columna
df['Ciudad'] = ['Lima', 'Bogota', 'Santiago']

print(df)
```

```
$ python example_14_pandas_add_column_values.py
  Nombre  Edad    Ciudad
0    Ana    25      Lima
1   Juan    30    Bogota
2  Maria    22  Santiago
```

Agregar nuevas columnas

Exportar DataFrame a CSV y DataFrame a Excel

- **Exportar DataFrame a CSV**

```python
import pandas as pd

# Crear DataFrame
data = {'Nombre': ['Ana', 'Juan'],
        'Edad': [25, 30]}
df = pd.DataFrame(data)

# Exportar a CSV
df.to_csv('datos_exportados.csv', index=
    False)
```

```
$ python example_15_pandas_export_csv.py
$ ll datos_exportados.csv
-rw-rw-r-- 1 henry henry 27 Apr  3 21:25 datos_exportad
```

Exportar DataFrame a CSV

- **Exportar DataFrame a Excel**

```python
import pandas as pd

# Crear DataFrame
data = {'Nombre': ['Ana', 'Juan'],
        'Edad': [25, 30]}
df = pd.DataFrame(data)

# Exportar a Excel
df.to_excel('datos_exportados.xlsx', index=
    False)
```

```
$ python example_15_pandas_export_xlsx.py
$ ll datos_exportados.xlsx
-rw-rw-r-- 1 henry henry 4949 Apr  3 21:27 datos_export
```

Exportar DataFrame a Excel

Análisis de Datos con Pandas

Pandas es una biblioteca poderosa para manipular y analizar datos estructurados.

```python
import pandas as pd

# Crear un DataFrame
data = {
    'Nombre': ['Ana', 'Juan', 'Maria','Jose', 'Roberto'],
    'Edad': [25, 30, 22, 23, 33],
    'Ciudad': ['Lima', 'Bogota', 'Santiago', 'Caracas', 'Bogota']
}

df = pd.DataFrame(data)

# Mostrar el DataFrame
print("DataFrame:")
print(df)

# Filtrar personas mayores de 24 anos
filtro = df[df['Edad'] > 24]
print("\nPersonas mayores de 24 anos:")
print(filtro)
```

Análisis básico con Pandas

```
$ python example_1_pandas_Dataframe.py
DataFrame:
    Nombre  Edad    Ciudad
0      Ana    25      Lima
1     Juan    30    Bogota
2    Maria    22  Santiago
3     Jose    23   Caracas
4  Roberto    33    Bogota

Personas mayores de 24 anos:
    Nombre  Edad  Ciudad
0      Ana    25    Lima
1     Juan    30  Bogota
4  Roberto    33  Bogota
```

```python

# Datos de tiempo y posicion
data = {'Tiempo (s)': [0, 1, 2, 3, 4],
        'Posicion (m)': [0, 5, 10, 15, 20]}

df = pd.DataFrame(data)

# Calcular velocidad media
df['Velocidad Media (m/s)'] = df['Posicion (m)'].diff() / df['Tiempo (s)'].diff()
print(df)
```

```
$ python example_2_pandas_Dataframe_Fisica.py
   Tiempo (s)  Posicion (m)  Velocidad Media (m/s)
0           0             0                    NaN
1           1             5                    5.0
2           2            10                    5.0
3           3            15                    5.0
4           4            20                    5.0
```

Velocidad media en física

```python
import pandas as pd

# Datos de temperatura
data = {
    'Ciudad': ['Lima', 'Bogota', 'Santiago'],
    'Temperatura Promedio (C)': [18, 14, 12],
    'Precipitacion Promedio (mm)': [20, 100, 40]
}
df = pd.DataFrame(data)

# Ordenar por temperatura
ordenado = df.sort_values(by='Temperatura Promedio (C)', ascending=False)
print(ordenado)
```

```
$ python example_6_pandas_Dataframe_Temperatura.py
     Ciudad  Temperatura Promedio (C)  Precipitacion Promedio (mm)
0      Lima                        18                           20
1    Bogota                        14                          100
2  Santiago                        12                           40
```

Epidemiología - Tasa de Infectados y Salud - Análisis de Pacientes

```python
import pandas as pd
# Datos de infectados
data = {
    'Pais': ['Peru', 'Brasil', 'Mexico'],
    'Poblacion': [33e6, 213e6, 128e6],
    'Infectados': [1e6, 30e6, 5e6]
}
df = pd.DataFrame(data)

# Calcular tasa de infectados
df['Tasa Infectados (%)'] = (df['Infectados']/df
    ['Poblacion']) * 100
print(df)
```

```
$ python example_7_pandas_Dataframe_Epidemiologia.py
     Pais    Poblacion  Infectados  Tasa Infectados (%)
0    Peru   33000000.0   1000000.0             3.030303
1  Brasil  213000000.0  30000000.0            14.084507
2  Mexico  128000000.0   5000000.0             3.906250
```

Tasa de infectados

```python
import pandas as pd

# Datos de pacientes
data = {
    'Paciente': ['P1', 'P2', 'P3'],
    'Edad': [45, 32, 58],
    'Presion Arterial': [120, 130, 140],
    'Diabetes': ['No', 'Si', 'No']
}
df = pd.DataFrame(data)

# Filtrar pacientes con diabetes
filtro = df[df['Diabetes'] == 'Si']
print(filtro)
```

```
$ python example_11_pandas_Dataframe_Salud.py
  Paciente  Edad  Presion Arterial Diabetes
1       P2    32               130       Si
```

Análisis de pacientes

Biología - Análisis de Especie y Bioinformática - Procesamiento de Secuencias Genéticas

```python
import pandas as pd
# Datos de especies
data = {
    'Especie': ['Leon', 'Tigre', 'Oso', 'Lobo'],
    'Poblacion': [1200, 3500, 20000, 15000],
    'Habitat': ['Sabana', 'Selva', 'Bosque', 'Tundra']
}
df = pd.DataFrame(data)

# Filtrar especies con poblacion > 5000
filtro = df[df['Poblacion'] > 5000]
print(filtro)
```

```
$ python example_4_pandas_Dataframe_Biologi.py
  Especie  Poblacion Habitat
2     Oso      20000  Bosque
3    Lobo      15000  Tundra
```

Distribución de especies

```python
import pandas as pd

# Datos de genes
data = {
    'Gen': ['GEN1', 'GEN2', 'GEN3'],
    'Secuencia': ['ATGCTA', 'GCTAGC', 'TTAGGG'],
    'Longitud': [6, 6, 6]
}
df = pd.DataFrame(data)

# Filtrar genes con longitud especifica
filtro = df[df['Longitud'] == 6]
print(filtro)
```

```
$ python example_5_pandas_Dataframe_Bioinformatica.py
    Gen Secuencia  Longitud
0  GEN1    ATGCTA         6
1  GEN2    GCTAGC         6
2  GEN3    TTAGGG         6
```

Ventas - Análisis de Productos y Economía - Análisis de PIB

```python
import pandas as pd

# Datos de ventas
data = {
    'Producto': ['A', 'B', 'C', 'D'],
    'Ventas': [100, 200, 150, 300],
    'Costo Unitario': [10, 15, 20, 25]
}
df = pd.DataFrame(data)

# Calcular ingresos totales
df['Ingresos Totales'] = df['Ventas'] * df['Costo Unitario']
print(df)
```

```
$ python example_8_pandas_Dataframe_Ventas.py
  Producto  Ventas  Costo Unitario  Ingresos Totales
0        A     100              10              1000
1        B     200              15              3000
2        C     150              20              3000
3        D     300              25              7500
```

Análisis de productos más vendidos

```python
import pandas as pd

# Datos de PIB
data = {
    'Pais': ['Peru', 'Chile', 'Colombia'],
    'PIB (USD)': [2.7e11, 3.1e11, 3.5e11],
    'Poblacion': [33e6, 19e6, 51e6]
}
df = pd.DataFrame(data)

# Calcular PIB per capita
df['PIB Per Capita'] = df['PIB (USD)']/df['Poblacion']
print(df)
```

```
$ python example_9_pandas_Dataframe_Analisis_PBI.py
       Pais     PIB (USD)   Poblacion  PIB Per Capita
0      Peru  2.700000e+11  33000000.0     8181.818182
1     Chile  3.100000e+11  19000000.0    16315.789474
2  Colombia  3.500000e+11  51000000.0     6862.745098
```

Análisis de PIB por país

Educación - Análisis de Calificaciones

```python
import pandas as pd

# Datos de calificaciones
data = {
    'Estudiante': ['Ana', 'Juan', 'Maria'],
    'Matematicas': [85, 90, 78],
    'Fisica': [88, 92, 81]
}
df = pd.DataFrame(data)

# Calcular promedio por estudiante
df['Promedio'] = df[['Matematicas', 'Fisica']].mean(axis=1)
print(df)
```

Análisis de calificaciones

```
$ python example_10_pandas_Dataframe_Educacion.py
  Estudiante  Matematicas  Fisica  Promedio
0        Ana           85      88      86.5
1       Juan           90      92      91.0
2      Maria           78      81      79.5
```

Ventajas

Resumen:

- Pandas es una herramienta versátil para el análisis y manipulación de datos estructurados.
- Los ejemplos mostrados demuestran su aplicabilidad en diversos campos como física, biología, economía, salud, etc.
- Facilita tareas como filtrado, cálculos, ordenación y visualización de datos.

Recomendaciones:

- Familiarícese con las funciones básicas de Pandas (DataFrames, Series, filtros, etc.).
- Explore bibliotecas complementarias como NumPy, Matplotlib y Scikit-learn.
- Practique con conjuntos de datos reales para mejorar sus habilidades.

¡Pandas es esencial para cualquier proyecto de ciencia de datos!

¿Qué es Matplotlib?

Matplotlib es una biblioteca de Python para crear gráficos y visualizaciones de alta calidad. Es ampliamente utilizada en ciencia de datos debido a:

- Matplotlib es una herramienta poderosa para la visualización de datos en ciencia de datos.
- Su flexibilidad para crear gráficos personalizados.
- Soporte para múltiples tipos de gráficos (líneas, barras, dispersión, etc.).
- Integración con Pandas y NumPy para análisis de datos.

Es ideal para explorar y comunicar resultados de manera visual.

Ejemplo: Visualización con Matplotlib

Matplotlib es una biblioteca popular para crear gráficos en Python.

```python
import matplotlib.pyplot as plt

# Datos
categorias = ['A', 'B', 'C', 'D']
valores = [10, 20, 15, 25]

# Crear grafico de barras
plt.bar(categorias, valores, color='skyblue')
plt.title('Grafico de Barras')
plt.xlabel('Categorias')
plt.ylabel('Valores')
plt.show()
```

Gráfico de barras con Matplotlib

Visualización con Matplotlib - Calculo de Velocidad Media

```python
import pandas as pd
import matplotlib.pyplot as plt

# Datos de tiempo y posicion
data = {'Tiempo (s)': [0, 1, 2, 3, 4],
        'Posicion (m)': [0, 5, 10, 15, 20]}

df = pd.DataFrame(data)

# Calcular velocidad media
df['Velocidad Media (m/s)'] = df['Posicion (m)'].diff() / df['Tiempo (s)'].diff()
print(df)

posicion = df['Posicion (m)']
tiempo = df['Tiempo (s)']
velocidad = df['Velocidad Media (m/s)']

# Crear grafico
plt.plot(tiempo, posicion, marker='o', color='b')
plt.title('Posicion vs Tiempo')
plt.xlabel('Tiempo (s)')
plt.ylabel('Posicion (m)')
plt.grid(True)
plt.show()
```

Gráfico de barras con Matplotlib

Visualización con Matplotlib - Análisis de Especies

```python
import pandas as pd
import matplotlib.pyplot as plt

# Datos de especies
data = {
    'Especie': ['Leon', 'Tigre', 'Oso', 'Lobo'],
    'Poblacion': [1200, 3500, 20000, 15000],
    'Habitat': ['Sabana', 'Selva', 'Bosque', 'Tundra']
}
df = pd.DataFrame(data)

# Filtrar especies con poblacion > 5000
filtro = df[df['Poblacion'] > 5000]
print(filtro)

especies = df['Especie']
poblacion = df['Poblacion']

# Crear grafico de barras
plt.bar(especies, poblacion, color=['orange', 'blue', 'green'])
plt.title('Poblacion de Especies')
plt.xlabel('Especies')
plt.ylabel('Poblacion')
plt.show()
```

Gráfico de barras con Matplotlib

Visualización con Matplotlib - Epidemiología

```python
import pandas as pd
import matplotlib.pyplot as plt

# Datos de infectados
data = {
    'Pais': ['Peru', 'Brasil', 'Mexico'],
    'Poblacion': [33e6, 213e6, 128e6],
    'Infectados': [1e6, 30e6, 5e6]
}
df = pd.DataFrame(data)

# Calcular tasa de infectados
df['Tasa Infectados (%)'] = (df['Infectados'] / df['Poblacion']) * 100
print(df)

paises = df['Pais']
tasa_infectados = df['Infectados']
poblacion = df['Poblacion']

# Crear grafico de barras
plt.barh(paises, tasa_infectados, color='skyblue')
plt.title('Tasa de Infectados por Pais (%)')
plt.xlabel('Tasa de Infectados (%)')
plt.ylabel('Pais')
plt.show()
```

Gráfico de barras con Matplotlib

Visualización con Matplotlib - Educación

```python
import pandas as pd
import matplotlib.pyplot as plt

# Datos de calificaciones
data = {
    'Estudiante': ['Ana', 'Juan', 'Maria'],
    'Matematicas': [85, 90, 78],
    'Fisica': [88, 92, 81]
}
df = pd.DataFrame(data)

# Calcular promedio por estudiante
df['Promedio'] = df[['Matematicas', 'Fisica']].mean(axis=1)
print(df)

asignaturas = df[['Matematicas', 'Fisica']]
promedios = df['Promedio']
estudiante = df['Estudiante']

# Crear grafico de lineas
plt.plot(estudiante, asignaturas, marker='D', color='brown')
plt.title('Calificaciones Promedio por Asignatura')
plt.xlabel('Asignatura')
plt.ylabel('Promedio')
plt.grid()
plt.show()
```

Gráfico de barras con Matplotlib

¿Qué es Seaborn?

Seaborn es una biblioteca de Python basada en Matplotlib que facilita la creación de gráficos estadísticos atractivos y profesionales. Es ampliamente utilizada en ciencia de datos debido a:

- Seaborn es una herramienta poderosa para la visualización de datos estadísticos en ciencia de datos.
- Su integración con Pandas para análisis de datos.
- Soporte para gráficos avanzados como mapas de calor, diagramas de caja, violín, etc.
- Estilos predefinidos que mejoran la estética de los gráficos.

Es ideal para explorar y comunicar resultados complejos de manera visual.

Visualización con Seaborn

Seaborn es una biblioteca basada en Matplotlib que facilita la creación de gráficos estadísticos.

```python
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Datos
data = pd.DataFrame({
    'X': [1, 2, 3, 4, 5],
    'Y': [2, 4, 1, 8, 7]
})

# Crear grafico de dispersion
sns.scatterplot(x='X', y='Y', data=data)
plt.title('Grafico de Dispersion')
plt.show()
```

Gráfico de dispersión con Seaborn

Visualización con Seaborn - Gráfico de Velocidad vs Tiempo

```python
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Datos de tiempo y posicion
data = {'Tiempo (s)': [0, 1, 2, 3, 4],
        'Posicion (m)': [0, 5, 10, 15, 20]}

df = pd.DataFrame(data)

# Calcular velocidad media
df['Velocidad Media (m/s)'] = df['Posicion (m)'].diff() / df['Tiempo (s)'].diff()
print(df)

posicion = df['Posicion (m)']
tiempo = df['Tiempo (s)']
velocidad = df['Velocidad Media (m/s)']

# Crear grafico
sns.regplot(x=tiempo, y=velocidad, ci=None, color='b')
plt.title('Relacion entre Velocidad y Tiempo')
plt.xlabel('Tiempo (s)')
plt.ylabel('Velocidad (m/s)')
plt.grid()
plt.show()
```

Gráfico de dispersión con Seaborn

Visualización con Seaborn - Población de Especies por Región

```python
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Datos
data = pd.DataFrame({
    'Region': ['Norte', 'Sur', 'Este', 'Oeste'],
    'Leones': [120, 80, 150, 200],
    'Tigres': [90, 70, 110, 130]
})

# Crear grafico
data_melt = data.melt(id_vars='Region', var_name='Especie', value_name='Poblacion')
sns.barplot(data=data_melt, x='Region', y='Poblacion', hue='Especie')
plt.title('Poblacion de Especies por Region')
plt.show()
```

Gráfico de barras con Matplotlib

Visualización con Seaborn - Tasa de Infectados por País

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Datos
data = {
    'Pais': ['Peru', 'Brasil', 'Mexico'],
    'Tasa Infectados (%)': [3.0, 14.2, 3.9]
}
df = pd.DataFrame(data).set_index('Pais')

# Crear grafico
sns.heatmap(df, annot=True, cmap='Blues',
     cbar=False)
plt.title('Tasa de Infectados por Pais (%)')
plt.show()
```

Gráfico de barras con Matplotlib

Visualización con Seaborn - PIB Per Cápita por País

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Datos
# Datos de PIB
data = {
    'Pais': ['Peru', 'Chile', 'Colombia'],
    'PIB (USD)': [2.7e11, 3.1e11, 3.5e11],
    'Poblacion': [33e6, 19e6, 51e6]
}
df = pd.DataFrame(data)

# Calcular PIB per capita
df['PIB Per Capita'] = df['PIB (USD)'] / df['Poblacion']

paises = df['Pais']
pib_per_capita = df['PIB (USD)']

# Crear grafico
sns.barplot(x=pib_per_capita, y=paises, palette='viridis')
plt.title('PIB Per Capita por Pais (USD)')
plt.xlabel('PIB Per Capita (USD)')
plt.ylabel('Pais')
plt.show()
```

Gráfico de barras horizontales

¿Qué es Machine Learning?

El **machine learning** es una rama de la inteligencia artificial que permite a las computadoras aprender de los datos sin ser explícitamente programadas. En ciencia de datos, se utiliza para:

- Clasificación (por ejemplo, predecir categorías).
- Regresión (por ejemplo, predecir valores numéricos).
- Agrupamiento (por ejemplo, segmentar datos).
- Reducción de dimensionalidad.

En esta presentación, exploraremos ejemplos prácticos utilizando Python y bibliotecas como Scikit-learn, Keras, Matplotlib y Seaborn.

Regresión Lineal con Scikit-learn

Scikit-learn es una biblioteca ampliamente utilizada para machine learning.

```python
from sklearn.linear_model import
      LinearRegression
import numpy as np

# Datos
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1.5, 3.1, 4.8, 6.2, 7.9])

# Crear modelo de regresion lineal
modelo = LinearRegression()
modelo.fit(X, y)

# Predecir
prediccion = modelo.predict([[6]])
print("Prediccion para X=6:", prediccion[0])
```

Regresión lineal simple

```
$ python example_1_sklearn_linear_regression.py
Prediccion para X=6: 9.47
```

Clasificación con Scikit-learn

Un ejemplo básico de clasificación usando Scikit-learn.

```python
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Datos
X = np.array([[1, 2], [2, 3], [3, 4], [6, 7], [7, 8]])
y = np.array([0, 0, 0, 1, 1])  # Etiquetas (clases)

# Crear modelo KNN
modelo = KNeighborsClassifier(n_neighbors=3)
modelo.fit(X, y)

# Predecir
prediccion = modelo.predict([[5, 6]])
print("Clase predicha:", prediccion[0])
```

```
$ python example_2_sklearn_KNeighborsClassifier.py
Clase predicha: 1
```

Clasificación con K-Nearest Neighbors

Gráfico de Dispersión con Matplotlib

```python
import matplotlib.pyplot as plt
import numpy as np

# Datos
x = np.random.rand(50)
y = np.random.rand(50)

# Crear grafico
plt.scatter(x, y, color='blue')
plt.title('Grafico de Dispersion')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.show()
```

Gráfico de dispersión

Evaluación de un Modelo de Clasificación

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Cargar datos
X, y = load_iris(return_X_y=True)

# Dividir datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Entrenar modelo
modelo = RandomForestClassifier()
modelo.fit(X_train, y_train)

# Evaluar modelo
y_pred = modelo.predict(X_test)
precision = accuracy_score(y_test, y_pred)
print("Precision del modelo:", precision)
```

Evaluación de un modelo

```
$ python example_6_sklearn_evaluacion.py
Precision del modelo: 0.9666666666666667
```

Machine Learning - Train/Test

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.xlabel(minutes before making a
      purchase)
plt.ylabel(money spent on the purchase)
plt.show()
```

Train/Test - Datos

```python
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

# Separando la data train 80% y test 20%
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

# Subplots
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('train 80% y test 20%s')
# Display the Training Set
ax1.scatter(train_x, train_y)
ax2.grid()
# Display the Testing Set
ax2.scatter(test_x, test_y)
ax1.grid()
plt.show()
```

Train/Test - Datos

Mejor ajuste sería una regresión polinomial, así que tracemos una **línea de regresión polinomial**.

```python
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

# Separando la data train 80% y test 20%
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

# fit the data set
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0, 6, 100)

plt.figure(2)
plt.scatter(train_x, train_y)
plt.plot(myline,mymodel(myline), 'r')
plt.xlabel('minutes before making a purchase')
plt.ylabel('money spent on the purchase')
plt.grid()
plt.show()
```

El resultado sugiere que el conjunto de datos se ajusta a una regresión polinómica, aunque podría darnos algunos resultados extraños si intentamos predecir valores fuera del conjunto de datos.
Observacion : la línea indica que un cliente que pase 6 minutos en la tienda haría una compra por un valor de 200. Eso probablemente sea una señal de sobreajuste (overfitting). **Qué pasa con el puntaje R-cuadrado:** El puntaje R-cuadrado es un buen indicador de qué tan bien se ajusta mi conjunto de datos al modelo.

Train/Test - Datos

```python
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

# Separando la data train 80% y test 20%
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

# fit the data set
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

# find the R2 score when using training data:
r2 = r2_score(train_y, mymodel(train_x))
print("R-squared - training data = %5.3f"% r2)

# find the R2 score when using testing data:
r2 = r2_score(test_y, mymodel(test_x))
print("R-squared - test data = %5.3f"% r2)

# How much money will a buying customer spend, if she or he stays in the shop for 5 minutes?
print("modelo = %5.3f"% mymodel(5))
```

R-squared o R-cuadrado: Mide la relación entre el eje-x y el eje-y, y el valor varía de 0 a 1, donde 0 significa que no hay relación y 1 significa que están totalmente relacionados. El libraria **sklearn** tiene un método llamado **r2_score()** que nos ayudará a encontrar esta relación.

- En este caso, nos gustaría medir la relación entre los minutos que un cliente permanece en la tienda y la cantidad de dinero que gasta. ajustan los datos de entrenamiento **(training data)** usando una regresión polinomial. El resultado 0.799 muestra que existe una relación aceptable.
- Encontrar el puntaje **r2_score()** usar los datos de prueba **test data**. El resultado 0.809 muestra que el modelo se ajusta bien al conjunto de prueba, y estamos seguros de que podemos usar el modelo para predecir valores futuros.
- ¿Cuánto dinero gastará un cliente comprador si permanece en la tienda durante 5 minutos?. El ejemplo predijo que el cliente gastaría 22.88 dólares, lo cual parece corresponder al diagrama

```
$ python example_10_models_predict_outcome_of_events_Fit_the_Data_Set.py
R-squared - training data = 0.799
R-squared - test data = 0.809
modelo = 22.880
```

Train/Test - Datos

```python
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

# Separando la data train 80% y test 20%
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

# fit the data set
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0, 6, 100)

# How much money will a buying customer spend, if she or he stays in the shop for 5 minutes?
print("modelo = %5.3f"% mymodel(5))

plt.figure(2)
plt.scatter(train_x, train_y)
plt.plot(myline,mymodel(myline), 'r')
plt.plot(5,mymodel(5), 'bo', markersize=12)
plt.text(5, mymodel(5), "(5, 22.880)", rotation=40, fontsize=12, verticalalignment='center')
plt.xlabel('minutes before making a purchase')
plt.ylabel('money spent on the purchase')
plt.grid()
plt.show()
```

```
$ python example_10_models_predict_outcome_of_events_Fit_the_Data_Set.py
R-squared - training data = 0.799
R-squared - test data = 0.809
modelo = 22.880
```

Arbol de decisiones de profundidad 3 - Datos

Caso de estudio : ¿De qué color es la flor?
Imagina que estás estudiando un tipo de flor que puede ser de dos colores: azul o rojo. Has medido la longitud y el ancho de los pétalos de varias flores y quieres construir un modelo para predecir el color de una flor basándote en estas medidas.

Observa la data

	X1	X2	X3
0	44	91	1
1	99	16	0
2	48	2	0
3	56	98	1
4	47	45	0
5	37	89	1
6	35	24	0
7	28	81	1
8	46	2	0
9	68	32	0
10	75	40	0
11	52	44	0
12	61	37	0
13	69	85	1
14	46	16	0
15	37	14	0
16	65	52	0
17	37	93	1
18	16	65	0
19	71	25	0
20	91	16	0
21	37	100	1
22	9	15	0
23	61	76	0
24	26	91	1

Variables Predictoras
X1 = Ancho de petalo (mm)
X2 = Longitud del petalo (mm)

Variable Objetivo
X3 = Color de la flor (azul o rojo)

Arbol de decisiones de profundidad 3

```python
import matplotlib.pyplot as plt
import pandas as pd

# Datos
data = {
    'X1': [44, 99, 48, 56, 47, 37, 35, 28, 46, 68, 75, 52, 61, 69, 46, 37, 65, 37, 16, 71, 91, 37, 9, 61, 26],
    'X2': [91, 16, 2, 98, 45, 89, 24, 81, 2, 32, 40, 44, 37, 85, 16, 14, 52, 93, 65, 25, 16, 100, 15, 76, 91],
    'X3': [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1]
}
# Crear DataFrame
df = pd.DataFrame(data)

# Asignar colores basados en X3
colors = ['red' if x == 1 else 'blue' for x in df['X3']]

# Crear grafico
plt.figure(figsize=(10, 6))
plt.scatter(df['X1'], df['X2'], c=colors, s=100, alpha=0.7, edgecolor='k')

# Etiquetas y titulo
plt.title('Ancho vs Longitud del Petalo', fontsize=16)
plt.xlabel('Ancho del Petalo (mm)', fontsize=14)
plt.ylabel('Longitud del Petalo (mm)', fontsize=14)
```

```python
# Leyenda
plt.scatter([], [], c='red', label='Flor Roja', s=100, edgecolor='k')
plt.scatter([], [], c='blue', label='Flor Azul', s=100, edgecolor='k')
plt.legend(fontsize=12)

# Mostrar grafico
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

Variables Predictoras	Variable Objetivo
X1 = Ancho de petalo (mm)	X3 = Color de la flor (azul o rojo)
X2 = Longitud del petalo (mm)	

Arbol de decisiones de profundidad 3

```python
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import tree
import matplotlib.pyplot as plt

# Datos
data = {
    'X1': [44, 99, 48, 56, 47, 37, 35, 28, 46, 68, 75, 52, 61, 69, 46, 37, 65, 37, 16, 71, 91, 37, 9, 61, 26],
    'X2': [91, 16, 2, 98, 45, 89, 24, 81, 2, 32, 40, 44, 37, 85, 16, 14, 52, 93, 65, 25, 16, 100, 15, 76, 91],
    'X3': [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1]
}
# Crear DataFrame
df = pd.DataFrame(data)

# Variables predictoras y objetivo
X = df[['X1', 'X2']]  # Ancho y longitud del petalo
y = df['X3']          # Color de la flor (1 = azul, 0 = rojo)

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Crear y entrenar el modelo de arbol de decision
modelo = DecisionTreeClassifier(max_depth=3, random_state=42)
modelo.fit(X_train, y_train)

# Realizar predicciones
y_pred = modelo.predict(X_test)

# Evaluar el modelo
precision = accuracy_score(y_test, y_pred)
print(f"Precision del modelo: {precision:.2f}")

# Visualizar el arbol de decision
plt.figure(figsize=(12, 8))
tree.plot_tree(modelo, feature_names=['Ancho del Petalo (X1)', 'Longitud del Petalo (X2)'],
               class_names=['Rojo', 'Azul'], filled=True)
plt.title("Arbol de Decision (Profundidad = 3)")
plt.show()
```

Arbol de decisiones de profundidad 3

Consideraciones Importantes

- **Preprocesamiento de datos:** Limpieza y transformación son pasos clave antes de cualquier análisis.
- **Visualización:** Ayuda a comprender mejor los datos y comunicar resultados.
- **Modelos de ML:** Elija el modelo adecuado según el problema (regresión, clasificación, clustering, etc.).
- **Evaluar modelos:** Use métricas como precisión, recall o MSE para evaluar el rendimiento.

Recomendaciones Prácticas

- Familiarícese con las principales bibliotecas (Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn).
- Aprenda a manejar grandes volúmenes de datos con herramientas como Dask o PySpark.
- Documente su código y resultados para reproducibilidad.
- Explore frameworks avanzados como TensorFlow o PyTorch para deep learning.

Conclusión

Resumen:

- Python es una herramienta poderosa para ciencia de datos gracias a sus bibliotecas especializadas.
- Pandas facilita el análisis de datos estructurados.
- Matplotlib y Seaborn permiten crear visualizaciones claras y atractivas.
- Scikit-learn es ideal para implementar modelos de machine learning.

Recomendaciones:

- Practique con conjuntos de datos reales para mejorar sus habilidades.
- Manténgase actualizado con nuevas bibliotecas y técnicas.
- Colabore con otros profesionales en proyectos de ciencia de datos.

¡Python es una excelente elección para explorar y resolver problemas de ciencia de datos!

Gracias por su atención

¿Preguntas?