23/09/2025

Herramientas de Ánalisis de datos

Extracción de Conocimiento en Bases de Datos

Luis Eduardo Aguilar Sarabia

IDGS91N

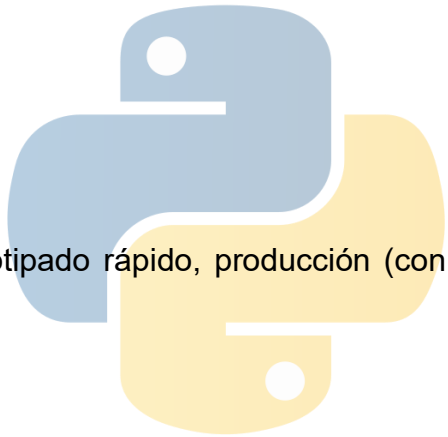
Docente: Luis Enrique Mascote Cano

Introducción

Conocer los lenguajes y las bibliotecas utilizadas en análisis de datos es importante porque facilita elegir la herramienta adecuada para cada tarea (limpieza, modelado, visualización, despliegue) y optimizar tiempo y recursos. Cada lenguaje ofrece ventajas distintas: algunos priorizan la rapidez de prototipado y la riqueza del ecosistema (p. ej. Python, R), otros aportan rendimiento y escalabilidad en producción (p. ej. Scala con Spark, SQL para acceso a bases de datos), y otros buscan un equilibrio entre rendimiento y expresividad numérica (Julia). Además, las bibliotecas especializadas (p. ej. pandas, dplyr, DataFrames.jl) encapsulan buenas prácticas y patrones comunes, reduciendo la probabilidad de errores y acelerando el desarrollo.

Este reporte ayuda a comparar más de cinco lenguajes populares en análisis de datos, presentar sus bibliotecas clave y ofrecer mini-ejemplos “Hola datos” para empezar a trabajar con un CSV. La idea es dar una vista práctica y aplicada, útil tanto para estudiantes como para profesionales que deben seleccionar la pila tecnológica para proyectos pequeños o a gran escala.

Python



Descripción general

- Paradigma: Interpretado, tipado dinámico.
- Ámbito de uso principal: Ciencia de datos, prototipado rápido, producción (con contenedores), integración back-end.

Bibliotecas y frameworks clave

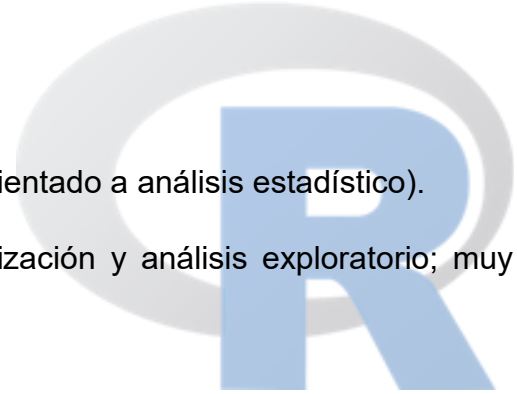
- **pandas** — manipulación y análisis de datos con DataFrames. *Caso de uso:* cargar, limpiar y agrupar datos de ventas para análisis exploratorio.
- **NumPy** — arreglos multidimensionales y operaciones numéricas eficientes. *Caso de uso:* operaciones vectorizadas y preprocesamiento numérico para ML.
- **scikit-learn** — modelos de machine learning clásicos (regresión, clasificación, clustering). *Caso de uso:* entrenamiento rápido de un clasificador y pipeline de validación.

Mini-ejemplo “Hola datos” (Python / pandas)

```
import pandas as pd

df = pd.read_csv("datos_ejemplo.csv")
print(df.head())        # primeras 5 filas
print(df.describe())    # resumen estadístico
```

R



Descripción general

- Paradigma: Interpretado, tipado dinámico (orientado a análisis estadístico).
- Ámbito de uso principal: Estadística, visualización y análisis exploratorio; muy usado en investigación y bioestadística.

Bibliotecas y frameworks clave

- **tidyverse (readr, dplyr, ggplot2)** — conjunto coherente para importación, manipulación y visualización. *Caso:* pipeline de limpieza con dplyr y gráficos con ggplot2.
- **data.table** — manipulación de tablas muy eficiente en memoria, útil para grandes conjuntos de datos. *Caso:* agregaciones rápidas sobre millones de filas.
- **caret / tidymodels** — frameworks para entrenamiento y validación de modelos. *Caso:* comparación y validación cruzada de modelos.

Mini-ejemplo “Hola datos” (R / tidyverse)

```
library(readr)
library(dplyr)

df <- read_csv("datos_ejemplo.csv")
print(head(df))
glimpse(df)
```

Scala (con Apache Spark)

Descripción general

- Paradigma: Compilado a bytecode (JVM), tipado estático, multiparadigma (funcional + OO).
- Ámbito de uso principal: Big Data y procesamiento distribuido (con Apache Spark), aplicaciones de backend de alto rendimiento.

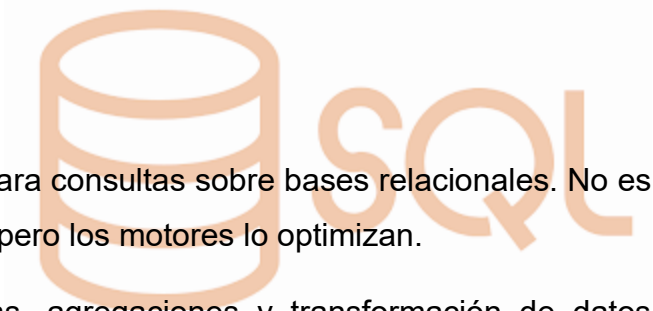
Bibliotecas y frameworks clave

- **Apache Spark (Spark SQL / DataFrame API)** — procesamiento distribuido de datos, ETL, ML en escala. *Caso:* análisis de logs a petabyte usando clusters.
- **Breeze** — álgebra lineal y operaciones numéricas en Scala. *Caso:* cálculos numéricos avanzados para modelos personalizados.
- **Spark MLlib** — librería de machine learning distribuido. *Caso:* entrenamiento de modelos sobre grandes volúmenes de datos.

Mini-ejemplo "Hola datos" (Spark Scala)

```
// ejecutarlo en spark-shell o en una aplicación Spark
val df =
spark.read.option("header","true").option("inferSchema","true").csv("datos_ejemplo.csv"
)
df.show(5)
df.describe().show()
```

SQL



Descripción general

- Paradigma: Lenguaje declarativo para consultas sobre bases relacionales. No es "compilado" en sentido tradicional, pero los motores lo optimizan.
- Ámbito de uso principal: consultas, agregaciones y transformación de datos almacenados en bases de datos (OLTP y OLAP).

Herramientas / engines y funcionalidades clave

- **PostgreSQL** — DBMS relacional robusto, funciones analíticas y extensibilidad. *Caso:* almacén de datos relacional con vistas y funciones almacenadas.
- **SQLite** — base embebida, ideal para prototipos y datasets pequeños. *Caso:* análisis local y pruebas con CSV importados.
- **SQLAlchemy (Python)** — ORM y motor SQL para integración con aplicaciones. *Caso:* ejecutar consultas parametrizadas desde Python y materializar resultados en DataFrames.

Mini-ejemplo "Hola datos" (SQL)

```
-- Ver primeras 5 filas de la tabla ventas
SELECT *
FROM ventas
LIMIT 5;
```

(En SQLite se podría importar el CSV con .import datos_ejemplo.csv ventas y luego ejecutar la consulta anterior.)

Julia

Descripción general

- Paradigma: Compilado a JIT (Just-In-Time), diseñado para alto rendimiento; tipado dinámico con posibilidad de tipos estáticos.
- Ámbito de uso principal: computación numérica, análisis científico y prototipado de algoritmos que requieren velocidad cercana a C/Fortran.

Bibliotecas y frameworks clave

- **DataFrames.jl** — estructuras tipo DataFrame y operaciones tabulares. *Caso:* manipulación similar a pandas pero con rendimiento mejorado en ciertos casos.
- **CSV.jl** — lectura/escritura rápida de archivos CSV. *Caso:* cargar grandes CSV eficientemente.
- **Flux.jl / MLJ.jl** — ecosistema para machine learning y deep learning en Julia. *Caso:* prototipado de modelos numéricos y redes neuronales con alto rendimiento.

Mini-ejemplo “Hola datos” (Julia)

```
using CSV, DataFrames

df = CSV.read("datos_ejemplo.csv", DataFrame)
first(df, 5)     # primeras 5 filas
describe(df)     # resumen
```

Java (opcional)

Descripción general

- Paradigma: Compilado a bytecode (JVM), tipado estático, orientado a objetos.
- Ámbito de uso principal: aplicaciones de producción, back-end robusto, integración con ecosistemas de big data (Hadoop, Spark Java API).

Bibliotecas y frameworks clave

- **Apache Commons CSV / OpenCSV** — lectura y escritura de CSV en Java. *Caso:* ingestión de archivos CSV en aplicaciones empresariales.
- **Apache Spark (Java API)** — procesamiento distribuido usando Java cuando se necesita integración con un stack Java. *Caso:* pipelines ETL escalables en clusters.
- **Deeplearning4j / Weka** — bibliotecas de ML en Java. *Caso:* modelos embebidos en aplicaciones corporativas.

Mini-ejemplo “Hola datos” (Java + OpenCSV)

```
// Simplificado; necesidad de añadir dependencias OpenCSV
import com.opencsv.CSVReader;
import java.io.FileReader;

CSVReader reader = new CSVReader(new FileReader("datos_ejemplo.csv"));
String[] fila;
int i = 0;
while ((fila = reader.readNext()) != null && i < 5) {
    System.out.println(String.join(", ", fila));
    i++;
}
reader.close();
```

Conclusión

Hay claras similitudes y diferencias entre los lenguajes cubiertos. En términos de **facilidad de uso**, Python y R sobresalen: su sintaxis es concisa, cuentan con ecosistemas ricos y abundante documentación y ejemplos, lo que acelera el inicio de proyectos y la iteración exploratoria. R destaca en análisis estadístico y visualización “a la carta” (ggplot2, tidyverse), mientras que Python ofrece mayor versatilidad para integrar análisis en aplicaciones y despliegues (APIs, microservicios).

Respecto a **rendimiento**, Scala (con Spark) y Julia tienden a ofrecer mejor escalabilidad y velocidad para cargas grandes o cálculos numéricos intensivos. Scala/Spark está pensado para procesamiento distribuido en clusters y es la opción dominante cuando se trabaja con big data. Julia ofrece rendimiento cercano al código compilado para cálculo numérico y puede ser excelente cuando se desarrolla algoritmos científicos que necesitan velocidad.

En cuanto al ecosistema de bibliotecas, Python tiene probablemente la oferta más amplia y madura para todo el flujo de trabajo (ingesta, visualización, ML, deep learning, despliegue). R mantiene una comunidad muy fuerte en estadística y bio-ciencias. SQL sigue siendo indispensable para extracción y transformación en cualquier flujo que implique bases de datos relacionales; su ubiquidad lo hace siempre relevante.

Recomendaciones prácticas

- Proyectos de análisis ligero / exploratorio (notebooks, prototipos): Python o R. Son rápidos de aprender, tienen muchas librerías y permiten llegar a resultados útiles con poco código.
- Proyectos de producción a gran escala (ETL, análisis distribuido, pipelines en cluster): Scala + Apache Spark o bien arquitecturas basadas en SQL (warehouses como PostgreSQL, Redshift, BigQuery según el entorno) combinadas con Spark o herramientas ETL. Java también es apropiado cuando la organización tiene una base de código Java y necesita integrar análisis en sistemas productivos.

- Proyectos numéricos científicos o cuando se requiere máximo rendimiento en código nuevo: considerar Julia (por rendimiento) o C/C++ para componentes críticos, con Julia como opción intermedia más productiva.

Referencias

https://docs.python.org/3/

https://www.r-project.org/

https://docs.scala-lang.org/

https://www.postgresql.org/docs/

https://docs.julialang.org/en/v1/