EXTRACCIÓN DE CONOCIMIENTOS EN BASES DE DATOS

ING. LUIS ENRIQUE MASCOTE CANO

Universidad Tecnológica de Chihuahua

LENGUAJES Y BIBLIOTECAS PARA ANÁLISIS Y PROCESAMIENTO DE DATOS

Lic. Ricardo Hernández Martínez

Fecha de Entrega: 23/SEPTIEMBRE/2025

Contenido

Introducción

En proyectos de inteligencia artificial (IA), aprendizaje automático (ML), minería de datos (DM) y Big Data, la elección del lenguaje de programación y su ecosistema de bibliotecas/frameworks condiciona la productividad del equipo, la facilidad para prototipar, la escalabilidad y el rendimiento en producción. Conocer los lenguajes y sus librerías clave permite seleccionar la combinación adecuada según el tamaño del dato, la complejidad del modelo, la necesidad de ejecución distribuida y la infraestructura disponible (GPU/TPU, clústeres, nube). Este reporte busca ofrecer una visión práctica y comparativa de los lenguajes más utilizados, sus bibliotecas emblemáticas y ejemplos “Hola datos” para cada uno.

Objetivos del reporte

- Identificar los lenguajes más relevantes para manipulación y modelado de datos (mínimo 5).
- Describir paradigmas de cada lenguaje, su ámbito de uso y bibliotecas/frameworks clave (2–3 por lenguaje).
- Mostrar un mini-ejemplo que cargue un CSV y muestre las primeras filas (“Hola datos”).
- Ofrecer una conclusión comparativa y recomendaciones para proyectos ligeros y de producción a gran escala.
- Proveer referencias en formato APA.

Python

Interpretado, multi-paradigma, tipado dinámico. Usado en ciencia de datos, ML/IA, back-end.

Bibliotecas y frameworks clave:

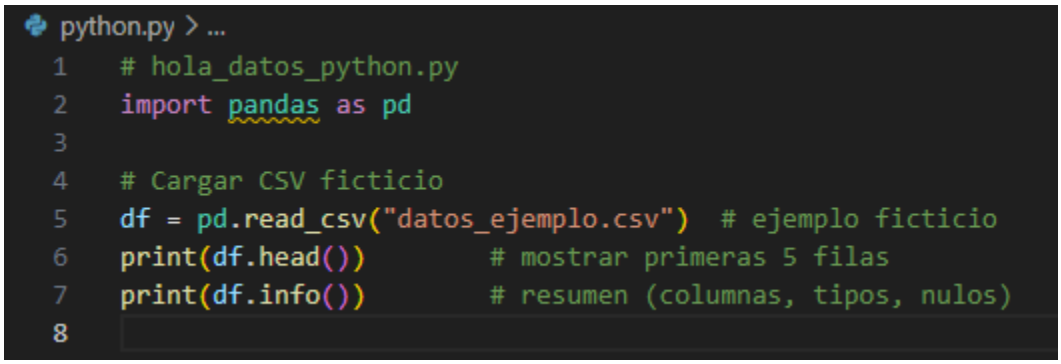
- pandas: Manipulación de datos tabulares (DataFrames), lectura/escritura de CSV/Parquet.

- NumPy: Arrays n-dimensionales y operaciones vectorizadas.

- scikit-learn: Modelos clásicos de ML (clasificación, regresión, clustering).

- TensorFlow/PyTorch: Frameworks de deep learning con soporte para GPU/TPU.

Ejemplo:



R

Interpretado, orientado a vectores, tipado dinámico. Usado en estadística, análisis exploratorio y visualización.

Bibliotecas y frameworks clave:

- tidyverse: Colección de paquetes para manipulación, visualización y entrada de datos.

- caret: Entrenamiento, selección y validación de modelos de ML.

- mlr3: Framework moderno y escalable para ML en R.

Ejemplo:

r.r

```r
library(readr)
library(dplyr)
df <- read_csv("datos_ejemplo.csv")
print(head(df))
glimpse(df)
```

Scala (Spark)

Compilado, tipado estático, multiparadigma. Usado en Big Data distribuido.

Bibliotecas y frameworks clave:

- Apache Spark: Procesamiento distribuido batch/streaming, DataFrames y SQL.

- MLlib: Biblioteca de ML distribuida en Spark.

Ejemplo:

scala.scala

```scala
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder.appName("HolaDatos").getOrCreate()
val df = spark.read.option("header","true").option("inferSchema","true").csv("datos_ejemplo.csv")
df.show(5)
df.printSchema()
spark.stop()
```

SQL

Lenguaje declarativo para consultas relacionales. Usado en bases de datos OLTP/OLAP.

Bibliotecas y frameworks clave:

- PostgreSQL: Base de datos robusta para consultas analíticas.

- SQLite: Base ligera para prototipos y datasets pequeños.

- SQLAlchemy: Librería para interactuar con bases SQL desde Python.

Ejemplo:

```
sqlite3 datos.db
sqlite> .mode csv
sqlite> .import datos_ejemplo.csv tabla_ejemplo
sqlite> SELECT * FROM tabla_ejemplo LIMIT 5;
```

Julia

Interpretado/compilado JIT, tipado dinámico con anotaciones. Diseñado para cómputo científico.

Bibliotecas y frameworks clave:

- DataFrames.jl: Manipulación tabular.

- CSV.jl: Lectura/escritura eficiente de CSV.

- Flux.jl: Deep learning en Julia.

Ejemplo:

julia.jl

```julia
using CSV, DataFrames
df = CSV.read("datos_ejemplo.csv", DataFrame)
first(df, 5)
describe(df)
```

Java

Compilado a bytecode, tipado estático, orientado a objetos. Usado en producción a gran escala.

Bibliotecas y frameworks clave:

- Weka: Biblioteca clásica de minería de datos.

- Deeplearning4j: Framework de deep learning para JVM.

- Apache Flink/Beam: Procesamiento distribuido y streaming.

Ejemplo:

java.java

```java
import weka.core.converters.CSVLoader;
import weka.core.Instances;
import java.io.File;
CSVLoader loader = new CSVLoader();
loader.setSource(new File("datos_ejemplo.csv"));
Instances data = loader.getDataSet();
System.out.println(data.toString());
```

Conclusión

Todos los lenguajes presentados permiten cargar, manipular y analizar datos, pero difieren en su facilidad de uso, ecosistema y escalabilidad.

- Python y R son más adecuados para análisis ligero y prototipado rápido.
- Scala (con Spark) y Java son recomendables para entornos de producción y Big Data.
- Julia ofrece un balance entre rendimiento numérico y expresividad, siendo útil en proyectos científicos.

En síntesis, Python se recomienda para proyectos exploratorios y escalables hacia producción, Scala/Java para grandes volúmenes de datos, y Julia para cómputo científico de alto rendimiento.

Referencias bibliográficas

The pandas development team. (s. f.). pandas: read_csv — PyData. https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

NumPy Developers. (s. f.). NumPy user guide. https://numpy.org/doc/stable/user/

Pedregosa, F., et al. (s. f.). scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/user_guide.html

TensorFlow Team. (2024). TensorFlow API documentation. https://www.tensorflow.org/api_docs

PyTorch Documentation. (s. f.). PyTorch docs. https://docs.pytorch.org/