UNIVERSIDAD TECNOLÓGICA DE CHIHUAHUA

DESARROLLO Y GESTIÓN DE SOFTWARE

DESARROLLO WEB INTEGRAL

I.4. Reporte de investigación de los lenguajes y bibliotecas para análisis y procesamiento de datos.

DOCENTE:

ING. LUIS ENRIQUE MASCOTE CANO

PRESENTA:

DARON TARÍN GONZÁLEZ

GRUPO:

IDGS91N

Chihuahua, Chih., 23 de septiembre de 2025

Contenido

Introducción 3
1.Importancia de conocer los lenguajes y bibliotecas 3
1.1 Objetivos del reporte 3
2. Python 4
3. R 4
4. Scala 5
5. SQL 6
6. Julia 7
7. Java 7
8. MATLAB 8
9. Astro Framework (extra) 9
Conclusión comparativa 11
Referencias 12

Introducción

En la actualidad, la **ciencia de datos y la inteligencia artificial** se han consolidado como ejes fundamentales de innovación tecnológica. La disponibilidad masiva de datos ha impulsado la necesidad de contar con **lenguajes de programación y bibliotecas especializadas** que permitan realizar tareas de extracción, transformación, carga (ETL), análisis exploratorio, modelado estadístico, aprendizaje automático y despliegue en producción (Python Software Foundation, 2025).

1. Importancia de conocer los lenguajes y bibliotecas

Conocer estos lenguajes y sus ecosistemas es esencial por cuatro razones principales:

- **Eficiencia y productividad:** seleccionar la herramienta adecuada permite reducir costos de tiempo y recursos.
- **Escalabilidad:** algunos lenguajes son más aptos para proyectos de gran escala (ejemplo: Scala con Spark), mientras que otros son ideales para análisis exploratorios rápidos (ejemplo: R).
- **Comunidad y soporte:** un lenguaje con amplia comunidad garantiza documentación, actualizaciones y soporte técnico (Harris et al., 2020).
- **Especialización:** algunos lenguajes están diseñados para campos específicos, como la estadística (R) o el cómputo numérico de alto rendimiento (Julia).

1.1 Objetivos del reporte

- Identificar los lenguajes de programación más utilizados en análisis de datos.
- Documentar sus bibliotecas y frameworks clave.
- Mostrar ejemplos prácticos de uso.
- Comparar las fortalezas y limitaciones de cada lenguaje.
- Recomendar escenarios de aplicación según la escala del proyecto.

2. Python

Paradigma: Interpretado; tipado dinámico.
Ámbito de uso: Ciencia de datos integral, desde análisis exploratorio hasta despliegue en producción.

Bibliotecas clave:

- **pandas:** manipulación de datos tabulares (McKinney, 2010).
- **NumPy:** operaciones numéricas de alto rendimiento (Harris et al., 2020).
- **scikit-learn:** machine learning clásico (Pedregosa et al., 2011).
- **Matplotlib / Seaborn:** visualización de datos (Matplotlib Developers, 2025).

Mini-ejemplo "Hola datos":

Aquí se importa **pandas** y se carga un archivo CSV en un **DataFrame**. El método head() muestra las primeras 5 filas. Es el flujo típico en Python: cargar datos → explorarlos rápidamente.

3. R

Paradigma: Interpretado; tipado dinámico.
Ámbito de uso: Estadística aplicada y visualización de datos.

Bibliotecas clave:

- **dplyr:** manipulación de datos (Wickham et al., 2023).

- **ggplot2:** visualización avanzada (Wickham, 2016).
- **tidyr:** limpieza y reestructuración de datos.
- **caret:** framework para machine learning.

Mini-ejemplo "Hola datos":

```r
library(readr)
df <- read_csv("datos.csv")
head(df)
```

En este ejemplo se usa la librería **readr** para leer un CSV en un **data frame** de R. Con head(df) se imprimen las primeras filas. Es el equivalente directo a pandas en Python.

4. Scala

Paradigma: Compilado; tipado estático; multiparadigma.
Ámbito de uso: Big Data y procesamiento distribuido.

Bibliotecas clave:

- **Apache Spark:** procesamiento distribuido en clúster (Apache Software Foundation, 2025).
- **MLlib:** machine learning a gran escala.
- **Delta Lake:** almacenamiento confiable de datos.

Mini-ejemplo “Hola datos”:

```scala
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder.appName("HolaDatos").getOrCreate()
val df = spark.read.option("header","true").csv("datos.csv")
df.show(5)

```

Este código crea una **sesión de Spark**, carga un archivo CSV como un **DataFrame distribuido** y muestra las primeras 5 filas. Representa la forma en que se manejan grandes volúmenes de datos en un clúster de Big Data.

5. SQL

Paradigma: Declarativo.
Ámbito de uso: Consultas y análisis en bases relacionales.

Motores clave:

- **PostgreSQL:** base relacional avanzada (PostgreSQL Global Development Group, 2025).
- **DuckDB:** OLAP embebido rápido (DuckDB Foundation, 2025).
- **SQLite:** motor ligero para prototipos.

Mini-ejemplo “Hola datos”:

```sql
SELECT * FROM datos
LIMIT 5;

```

Una consulta SQL sencilla que selecciona todos los registros de la tabla datos, pero limitando la salida a las **primeras 5 filas**. Es la manera estándar de "asomarse" a un dataset en una base relacional.

6. Julia

Paradigma: Compilado JIT; tipado dinámico con alto rendimiento.
Ámbito de uso: Computación científica y machine learning.

Bibliotecas clave:

- **DataFrames.jl:** manipulación de datos (JuliaData, 2025).
- **CSV.jl:** lectura/escritura de CSV.
- **Flux.jl:** deep learning en Julia.
- **StatsModels.jl:** modelado estadístico.

Mini-ejemplo "Hola datos":

En Julia se cargan dos paquetes: **CSV.jl** para leer el archivo y **DataFrames.jl** para manipularlo. first(df, 5) devuelve las primeras 5 filas, similar a head() en R o Python

7. Java

Paradigma: Compilado; tipado estático.
Ámbito de uso: Producción y sistemas distribuidos.

Bibliotecas clave:

- **Apache Spark (API Java):** ETL y ML distribuidos.
- **Hadoop:** almacenamiento distribuido (Apache Software Foundation, 2025).

- **Deeplearning4j:** deep learning en la JVM (Eclipse Deeplearning4j, 2025).

Mini-ejemplo "Hola datos":

```java
import org.apache.spark.sql.*;

SparkSession spark = SparkSession.builder().appName("HolaDatos").getOrCreate();
Dataset<Row> df = spark.read().option("header","true").csv("datos.csv");
df.show(5);

```

Muy parecido al ejemplo en Scala, pero en la sintaxis de Java. Se crea una sesión de Spark, se lee el CSV en un **Dataset<Row>** y se muestran las primeras 5 filas. Es útil en entornos donde la infraestructura ya está basada en Java.

8. MATLAB

Paradigma: Interpretado, orientado a matrices.
Ámbito de uso: Computación numérica y prototipado.

Bibliotecas clave:

- **Statistics and Machine Learning Toolbox:** modelos estadísticos.
- **Database Toolbox:** conexión a bases de datos.
- **Parallel Computing Toolbox:** paralelización (MathWorks, 2025).

Mini-ejemplo "Hola datos":

```java
Dataset<Row> df = spark.read().option("header","true").csv("datos.csv");
df.show(5);

```

MATLAB usa la función readtable para leer un CSV en una tabla. Luego head(T,5) devuelve las primeras 5 filas. Es la forma típica de comenzar el análisis numérico en MATLAB.

9. Astro Framework (extra)

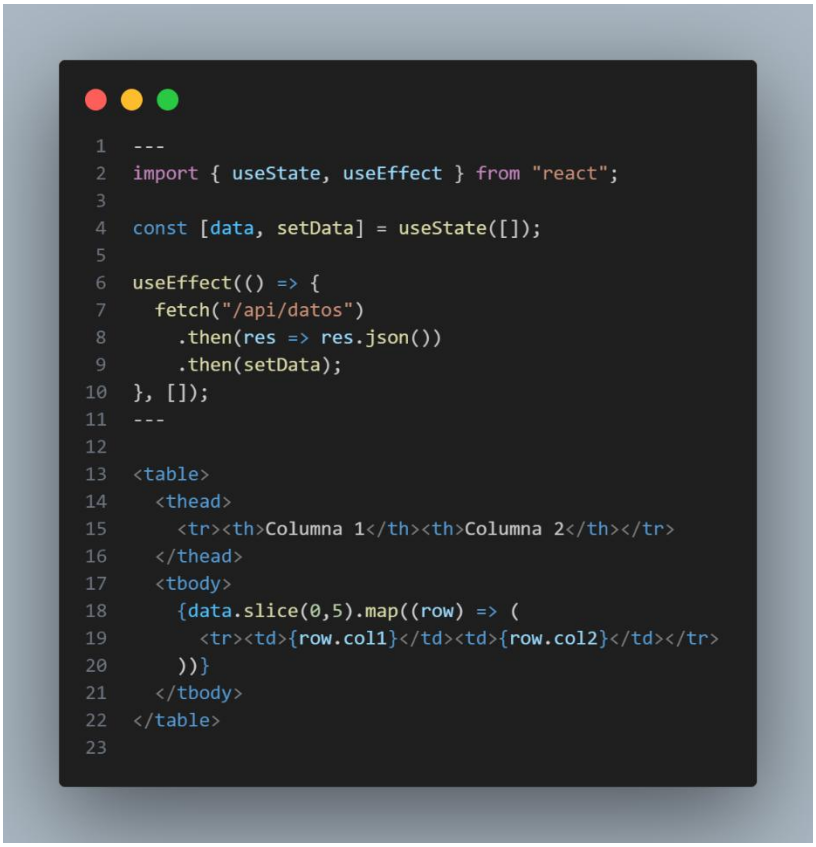
Paradigma: Framework basado en JavaScript/TypeScript.
Ámbito de uso: Desarrollo web moderno para visualización y publicación de datos.

Características clave:

- **Renderizado híbrido (SSR, SSG):** optimización del rendimiento web.
- **Integración con bibliotecas de visualización:** D3.js, Chart.js, Recharts.
- **Arquitectura de islas de componentes:** eficiente para dashboards interactivos.
- **Compatibilidad multiplataforma:** combina React, Vue, Svelte en un mismo proyecto (Astro Build, 2025).

Caso de uso:
Astro permite crear dashboards para visualizar resultados de análisis en tiempo real, consumiendo datos procesados en Python o R y mostrándolos en gráficos dinámicos.

Mini-ejemplo “Hola datos”:



```
---
import { useState, useEffect } from "react";

const [data, setData] = useState([]);

useEffect(() => {
  fetch("/api/datos")
    .then(res => res.json())
    .then(setData);
}, []);
---

<table>
  <thead>
    <tr><th>Columna 1</th><th>Columna 2</th></tr>
  </thead>
  <tbody>
    {data.slice(0,5).map((row) => (
      <tr><td>{row.col1}</td><td>{row.col2}</td></tr>
    ))}
  </tbody>
</table>

```

Aquí, en un **componente de Astro con React**, se hace una petición a un backend (/api/datos) que entrega datos en JSON (previamente procesados de un CSV). Se muestran las primeras 5 filas en una tabla HTML. Representa cómo **publicar los resultados del análisis en la web**.

Conclusión comparativa

La comparación entre los lenguajes y frameworks analizados muestra que:

- **Python y R**: sobresalen en facilidad de uso, ecosistema de bibliotecas y análisis exploratorio.
- **Scala y Java**: indispensables en **Big Data y producción**, con Apache Spark y Hadoop.
- **SQL**: sigue siendo el lenguaje universal para consultas y transformaciones tabulares.
- **Julia**: destaca en proyectos de cómputo numérico de alto rendimiento.
- **MATLAB**: útil en investigación y prototipado en ingeniería.
- **Astro Framework**: no compite en procesamiento, pero fortalece la **presentación y comunicación de resultados**, permitiendo que los análisis lleguen de manera interactiva a usuarios finales.

Recomendaciones:

- Para análisis ligeros y rápidos: **Python, R, SQL/DuckDB + Astro para dashboards**.
- Para proyectos de producción masivos: **Scala/Java con Spark**, complementado con Astro como capa de visualización.
- Para proyectos científicos de alto rendimiento: **Julia + Astro** para integrar cómputo y comunicación.

Referencias

Apache Software Foundation. (2025). Apache Hadoop. https://hadoop.apache.org/

Apache Software Foundation. (2025). Apache Spark documentation. https://spark.apache.org/docs/latest/

Astro Build. (2025). Astro documentation. https://docs.astro.build/

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. SIAM Review, 59(1), 65–98. https://doi.org/10.1137/141000671

CSV.jl Developers. (2025). CSV.jl documentation. https://csv.juliadata.org/

DuckDB Foundation. (2025). DuckDB documentation. https://duckdb.org/docs/

Eclipse Deeplearning4j. (2025). DL4J documentation. https://deeplearning4j.org/

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

JuliaData. (2025). DataFrames.jl documentation. https://dataframes.juliadata.org/

MathWorks. (2025). MATLAB & Toolboxes documentation. https://www.mathworks.com/help/

Matplotlib Developers. (2025). Matplotlib documentation. https://matplotlib.org/

McKinney, W. (2010). Data structures for statistical computing in Python. Proceedings of the 9th Python in Science Conference (SciPy 2010). https://pandas.pydata.org/

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://scikit-learn.org/

PostgreSQL Global Development Group. (2025). PostgreSQL documentation. https://www.postgresql.org/docs/

Python Software Foundation. (2025). Python documentation. https://docs.python.org/

R Core Team. (2025). R: A language and environment for statistical computing. R Foundation for Statistical Computing. https://www.r-project.org/

Wickham, H. (2016). ggplot2: Elegant graphics for data analysis. Springer. https://ggplot2.tidyverse.org/

Wickham, H., François, R., Henry, L., & Müller, K. (2023). dplyr: A grammar of data manipulation. https://dplyr.tidyverse.org/