Scala en el desarrollo de la analítica en big data

Scala in the development of big data analytics

D. Acuñar*, A. De La Rosa*, K. García*, A. Pico*, D. Rovira* & D. Heredia**
{didier.acuña, adrian.delarosa, kevin.garcia, adrian.pico, daniel.rovira} @unisimon.edu.co – { dianahv } @unisimonbolivar.edu.co
*Estudiante de Ingeniería de sistemas **Profesor investigador del grupo Ingebiocaribe
Universidad Simón Bolívar, Barranquilla-Colombia.

Resumen Scala es uno de los lenguajes que últimamente ha ido agarrando fama en la analítica big data. Este lenguaje se ha diseñado para adaptarse a los requisitos futuros y, por lo tanto, se denomina Lenguaje escalable (Scala). Scala no solo es un lenguaje totalmente orientado a objetos, sino también un lenguaje totalmente funcional.

Scala es un lenguaje que ya lleva más de una década desde que hizo su primera aparición, pero este se ha visto opacado por grandes lenguajes como Python, R, entre otros. Sin embargo, en los últimos años este ha comenzado a volverse más popular entre la comunidad de analistas y programadores, llevando su mayor enfoque y uso en el ámbito de la analítica big data, en el que tiene gran aplicabilidad debido a su estructuración y características.

Con esta investigación queremos determinar las ventajas y funcionalidades que puede tener Scala, así mismo comprobar que tan eficiente es y cómo se desenvuelve en el mundo del big data. Para esto usaremos las metodologías del estudio y evaluación de criterios comparativos con el fin de evaluar la efectividad de Scala en el ámbito del big data. Por eso nuestros objetivos se enfocan en hallar la forma adecuada de solucionar y mejorar el análisis en big data, soportado en el lenguaje de programación SCALA, así como también en las diversas herramientas que ofrece este lenguaje para agilizar desarrollos y análisis de datos que en los lenguajes comunes no se podría. De esta forma se puede hacer la comparativa de SCALA con otros lenguajes que también son usados en el ámbito de la analítica big data, como lo es Python.

Palabras clave: *Lenguaje de programación Scala,Analitica Big data, Scala vs Python, prueba piloto scala-python.*

Abstract Scala is one of the languages that has been gaining fame in big data analytics lately. This language has been designed to accommodate future requirements and is therefore called a Scalable Language (Scala). Scala is not only a fully object-oriented language, but also a fully functional language.

Scala is a language that has been over a decade since it first appeared, but it has been overshadowed by great languages such as Python, R, among others. However, in recent years it has begun to become more popular among the community of analysts and programmers, taking its greater focus and use in the field of big data analytics, in which it has great applicability due to its structure and characteristics.

With this research we want to determine the advantages and functionalities that Scala can have, as well as verify how efficient it is and how it operates in the world of big data. For this we will use the methodologies of the study and evaluation of comparative criteria in order to evaluate the effectiveness of Scala in the field of big data. That is why our objectives are focused on finding the appropriate way to solve and improve the analysis in big data, supported in the SCALA programming language, as well as in the various tools that this language offers to speed up development and data analysis that in the common languages could not. In this way, SCALA can be compared with other languages that are also used in the field of big data analytics, such as Python.

Keywords: *Scala programming language, Big data analytics, Scala vs Python, scala-python pilot test.*

I. INTRODUCCIÓN

Scala es un lenguaje de programación funcional y basado en objetos cuyo inicio en el ambiente de desarrolladores se remonta al 2003, creado por Martin Odersky y cuya popularidad se vio opacada por java y su rápido desarrollo. Scala permite orientación a objetos en el sentido que cada valor es un objeto, definiendo los objetos por clase y es funcional en el sentido de que cada función es un valor en el lenguaje; actualmente se consideran las comparativas de Scala con Python en desarrollo Spark, dado que la popularidad de Scala y su demanda (precio pagado al programador) ha aumentado con respecto a la primera década de este lenguaje, en países como China e India, potencias en la industria de la tecnología, se evidencia mediante las siguientes gráficas el aumento en búsquedas en Scala y temas relacionados en hasta un 450% en los últimos 12 meses.

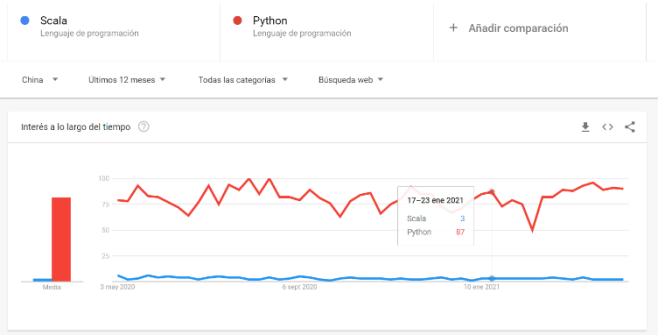

ILUSTRACIÓN 1: SCALA VS PYTHON TENDENCIA DE BÚSQUEDA EN GOOGLE ULTIMO 12 MESES

La grafica anterior evidencia que si comparamos la popularidad de búsquedas de Scala y Python, Python tiene un 71 % más de popularidad que SCALA, también se puede observar que escala a lo largo de los años se han mantenido con búsquedas constantes bajas, en cambio Python se ha ido haciendo más popular con el paso de los años pero su tendencia actual no es tan buena como algunas anteriores.

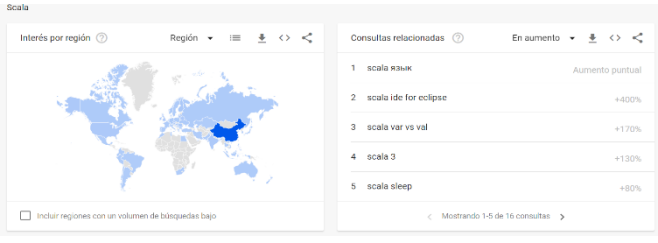

ILUSTRACIÓN 2: INTERÉS POR REGIÓN EN PORCENTAJES SOBRE BÚSQUEDA DE SCALA Y RELACIONADO

ILUSTRACIÓN 3: INTERÉS POR REGIÓN SOBRE BÚSQUEDA DE SCALA

En las gráficas anteriores se muestran el interés de la búsqueda de escala por países/Región, y así mismo el incremento en porcentaje de búsquedas relacionada con el tema Scala.

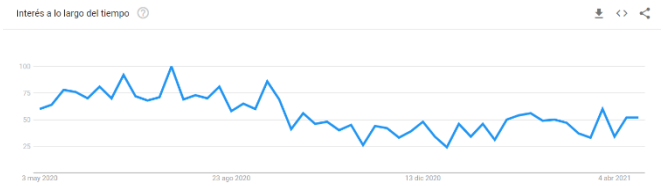

ILUSTRACIÓN 4: INTERÉS A LO LARGO DEL TIEMPO SOBRE SCALA

La grafica anterior representa los niveles de búsqueda de Scala en el buscador de Google a lo largo del tiempo de 12 meses.

Como se observa en las gráficas de Scala en comparación con Python, Python es porcentualmente más popular pero su tendencia gráfica se manifiesta en porcentaje negativo, se entiende que a mayor uso (popularidad) equivale a una mejor aceptación del usuario desarrollador, teniendo en cuenta que Scala y Python comparten características similares en el uso de grandes cantidades de datos, se puede decir que Scala es menos utilizado en la analítica de big data contra Python.

El presente proyecto busca Evidenciar las ventajas y funcionalidades de Scala en el desarrollo de analítica de big data.

II. ESTADOS DEL ARTE

A. Construcción de hardware en un lenguaje integrado Scala

Al incorporar Chisel(El cual es un nuevo lenguaje de construcción de hardware que admite el diseño de hardware avanzado utilizando generadores altamente parametrizados y lenguajes de hardware específicos de dominio en capas.) en el lenguaje de programación Scala, se aumenta el nivel de abstracción del diseño de hardware al proporcionar conceptos que incluyen orientación a objetos,

programación funcional, tipos parametrizados e inferencia de tipos. Chisel puede generar un simulador de software de alta velocidad con precisión de ciclo basado en C ++, o Verilog de bajo nivel diseñado para mapear ya sea en FPGA o en un flujo ASIC estándar para síntesis. Este artículo presenta Chisel, su incrustación en Scala, ejemplos de hardware y resultados para la simulación C ++, la emulación Verilog y la síntesis ASIC. [1]

B. Rendimiento y eficiencia energética de Scala en dispositivos móviles

El objetivo de este estudio fue comparar cómo funcionan los nuevos lenguajes como Scala en entornos móviles en comparación con los lenguajes clásicos. Dado que Scala también se ejecuta en Java Virtual Machine (JVM), es posible ejecutar el código en dispositivos Android y compararlo con Java. [2]

C. Simulación de tráfico urbano a gran escala con Scala y sistema informático de alto rendimiento/Large-scale urban traffic simulation with Scala and high-performance computing system

En este artículo de investigación se centra en los sistemas de información con máximo rendimiento que permiten simulación a gran escala, se expresa que los paradigmas anteriores desarrollaban soluciones eficaces y eficientes basados en plataformas populares de interfaz de paso de mensajes; Se investiga la implementación escalable para un sistema de tráfico asincrónica, usando Scala/AKKA para la programación paralela y distribuida necesaria, además de su funcionalidad optima en sistema de información similares al estudiado. [3]

D. BIG DATA: changes in data management

En este diario podemos apreciar la importancia de la aplicación de big data en el mundo y nos presenta la idea de como podemos aplicar los diferentes conocimientos como lenguajes de programación como SCALA para optimizar la interpretación y el manejo de esta gran cantidad de datos, para que con las conclusiones que nos de el programa, el analista de datos pueda crear un veredicto que afecte de forma positiva el entorno sobre el cual se estudio el dato y así mejorarlo. [4]

E. Técnicas big data: análisis de textos a gran escala para la investigación científica y periodística

En este artículo se nos presenta el análisis (basado en datamining, data science, big data y análisis sentimiento) de grandes cantidades de textos, para poder concluir lo que estos nos dicen y como estos datos pueden ser aplicados en el beneficio de la empresa, en donde existe la posibilidad de aplicar a este entorno el uso del lenguaje de programación SCALA en conjunto de la herramienta JUPYTER para lograr una versatilidad y mayor escalabilidad cuando de resultados exactos se trata. [5]

F. Autómatas de datos en Scala

Este articulo trata de la temática de los autómatas incorporados en el mundo de los datos, así como también hablan del monitoreo y secuencia de eventos y como estos se realizan por medio de una API realizada con el lenguaje de programación SCALA. [6]

G. Un estudio de analítica de Big Data usando Apache Spark con Python y Scala

En este estudio se hace una comparativa de la implementación en Apache Spark de los lenguajes Python y Scala y como estos se comportan ante los altos flujos y variabilidad de datos que se trabajan con Big Data. También se explica brevemente como es la estructura de clústeres en memoria de Apache Spark. [7]

III. MARCO TEORICO

A. Definición de Scala

Scala es relativamente nuevo en la escena de la programación comparado con la mayoría de lenguaje cuyos orígenes rondan los 90, pero de igual manera se ha vuelto popular muy rápidamente. Su nombre es un acrónimo de escalable y lenguaje, especialmente diseñado para crecer con la demanda del usuario.

Scala fue diseñado para ser orientado a objetos y funcional. Es un lenguaje puro orientado a objetos en el sentido de que cada valor es un objeto. Los objetos se definen por clases, que se pueden componer usando la composición mixin. Scala también es un lenguaje funcional en el sentido de que cada función es un valor. Las funciones se pueden anidar y pueden operar en datos utilizando la coincidencia de patrones. [8]

Scala y java al ser parecido su código fuente se compila en código de bytes java, el código ejecutable resultante se ejecuta en una máquina virtual de java (JVM) aunque admite la interoperabilidad del lenguaje con java, por lo que en Scala se puede hacer referencia a bibliotecas escritas en java y viceversa, es decir las bibliotecas se pueden usar en

ambos lenguajes además al ser de naturaleza similar ya que ambos son orientado a objetos y con sintaxis de llaves permite la funcionalidad entre los lenguajes más fácil.

A pesar de toda la semejanza entre Scala y java, Scala a diferencias de java admite características de programación funcional, como inferencia de tipos, evaluación diferida, inmutabilidad, coincidencia de patrones y funciones anidadas. admiten a su vez funciones de orden superior, donde una función puede devolver otra o tomarla como parámetro y cuenta con un sistema de tipos avanzado que admite tipos de datos algebraicos, tipos de orden superior, tipos anónimos y covarianza y contra varianza. Para culminar Scala permite las funcionalidades de sobrecarga del operador, cadenas sin formato, parámetros con nombre y parámetros opcionales, sin embargo, no admite excepciones marcada la cual si admite java.

Como dato extra, el primer lanzamiento público de Scala se produjo en 2004, seguido de la versión 2.0 en marzo de 2006. En 2012 fue galardonado con el ganador del concurso ScriptBowl en la conferencia JavaOne.

B. Historia de Scala

Scala tiene su inicio en el 2001 en la École Polytechnique Fédérale de Lausanne (suiza) por **Martin Odersky**, cocreador del lenguaje de programación **Funnel** genérico de Java, javac y EPFL, donde **Martin odersky** comenzó su investigación y desarrollo de scala como lenguaje de programación funcional y de objeto entre muchos otros objetivos finales del lenguaje.

En palabras de **Martin Odersky** tomada de artima expresa como fue su experiencia a crear el lenguaje de programación Scala [8]:

> En 1995 aprendí sobre Java y me uní a Philip Wadler para escribir un lenguaje funcional que compilara códigos de bytes Java. Este trabajo condujo eventualmente a GJ, el nuevo compilador de Java, y a los genéricos de Java. Fue emocionante trabajar en Java debido a su gran impacto, pero también fue muy difícil porque Java es un lenguaje rico con muchas características que a menudo entran en conflicto con las extensiones de manera imprevista.
>
> En 1999, me uní a EPFL. Pensé que ahora usaría mi libertad académica para hacer diseño de lenguajes a partir de una hoja blanca. Todavía quería combinar programación funcional y orientada a objetos, pero sin las restricciones impuestas por Java. Todo esto me llevó a Funnel, un lenguaje de programación para redes funcionales. Este era un diseño maravillosamente simple, con muy pocas características de lenguaje primitivas. Casi todo, incluidas las clases y la coincidencia de patrones, sería realizado por bibliotecas y codificaciones.
>
> Sin embargo, resultó que el lenguaje no era muy agradable de usar en la práctica. El minimalismo es excelente para los diseñadores de lenguajes, pero no para los usuarios. Los usuarios no expertos no saben cómo hacer las codificaciones necesarias, y los usuarios expertos se aburren de tener que hacerlo una y otra vez.

Funnel condujo a Scala, cuyo diseño comenzó en 2001, y que se lanzó por primera vez en 2003. Scala no es una extensión de Java, pero es interoperable con él. Scala se traduce a códigos de bytes de Java, y la eficiencia de sus programas compilados generalmente es igual a la de Java. [8]

C. Características de Scala en las colas

1. **Open Source:** Scala es un lenguaje de código abierto lo que quiere decir que todos pueden ver, modificar y distribuir este lenguaje, lo que lleva a la ventaja de que no hay restricciones de usuario ni de licencias por ende hay mucha información y contenido de este en la red. Este lenguaje está disponible bajo el BSD-Style Scala License.[9]

2. **Orientado a objetos:** Scala está diseñado como un lenguaje orientado a objetos lo que facilita su entendimiento y magnifica su uso.[11]

3. **Funciones hechas valores:** Todas las funciones en scala son expresiones abstractas, donde dichas funciones son en realidad vistas como valores, estas pueden tener nombres o ser anónimas.[16]

4. **Fácil agrupación:** En este lenguaje todo puede agruparse y anidarse.[16]

5. **Tipado estático:** Este tipado, detecta y evita muchos tipos de errores de aplicación a la hora de compilar.[13]

6. **Deducción de tipo de variable:** A pesar de que escala es de tipado estático, puede deducir de que tipo son las variables que declaremos.[11]

7. **Extensible:** Scala es un lenguaje extensible lo que facilita la implementación y construcción en otros tipos de lenguajes, y de igual forma lo lleva a ser muy manejable para avances futuros.[13]

8. **Interoperabilidad con Java:** Scala es compilado directamente a Java byte code, corriendo en la Java Virtual Machine (JVM), lo que permite invocar métodos de Java, heredar clases de Java, entre otras cosas.[14]

9. **Conciso:** Scala es un lenguaje conciso lo que permite mejor la abstracción del código y haciendo este más entendible.[16]

D. Ventajas y desventajas del lenguaje de programación Scala

1. Ventajas de Scala

Los desarrolladores pueden disfrutar de un mayor rendimiento y una mejor codificación con las complejas funciones de Scala. El pequeño avance que ofrece Scala son las macros, funciones y tuplas. Esos avances son solo alguno

La razón por la que los desarrolladores comparan Scala con Java es que el lenguaje requiere un entorno Java Virtual Machine para funcionar.

Los desarrolladores prefieren Scala debido a su fortaleza como lenguaje y aprecian plenamente la funcionalidad impecable que ofrece. Además de eso, el lenguaje fue desarrollado especialmente para eliminar las restricciones de Java, ofreciendo un lenguaje limpio y bien organizado; Entre los beneficios que ofrece Scala está que la cantidad de código necesario para el programa es menor de lo necesario con el lenguaje Java, y el hecho de que es compatible con la Máquina Virtual Java.

Cada vez son más las empresas que apuestan por Scala, de la que se han destacado algunos ejemplos en Twitter, o la web del diario The Guardian, que también ha realizado cambios. Martin Oddsky explicó al sitio web de Cnet que es la idea que las páginas de gran volumen de datos se pueden cambiar sin problemas de Java a Scala, sin afectar su infraestructura, se pueden beneficiar de su arquitectura JVM.

Ahora bien, Scala ofrece funciones avanzadas y código limpio, junto con programación funcional y orientada a objetos en un paquete de código abierto que aprovecha lo mejor del entorno de Java; Aun así no se puede decir que ningún lenguaje de software sea impecable para cada caso de uso, pero optar por utilizar Scala ofrece varias ventajas. Scala, dado que es escalable, es más fácil de codificar, probar, depurar e implementar. El lenguaje de Scala es versátil y se puede utilizar para programar cualquier cosa, ya sean aplicaciones web, soluciones móviles, juegos, software como servicio y software de escritorio, entre otros.

Es decir:

Es adecuado para el desarrollo web

La ventaja de Scala sobre otros lenguajes en este ámbito es que Scala ofrece notación concisa y proporciona optimización de la complejidad del código. Lo cual lo hace adecuado para tareas de desarrollo web. Con sus características complejas, Scala ofrece una mejor codificación y un aumento del rendimiento. Macros, funciones, tuplas, POO son algunos de los avances que ofrece Scala.

Codificación funcional y orientada a objetos

Con Scala es posible la codificación funcional y orientada a objetos. Los desarrolladores que usan Scala pueden crear sin esfuerzo código que sea funcional y conciso. Las pruebas y el desarrollo también se ampliarán con Scala. De hecho, Scala puede realizar las mismas tareas que Java con solo unas pocas líneas de código. La menor cantidad de líneas de codificación en Scala, junto con el uso conjunto de codificación funcional y orientada a objetos, aseguran un desarrollo, prueba e implementación más rápidos. Las funciones y cierres son parte del lenguaje Scala.

Compatible e interoperable con Java

Aunque Scala es un lenguaje diferente en comparación con Java, los desarrolladores no tienen que preocuparse ya que Scala permite la interoperabilidad y compatibilidad con Java. Los desarrolladores aún pueden aprovechar los beneficios de JVM y también conservar sus

bibliotecas de Java. El uso de objetos singleton en Scala permite una codificación más limpia.

Características avanzadas y funcionales
Scala tiene varias características funcionales como avances en la comparación de cadenas, coincidencia de patrones y combinaciones que incorporan funciones dentro de las definiciones de clases. Scala puede agregar Java para hacer que un programa sea más funcional y dar a los desarrolladores acceso a funciones avanzadas. Otra característica de Scala es su biblioteca API con todas las funciones. Scala no solo está bien documentado, sino que también es accesible.

2. Desventajas de Scala

Hay algunas desventajas en Scala las cuales son las siguientes:

- Puedes pensar en Scala como una espada de doble filo porque con Scala vuelves a cambiar a un paradigma orientado a objetos. Así que no hay obligación de pensar funcionalmente.
- A veces, la información de tipo es un poco más difícil de entender debido a que es un híbrido de orientación funcional y a objetos.
- Se ejecuta en JVM, por lo que no tiene optimización de cola recursiva. Puede usar la anotación @tailrec para obtener beneficios parciales como solución alternativa.
- Scala tiene un grupo de desarrolladores limitado. Es más fácil encontrar desarrolladores de Java en número, no todos los desarrolladores de Java tienen lo que se necesita para codificar en Scala.

Por lo que se atribuye las siguientes desventajas a los siguientes factores:

- Presencia comunitaria limitada. Recursos para ayudarlo a solucionar sus problemas, hay comunidades en línea limitadas disponibles.
- Difícil de aprender. Sintácticamente, Scala es significativamente diferente del java tradicional. Presenta un paradigma completamente diferente.
- Falta de facilidad de adopción. Necesita un equipo que esté ansioso por adoptar Scala para que sea factible; de lo contrario, codifique en un equipo que rara vez encontrará a alguien que escriba únicamente en Scala.
- Compatibilidad con versiones anteriores limitada. Cada vez que se lanza una nueva versión de Scala, es incompatible con la versión anterior y, por lo tanto, genera problemas que pueden retrasar un producto.

E. ¿En qué se puede aplicar Scala y cuáles son sus usos?

Scala es un lenguaje que permite funciones de orden superior (funciones que aceptan funciones como entradas o da funciones como salidas), es open source y básicamente te permite poder crear tus propios frameworks además de que es compatible con java y se ejecuta en JVM. [12]

Teniendo en cuenta lo anterior SCALA se aplica a: [10]

- Industria de los video juegos
- Desarrollo web
- Desarrollo de apps
- Investigaciones académicas
- Big data
- Ciencias de la salud
- Análisis de datos
- Mercados bursátilesFinanzas
- Administración
- Ocio

ILUSTRACIÓN 5: COSTO/PAGO DE UN DESARROLLADO EN UN LENGUAJE X

Escala es un lenguaje de programación muy versátil, lo anteriormente mencionado son las aplicaciones más notables en la que SCALA pude ser aplicado, pero en realidad tiene demasiadas formas de ser usado ya que como su nombre lo dice "es escalable" (la escalabilidad es la capacidad que posee un sistema de adaptarse a un problema) lo que nos quiere decir que puede ser aplicado a cualquier entorno. [11]

F. Entornos de desarrollos y editores de textos aplicables a Scala

Para trabajar con Scala, se puede utilizar cualquier editor de texto y un terminal, junto con un compilador para este. Aunque también se puede trabajar Scala por medio de un Entorno de desarrollo integrado (IDE) utilizando plugins y bibliotecas.
Entre los editores de texto, IDE y herramientas de compilación más utilizados para trabajar escala están:

- Eclipse
- SBT
- NetBeans
- IntelliJ IDEA
- Maven
- Atom
- Anaconda

[15]

IV. METODOLOGÍA DE LA INVESTIGACIÓN

Para el desarrollo de esta investigación se identificaron las funcionalidades del lenguaje en estudio (Scala) en el desarrollo de la analítica de big data, para esta identificación se analizaron y tomaron de base estudios científicos de investigación y/o desarrollo de programas académicos de expertos de big data en Scala, profundizando en el uso y aplicabilidad que posee Scala permitiendo así establecer comparativa de funcionalidades entre Scala y Python que derivaron en las siguientes métricas.

Tabla 1. Metricas evaluativas

METRICAS EVALUATIVAS		
Marcadores: Muy Negativa (1) - Muy positiva (5)		
Lenguaje	Python	Scala
Complejidad de comandos		
Tiempo de ejecucion		
Funcionalidad resultante		

Estas métricas representan las 3 concepciones que se determinaron más importantes, el cómo, cuanto, y que resultado emerge luego de todo el proceso, siendo así que la **Complejidad de comandos** se concibe en el cómo hago para que la maquina haga lo que yo quiera, y que tan funcional y complejo es dicho comando, es decir, si para mostrar por pantalla solo debo usar el comando println() que tanto parámetros le puedo colocar en el argumento, la métrica de **Tiempo de ejecución** es como su nombre lo indica, un medidor del tiempo de ejecución por cada función, para poder monitorear un tiempo promedio, las pruebas se ejecutaron múltiples veces, por último la métrica de **Funcionalidad resultante** es la encargada de medir que tan complejo pero al tiempo fácil de entender (y/o modificar) es el resultado de una línea de comando; Claramente también se tuvo en cuenta otras métricas evaluativas que afectan implícitamente al desarrollo de programas/software dentro de estos lenguajes, métricas tales como: 1) La curva de aprendizaje, 2) Documentación accesible para aprender, y 3) Comunidad colaboradora de errores.

una vez terminada la construcción de las métricas se procedió en la creación de los requerimientos para la prueba piloto la cual en indicadores consta de los siguientes pasos:

- Importar librerías
- Leer datos
- Seleccionar y renombrar columnas
- Explorar variables numéricas con reporte básico y completo
- Análisis Descriptivo
- Gráficos de muestra de Dataframe (filtrado)
- Limpieza de los datos
- Matriz correlación
- Partición de los datos
- Ajuste del modelo y predicción
- Cálculo de parámetros y precisión

Culminado las fases anteriores se procedió a adquirir el conjunto de datos a utilizar en la prueba de python, el cual fue tomado de los datos abiertos del país del siguiente link: https://www.datos.gov.co/api/views/3san-pce5/rows.csv?accessType=DOWNLOAD, cuya base de datos original consta de 54 columnas/variables y un total de registro de 410.809, de todas esas columnas trabajamos con las siguientes:

BD = Base de datos

DF = Dataframe

Tabla 2. Columnas escogidas Python

Nombre BD	Nombre DF	Tipo de datos BD	Tipo de datos DF
estu_edad	edad	int	int
estu_genero	genero	String	int
estu_nacimiento_anno	año	float	float
estu_reside_mncipio	mcpio	String	int
estu_reside_depto	dpto	String	int
cole_calendario	col_cal	String	int
cole_genero	col_gen	String	int
cole_jornada	col_jor	String	int
punt_lenguaje	lenguaje	float	float
punt_matematicas	matematicas	float	float
punt_filosofia	filosofia	float	float
estu_puesto	puesto	float	float

Y cuya descripción de que representa los datos está directamente asociada a su nombre, en la Tabla 2 también se evidencia el preprocesamiento descriptivo a cada variable para la posterior implementación de las pruebas, este preprocesamiento se evidencia como Limpieza de datos en los indicadores de los requerimientos de la prueba, donde se cambiaron el nombre y tipo de las variables para el uso de los modelos de Machine Learning y posterior evaluación, los modelos de Machine Learning usados son el de Regresión Lineal Múltiple (RLM) y la Red Neuronal MLP bajo la prueba piloto en Python.

Para el desarrollo de la prueba piloto en Python se usó una distribución libre y abierta de los lenguajes Python y R, el cual es Anaconda, siendo usado anaconda en ciencia de datos, y aprendizaje automático (machine learning) esta selección es derivada de la experiencia de los investigadores con respecto a la herramienta, por el lado del desarrollo de la prueba piloto en Scala se optó por Databricks de Azure Microsoft, Databricks es una plataforma brindada como servicio Cloud en Microsoft Azure, es una plataforma analítica de datos basada en Apache Spark y cuyo cluster incluye Scala por defecto entre otro lenguajes, se eligió Databricks porque otros IDEs para ejecutar líneas de código bajo el lenguaje Scala, se tenía que realizar una serie de pasos de configuración bastante complicados y muy pocos documentados, lo cual aumenta el nivel de dificultad de programar bajo Scala a nivel local de IDE.

Luego de la definición de plataformas para desarrollar las pruebas piloto se tomaron los datos para la prueba piloto de Scala del siguiente link: (https://www.datos.gov.co/Salud-y-Protecci-n-Social/SIVICAP-Irca-Anual-por-Municipio-2018/nde2-wquc), cuya base de datos original consta de 8 columnas/variables y un total de registro de 1055, trabajando de todas las columnas solo con las siguientes:

Tabla 3. Columnas escogidas Scala

Nombre	Tipo de datos BD	Tipo de datos DF
Muestras	float	float
IRCA	float	float
Nivel_riesgo	String	int

La descripción de que representan los datos está directamente asociada a su nombre, para muestra de ejemplo, la columna Muestras, indica la cantidad de yacimientos de agua que se evaluaron, y el IRCA su índice IRCA (Índice de calidad de agua), en Scala se optó por otra base de datos de menor tamaño en términos de registro y peso (mb), ya que la plataforma Databricks presento problemas para subir la base de datos tratada en Python, por lo que migrar a otra base de datos fue la opción resultante.

Una vez hecha la Limpieza de datos en la prueba bajo Scala, se convierte el tipo de datos original (tipo de datos BD) a el tipo de datos necesario para los modelos de Machine Leraning (Tipo de datos DF), posteriormente el tratamiento de datos necesario como la partición del conjunto para implementar los modelos de Regresión Lineal Múltiple (RLM) y la Red Neuronal MLP bajo Scala - Databricks.

V. RESULTADOS

En Python para la evaluación de métricas se optó por las líneas de comandos importantes en el proceso del desarrollo de la prueba, las cuales se muestran a continuación

Tabla 4. Comandos escogidos en python con descripcion

Linea de comando	Descripcion
df = pd.read_csv('Saber_11__2006-2.csv')	Leer Datos
df.describe()	Reporte de variables numericas

plt.figure(figsize=(8,6));
plt.hist(df['filosofia']);
plt.xlabel('Puntaje Filosofia',fontdict={'size':16,'color':'orange','family':'Arial'});
	Realizar histograma bajo una variable

df_baq = df[df['mcpio']=='BARRANQUILLA']	Filtrado de datos en DataFrame
encoder = preprocessing.LabelEncoder() encoder.fit(df['genero']) df['genero'] =encoder.transform(df['genero']	Transformacion de variables categoricas a numericas
lr = linear_model.LinearRegression() lr.fit(X_train, y_train) y_pred = lr.predict(X_test)	Modelo de machine learning, regresion lineal multiple

A estas líneas se le aplico las métricas evaluativas de tal manera que para la **complejidad de comandos** se obtuvo la siguiente tabla

Tabla 5. Comandos Python evaluados en complejidad de comandos

Linea de comando	Puntaje
df = pd.read_csv('Saber_11__2006-2.csv')	5
df.describe()	5

plt.figure(figsize=(8,6));
plt.hist(df['filosofia']);
plt.xlabel('Puntaje Filosofía',fontdict={'size':16,'color':'orange','family':'Arial'});
	4

df_baq = df[df['mcpio']=='BARRANQUILLA']	5
encoder = preprocessing.LabelEncoder() encoder.fit(df['genero']) df['genero'] =encoder.transform(df['genero']	4
lr = linear_model.LinearRegression() lr.fit(X_train, y_train) y_pred = lr.predict(X_test)	5

Esta tabla representa lo que, según la experiencia de los investigadores, que tan comprensible es la línea de comando para su ejecución, y que otros argumentos acepta el comando. Para esto se consideró lo que representa cada línea de comando, sus argumentos secundarios, cantidad de argumentos y que tan lógicos y fáciles de usar son, este análisis será comparado con la **complejidad de comandos** en Scala más adelante y se expondrá las conclusiones.

Para la métrica de tiempo de ejecución, en Python se decidió usar la función .time(), que permitió comenzar a cronometrar el tiempo de ejecución de una secuencia de código hasta que se le indique un stop-alto, gracias a .time() se obtuvieron los siguiente tiempo de ejecución

Tabla 6. Comandos Python evaluados en tiempo de ejecucion

Linea de comando	Tiempo de ejecucion (seg)
df = pd.read_csv('Saber_11__2006-2.csv')	7,02
df.describe()	0.10

plt.figure(figsize=(8,6));
plt.hist(df['filosofia']);
plt.xlabel('Puntaje Filosofía',fontdict={'size':16,'color':'orange','family':'Arial'});
	0.08

df_baq = df[df['mcpio']=='BARRANQUILLA']	0,03
encoder = preprocessing.LabelEncoder() encoder.fit(df['genero']) df['genero'] =encoder.transform(df['genero']	0.08
lr = linear_model.LinearRegression() lr.fit(X_train, y_train) y_pred = lr.predict(X_test)	0.09

En lo que para los investigadores respecta, basado en la experiencia, son tiempos bastante aceptables para los procesos que realiza cada código o línea de comando.

Por ultima métrica evaluativa en Python, tenemos la **funcionalidad resultante** que para facilidad de comprensión y desarrollo será evaluada también bajo las mismas líneas de comando anteriores, la cual esta representada en la siguiente tabla

Tabla 7. Comandos Python evaluados en funcionalidad resultante

Linea de comando	resultado
df = pd.read_csv('Saber_11__2006-2.csv')	5
df.describe()	5

plt.figure(figsize=(8,6));
plt.hist(df['filosofia']);
plt.xlabel('Puntaje Filosofía',fontdict={'size':16,'color':'orange','family':'Arial'});
	5

df_baq = df[df['mcpio']=='BARRANQUILLA']	5
encoder = preprocessing.LabelEncoder() encoder.fit(df['genero']) df['genero'] =encoder.transform(df['genero']	4
lr = linear_model.LinearRegression() lr.fit(X_train, y_train) y_pred = lr.predict(X_test)	5

Estos resultados representan la función que cumple cada línea de comando y el como lo expresa de manera grafica o textual, estos resultados están sujetos bajo la experiencia con inteligencias artificial y modelos de Machine Learning de los investigadores, pero claramente evaluados bajo lógica, por ejemplo, el poder controlar en

$$plt.figure(figsize = (8,6));$$

$$plt.hist(df['filosofia']);$$

$$plt.xlabel('Puntaje\ Filosofía', fontdict = \{'size': 16, 'color': 'orange', 'family': 'Arial'\});$$

gran medida el resultado de un histograma con el código:

ILUSTRACIÓN 6: CODIGO PARA GENERAR HISTOGRAMA

Siendo capaces de cambiar no solo el tamaño, sino el color representativo de la variable en el histograma, los labels y el Font-family son argumentos de funcionalidades bastante apreciados.

Por todo lo anterior en términos de la métrica evaluativa original, la evaluación de Python se obtuvo de la siguiente manera

Tabla 8. Metricas evaluativas originales / Python

METRICAS EVALUATIVAS		
Marcadores: Muy Negativa (1) - Muy positiva (5)		
Lenguaje	Python	Scala
Complejidad de comandos	5	
Tiempo de ejecucion	4	
Funcionalidad resultante	5	

Para la realización y evaluación de las métricas bajo Scala en Databricks los investigadores optaron por evaluar las líneas de comandos equivalentes a las evaluadas en Python generando la siguiente tabla

Tabla 9. Comandos escogidos en Scala con descripcion

Linea de comando	Descripcion
val testcsvgit ="/FileStore/tables/IRCA-1.csv" val df = spark .read.option("header", true) .option("inferSchema", true) .csv(testcsvgit)	Leer Datos
df.describe()	Reporte de variables numericas
display(df)	Realizar histograma bajo una variable
val filtrar = DF.filter("Muestras > 152 AND IRCA > 0")	Filtrado de datos en DataFrame
val nivel_codes = DF.groupBy("Nivel_riesgo").count().orderBy("count").withColumnRenamed("Count","Nivel_count") val df2 =	Transformacion de variables categoricas a numericas
DF.join(nivel_codes,Seq("nivel_riesgo"))df2.createOrReplaceTempView("df2")	
val features = new VectorAssembler() .setInputCols(Array("IRCA")) .setOutputCol("features") val lr = new LinearRegression().setLabelCol("Nivel_count") val pipeline = new Pipeline().setStages(Array(features, lr)) val model = pipeline.fit(df2) val linRegModel = model.stages(1).asInstanceOf[LinearRegressionModel] println(s"RMSE: ${linRegModel.summary.rootMeanSquaredError}") println(s"r2: ${linRegModel.summary.r2}") println(s"Model: Y = ${linRegModel.coefficients(0)} * X + ${linRegModel.intercept}") linRegModel.summary.residuals.show()	Modelo de machine learning, regresion lineal multiple

De tal manera que para las líneas de comando en Scala queda la **Complejidad de comandos** de la siguiente manera

Tabla 10. Comandos Scala evaluados en complejidad de comandos

Linea de comando	Puntaje
val testcsvgit ="/FileStore/tables/IRCA-1.csv" val df = spark .read.option("header", true) .option("inferSchema", true) .csv(testcsvgit)	5
df.describe()	4
display(df)	5
val filtrar = DF.filter("Muestras > 152 AND IRCA > 0")	4
val nivel_codes = DF.groupBy("Nivel_riesgo").count().orderBy("count").withColumnRenamed("Count","Nivel_count") val df2 =	2
DF.join(nivel_codes,Seq("nivel_riesgo"))df2.createOrReplaceTempView("df2")	
val features = new VectorAssembler() .setInputCols(Array("IRCA")) .setOutputCol("features") val lr = new LinearRegression().setLabelCol("Nivel_count") val pipeline = new Pipeline().setStages(Array(features, lr)) val model = pipeline.fit(df2) val linRegModel = model.stages(1).asInstanceOf[LinearRegressionModel] println(s"RMSE: ${linRegModel.summary.rootMeanSquaredError}") println(s"r2: ${linRegModel.summary.r2}") println(s"Model: Y = ${linRegModel.coefficients(0)} * X + ${linRegModel.intercept}") linRegModel.summary.residuals.show()	5

Como se puede observar, la línea de comando que representa el leer Datos es mucho mas compuesta que la de Python, aun así, tiene un puntaje perfecto de 5 y este resultado es porque justamente su complejidad de comando y argumentos que puede usar son superior a Python pero al ser superior es de mayor dificultad su comprensión (a pesar de ser lógica) y construcción, como tal esta línea de comando que representa el leer datos, faculta modificar el diseño del DataFrame y otras opciones que permite tener una construcción mas especifica de los datos relevantes que se desea tener en el DataFrame, y todo desde el la misma línea de comando, claramente como es observable en la tabla 10, se agrega los argumentos como funciones, por ejemplo el .read.option("header",true)->su construcción y el llamado es de una función.

Para la evaluación de las líneas de comando en Scala bajo la métrica de **tiempo de ejecución** se obtuvieron los siguientes datos

Tabla 11. Comandos Scala evaluados en tiempo de ejecucion

Linea de comando	Tiempo de ejecución (s)
val testcsvgit ="/FileStore/tables/IRCA-1.csv" val df = spark .read.option("header", true) .option("inferSchema", true) .csv(testcsvgit)	1,88
describe()	0,57
display(df)	2
val filtrar = DF.filter("Muestras > 152 AND IRCA > 0")	1
val nivel_codes = DF.groupBy("Nivel_riesgo").count().orderBy("count").withColumnRenamed("Count","Nivel_count") val df2 = DF.join(nivel_codes,Seq("nivel_riesgo"))df2.createOrReplaceTempView("df2")	2,3
val features = new VectorAssembler() .setInputCols(Array("IRCA")) .setOutputCol("features") val lr = new LinearRegression().setLabelCol("Nivel_count") val pipeline = new Pipeline().setStages(Array(features, lr)) val model = pipeline.fit(df2) val linRegModel = model.stages(1).asInstanceOf[LinearRegressionModel] println(s"RMSE: ${linRegModel.summary.rootMeanSquaredError}") println(s"r2: ${linRegModel.summary.r2}") println(s"Model: Y = ${linRegModel.coefficients(0)} * X + ${linRegModel.intercept}") linRegModel.summary.residuals.show()	3,3

Para el desarrollo evaluativo de esta métrica, los investigadores al usar Databricks, no tuvieron la necesidad de usar ninguna función tipo. time() o similar, ya que la plataforma Databricks permite visualizar el tiempo de ejecución, una vez ejecutada el segmento de código; Como es posible ver, lo tiempo de ejecución son variados en comparación con el de Python, el leer los datos es mas rápido en Scala, posiblemente por el tamaño de la BD, otros valores son de mayor tiempo en comparación con Python y posiblemente se deba a que usa una plataforma Cloud y la frecuencia del internet es variada.

Para la evaluación de las líneas de comando en Scala bajo la métrica de **funcionalidad resultante** se obtuvieron los siguientes datos

Tabla 12. Comandos Scala evaluados en funcionalidad resultante

Linea de comando	Resultado
val testcsvgit ="/FileStore/tables/IRCA-1.csv" val df = spark .read.option("header", true) .option("inferSchema", true) .csv(testcsvgit)	5
describe()	4
display(df)	5
val filtrar = DF.filter("Muestras > 152 AND IRCA > 0")	3
val nivel_codes = DF.groupBy("Nivel_riesgo").count().orderBy("count").withColumnRenamed("Count","Nivel_count") val df2 = DF.join(nivel_codes,Seq("nivel_riesgo"))df2.createOrReplaceTempView("df2")	4
val features = new VectorAssembler() .setInputCols(Array("IRCA")) .setOutputCol("features") val lr = new LinearRegression().setLabelCol("Nivel_count") val pipeline = new Pipeline().setStages(Array(features, lr)) val model = pipeline.fit(df2) val linRegModel = model.stages(1).asInstanceOf[LinearRegressionModel] println(s"RMSE: ${linRegModel.summary.rootMeanSquaredError}") println(s"r2: ${linRegModel.summary.r2}") println(s"Model: Y = ${linRegModel.coefficients(0)} * X + ${linRegModel.intercept}") linRegModel.summary.residuals.show()	5

La evaluación que representa la tabla 12 está basada en los hechos que bajo la experiencia de los investigadores son precisos al momento de evaluar la funcionalidad resúltate de una línea de código o comando, en contexto, el comando describe() trae un gran número de datos valiosos para cada uno de las variables numéricas del DateFrame, aunque para los dos lenguajes es el mismo hay que conocer que en Python es mucho mejor por el hecho que da más información y datos estadísticos de las variables

	edad	año	lenguaje	matematicas	filosofia	puesto
count	410808.000000	409336.000000	410808.000000	410808.000000	410808.000000	410546.000000
mean	17.392697	1988.211323	48.450053	45.002582	47.311928	489.068087
std	3.639453	3.969372	6.835133	8.369117	7.716471	275.857891
min	11.000000	982.000000	-1.000000	-1.000000	-1.000000	1.000000
25%	16.000000	1988.000000	43.060000	39.380000	42.380000	253.000000
50%	17.000000	1989.000000	47.410000	45.660000	47.300000	492.000000
75%	18.000000	1990.000000	53.300000	50.970000	51.780000	727.000000
max	76.000000	2091.000000	100.280000	115.670000	92.810000	966.000000

ILUSTRACIÓN 7: DESCRIBE() EN PYTHON

En cambio, para Scala se limita a simple cinco estadísticos y por ende un reporte más básico, pero de igual manera funcional.

```
+-------+------------------+-----------------+
|summary|          Muestras|             IRCA|
+-------+------------------+-----------------+
|  count|              1054|             1054|
|   mean| 45.32732447817837| 22.9810626185958|
| stddev|112.70566964715033|21.89465438429346|
|    min|                 1|              0.0|
|    max|              2774|            96.25|
+-------+------------------+-----------------+
```

ILUSTRACIÓN 8: DESCRIBE() EN SCALA

Otros comandos evaluado por los investigadores y cuyo resultado es menester saber la explicación, son los usados para construir gráficos, específicamente el histograma; En Scala gracias a Apache Spark se permitió usar la función displays(DF) que es mucho más complejo a nivel grafico que su contraparte en Python, con una sola línea de código permite generar nueve graficas diferentes, también posee una interfaz gráfica que posibilita la escogencia de variables a usar y otras opciones más.

Colisionando toda la información expresada de los resultados de las pruebas basadas en las métricas de la tabla 1, que son las métricas evaluativas, se obtiene que

Tabla 13. Metricas evaluativas originales / Python vs Scala

METRICAS EVALUATIVAS		
Marcadores: Muy Negativa (1) - Muy positiva (5)		
Lenguaje	Python	Scala
Complejidad de comandos	5	4
Tiempo de ejecucion	4	4
Funcionalidad resultante	5	4

Cuyo entendimiento completo de los datos que representa cada métrica se sustenta de toda la información dada en este segmento de resultado, y al ser evaluada la equivalencia de los comandos hechos en Python y replicados en Scala, se encontró, que algunos comandos son más fáciles e intuitivos en un lenguaje que en el otro.

VI. CONCLUSIONES

El odjetivo de este articulo esta en evidenciar las ventajas y funcionalidades de Scala en el desarrollo de analítica de Big Data, el motivo de este análisis surge de la necesidad de la analítica de Big Data en la actualidad y con esta necesidad un gran campo de herramientas para trabajar bajo Big Data, teniendo esto en cuenta se opto por desarrollar pruebas pilotos basada en Big Data, que compararan 2 lenguajes con herramientas en las cuales se puede trabajar bajo Big Data, cuales lenguajes se tomaron con el criterio que un lenguaje sea conocido por las comunidades de desarrolladores y académicas y el segundo, que sería el eje central de la investigación, no tan conocido.

Gracias al desarrollo de las pruebas, en esta investigación podemos concluir que después de desarrollar las pruebas pilotos bajo herramientas y lenguajes diferentes, se puede apreciar que Scala tiene una funcionalidad de calidad tanto como Python, mas sin embargo este gran potencial que tiene Scala se ve obstruido por la poca documentación y tutoriales de enseñanza que se tienen con este lenguajes enfocado en la Big Data, en parte es porque Scala es de gran espectro y los tutoriales están enfocados a programación bajo

desktop, aun así como se puede evidenciar en la tabla 13, Scala y su potencial se puede equiparar a Python, lenguaje que en la actualidad domina el mundo de la analítica de Big Data.

Queda como tarea del proyecto y de los investigadores, un proyecto para sacar datos de streaming con Databricks bajo Scala, esto para dar cierre a los problemas presentados por la plataforma Databricks para subir tables o base de datos más robusta, y poder sacar mayor rendimiento a Scala.

Referencias

[1] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek y K. Asanović, «construcción de hardware en un lenguaje integrado Scala,» 3-7 5 2012. [En línea]. Available: https://ezproxy.unisimon.edu.co:2131/document/6241660.

[2] M. Denti y J. Nurminen, «Rendimiento y eficiencia energética de Scala en dispositivos móviles,» *ScienceDirect*, Vols. %1 de %225-27, nº (25-27 de septiembre de 2013). Prhttps://ezproxy.unisimon.edu.co:2131/document/6658099/authors#authors., p. 9, 2013.

[3] M. Janczykowski, W. Turek, M. Malawski y A. Byrski, «Large-scale urban traffic simulation with Scala and high-performance computing system,» de *Journal of Computational Science*, 2019, pp. Pages 91-101.

[4] D. Sebalj, A. Živković y K. Hodak, «Big Data: Changes in Data Management,» *Ekomomski Vjesnik*, vol. 29, pp. 487-499, 2016.

[5] C. Arcila, E. Barbosa y F. Cabezuelo, «Técnicas big data: análisis de textos a gran escala para la investigación científica y periodística,» *El profesional de la información*, vol. 25, nº 4, pp. 626-631, 2016.

[6] K. Havelund, «Data Automata in Scala,» de *Conferencia sobre aspectos teóricos de la ingeniería de software*, 2014.

[7] Y. Gupta y S. Kumari, «A Study of Big Data Analytics using Apache Spark with Python and Scala,» *3rd International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 471-478, 3-5 Diciembre 2020.

[8] M. Odersky, «Artima,» 9 6 2006. [En línea]. Available: https://www.artima.com/weblogs/viewpost.jsp?thread=163733. [Último acceso: 30 3 2021].

[9] K. CODING, «Motivos para aprender scala,» 2018.

[10] K. CODING, «KEEP CODING,» 24 MAYO 2018. [En línea]. Available: https://keepcoding.io/blog/10-motivos-por-los-que-debes-aprender-scala/.

[11] M. Odersky, Programing in scala, mountain view california: PrePrintTM, 2007.

[12] M. Odersky, The programming scala specification, suiza: EPFL, 2011.

[13] «DOCS SCALA,» [En línea]. Available: https://docs.scala-lang.org/es/tour/tour-of-scala.html#:~:text=Scala%20es%20un%20lenguaje%20de,orientados%20a%20objetos%20y%20funcionales..

[14] «DCCIA,» [En línea]. Available: http://www.dccia.ua.es/dccia/inf/asignaturas/LPP/seminarios/seminario2-scala/seminario2-scala.html.

[15] «SCALA LANG,» [En línea]. Available: https://www.scala-lang.org/download/.

[16] Gary Briceño, «CLUB DE TECNOLOGIA,» 23 Abril 2015. [En línea]. Available: https://www.clubdetecnologia.net/blog/2015/scala-caracteristicas-del-codigo/.