UNIVERSIDAD DE Belgrano

BUENOS AIRES - ARGENTINA

Análisis comparativo entre PySpark y Pandas para el procesamiento de datos masivos de covid19

ALUMNO: ELISEO ROBERTO MORENO VILLA

CARRERA: INGENIERÍA EN INFORMÁTICA

MATRICULA: 502-11608

TUTOR: MG. ING. PABLO PANDOLFO

DIR. DE CARRERA: MG. PAULA ANGELERI

CICLO: 2021

A mi familia y compañera de vida.

AGRADECIMIENTOS

Este trabajo final no hubiese sido posible sin la colaboración de mi tutor Pablo Pandolfo. Desde el comienzo Pablo me acompaño y ofreció toda la experiencia que tiene en este rubro y como investigador para delinear, desarrollar y consolidar este trabajo final de carrera.

A mis padres, hermanos y mi pareja que me han acompañado y sostenido antes y durante todo el periodo de mi formación de forma incondicional, gracias a todos.

A la Universidad de Belgrano le debo las gracias porque gracias a la institución y su nivel conocí compañeros y profesores excepcionales ya son parte de mi vida. Además, agradezco infinitamente por el acompañamiento durante la carrera y especialmente durante el desarrollo de este trabajo final de carrera ocurrido en pandemia.

RESUMEN

Históricamente, las computadoras se volvieron más rápidas cada año a través del aumento de la velocidad del procesador, los nuevos procesadores cada año podían ejecutar más instrucciones por segundo que el año anterior. Por consiguiente, las aplicaciones se volvieron más rápidas cada año sin que se necesitaran cambios en su código. Esta tendencia condujo a un ecosistema de aplicaciones que se acumuló con el tiempo, la mayoría de las cuales fueron diseñadas para ejecutarse en un solo procesador. Estas aplicaciones siguieron aprovechando los avances de velocidades de procesador para escalar a cálculos y volúmenes de datos mayores. [ISAACSON, 2014]

Desafortunadamente, estos avances encontraron su límite alrededor de 2005 debido a los límites en la disipación de calor, los desarrolladores de hardware dejaron de hacer procesadores individuales más rápidos y cambiaron a agregar más núcleos de CPU paralelos, todos funcionando a la misma velocidad. Este cambio significó que, de repente, las aplicaciones debían modificarse para agregar paralelismo con el fin de funcionar más rápido, lo que sentó las bases para nuevos modelos de programación como Apache Spark. [ISAACSON, 2014]

Sumado a lo anterior, las tecnologías de almacenamiento de datos no se ralentizaron. El costo de almacenar 1 terabytes de datos continúa disminuyendo aproximadamente dos veces cada 14 meses, lo que significa que es muy económico para organizaciones de todos los tamaños almacenar grandes cantidades de datos. [ISAACSON, 2014]

Los datos generados por cada individuo cada vez crecen más, pero de manera más significativa, los datos generados por los sistemas de computación son aún mayor que los generadas por las personas. Logs de sistemas, lectores RFID, sensores, rastreos

GPS de vehículos, transacciones minoristas: todo esto contribuye a la actual montaña de datos.

Por esto, a la actualidad se la conoce como la Era de Los Datos. No es fácil medir el volumen total de datos digitales almacenados, pero una estimación de IDC (International Data Corporation) sitúa al año 2020 con 59 zettabytes de datos creados, capturados, copiados y consumidos en el mundo. Un zettabyte son 10^21 bytes, o lo que equivale a mil exabytes, un millón de petabytes o mil millones de terabytes. [IDC, 2020]

El resultado final de lo expuesto, es un mundo en el que la recopilación de datos es extremadamente económica ya que muchas organizaciones en la actualidad incluso consideran negligente no registrar datos de posible relevancia, pero procesarlos requiere grandes cálculos paralelos. Además, en este nuevo mundo, el software desarrollado en los últimos 50 años no puede escalar automáticamente, ni tampoco los modelos de programación tradicionales para aplicaciones de procesamiento de datos, lo que crea la necesidad de nuevos modelos de programación.

De esta nueva Era surgen las herramientas de Big Data que ayudan a recolectar, almacenar, transformar, y lo más importante, aprovechar esta creciente montaña de datos. Este gran salto en la cantidad de datos permite responder preguntas que antes eran imposibles de responder. Gracias a esto existe la oportunidad de encontrar nuevos caminos para solucionar problemas que afectan a la sociedad hace años como por ejemplo los planteados en las Naciones Unidas. [ONU, 2015]

Por estos motivos, los organismos de la actualidad tienen que analizar si es necesario aplicar herramientas de analytics de Big Data y, en el caso de que sea necesario, cuál sería la más conveniente.

En este trabajo final se desarrolló un producto de software que contiene los scripts necesarios para determinar entre PySpark y Pandas cual presenta mejor rendimiento para el procesamiento de datos haciendo uso de un único nodo. Los scripts están desarrollados con Python, haciendo uso del entorno de trabajo Jupyter Notebook y apoyados en el Framework para la evaluación de software MyFEPS para determinar las métricas a evaluar y comparar.

ABSTRACT

Historically, computers became faster each year through increased processor speed, new processors each year able to execute more instructions per second than the previous year. As a result, applications became faster every year without requiring any changes to their code. This trend led to an ecosystem of applications that accumulated over time, most of which were designed to run on a single processor. These applications continued to take advantage of advances in processor speeds to scale to larger volumes of data and calculations. [ISAACSON, 2014]

Unfortunately, these advancements found their limit around 2005 due to limits on heat dissipation, hardware developers stopped making faster individual processors and switched to adding more parallel CPU cores, all running at the same speed. This change meant that applications suddenly had to be modified to add parallelism in order to run faster, laying the foundation for new programming models like Apache Spark. [ISAACSON, 2014]

In addition to the above, data storage technologies did not slow down. The cost of storing 1 terabytes of data continues to decline approximately twice every 14 months, which means that it is very economical for organizations of all sizes to store large amounts of data. [ISAACSON, 2014]

The data generated by each individual is growing more and more, but more significantly, the data generated by computer systems is even greater than that generated by people. System logs, RFID readers, sensors, GPS vehicle tracks, retail transactions - all of these contribute to today's mountain of data.

For this reason, today it is known as the Age of Data. It is not easy to measure the total volume of digital data stored, but an estimate by IDC (International Data Corporation) places the year 2020 with 59 zettabytes of data created, captured, copied and consumed in the world. One zettabyte is 10 ^ 21 bytes, or what equates to one thousand exabytes, one million petabytes, or one billion terabytes. [IDC, 2020]

The end result of the above is a world in which data collection is extremely economical since many organizations today even consider it negligent not to record data of possible relevance, but processing them requires large parallel calculations. Furthermore, in this new world, software developed in the last 50 years cannot scale automatically, nor can traditional programming models for data processing applications, creating the need for new programming models.

From this new Era come the Big Data tools that help collect, store, transform, and most importantly, take advantage of this growing mountain of data. This huge jump in the amount of data allows answering questions that were previously impossible to answer. Thanks to this, there is the opportunity to find new ways to solve problems that have affected society for years, such as those raised in the United Nations. [UN, 2015]

For these reasons, today's organizations have to analyze whether it is necessary to apply Big Data analytics tools and, if necessary, which would be the most convenient.

In this final work, a software product was developed that contains the necessary scripts to determine between PySpark and Pandas which has the best performance for data processing using a single node. The scripts are developed with Python, making use of the Jupyter Notebook work environment and supported by the MyFEPS Software Evaluation Framework to determine the metrics to evaluate and compare.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. ORGANIZACIÓN DEL DOCUMENTO

En la introducción se describen las características de la problemática, los interrogantes que la guían, las limitaciones que presenta y el resultado esperado, así como los pasos que se llevan a cabo para la construcción del prototipo.

El trabajo consta de tres etapas:

Marco conceptual:
En esta sección se profundizan los conceptos de las distintas áreas involucradas en el desarrollo del trabajo. Sobre esta base, se avanza hacia el estudio de la implementación y el potencial de las herramientas de análisis de datos en entornos Big Data.

Para el desarrollo de los distintos conceptos, se presenta material obtenido de distintos autores. Dichos autores presentan un trabajo más amplio/profundo sobre los conceptos, por ese motivo se extrajo del marco teórico los extractos necesarios con el fin de definir los conceptos, características, limitaciones y desafíos a la hora de desarrollar soluciones de datos en entornos Big Data.

Desarrollo del trabajo:
Se sientan las bases de la comparativa, justificando la elección y el uso de las distintas herramientas utilizadas como entorno de trabajo en la comparativa entre Pandas y PySpark.

Se definen los ejes de comparación y se desarrollan los scripts necesarios para llevarla a cabo.

Para cumplir con los scripts, se define una metodología de desarrollo y testing adecuada, y se presenta un entregable que consta de un Jupyter Notebook con los scripts de

procesamiento de datos correspondientes sobre la base de un DataFrame con casos de covid19 en Argentina ofrecido por el Gobierno de la Nación.

Conclusiones, bibliografía y anexos:

Por último, se presentan las conclusiones del trabajo, futuras líneas de investigación, la bibliografía que sirvió de base para este trabajo, un glosario de términos utilizados en el desarrollo del mismo y una guía para la instalación y configuración del entorno de trabajo.

1.1. INTRODUCCIÓN

1.2. Planteamiento y Contexto del Problema

La explotación de datos es uno de los problemas que todos los organismos privados y/o públicos enfrentan diariamente para obtener información importante en busca de una mejora en procesos de una empresa o en la calidad de vida si se piensa en un organismo gubernamental.

El avance de la tecnología y el decremento de los costos del almacenamiento permiten que sea posible (y hoy en día obligatorio) recolectar la mayor cantidad de datos posibles con el fin de explotarlos. Estos datos por si solos no sirven de mucho, pero al analizarlos en conjunto dan información, la cual es una herramienta fundamental para conocer el estado actual y que sería conveniente cambiar para mejorarlo.

Pobreza, enfermedad, hambre, cambio climático y desigualdad son algunos de los problemas que enfrenta el mundo hoy en día. La Agenda 2030 para el Desarrollo Sostenible, adoptada por todos los Estados Miembros de las Naciones Unidas en 2015, proporciona un plan común para la paz y la prosperidad para las personas y el planeta, ahora y en el futuro. En su esencia, se encuentran los 17 Objetivos de Desarrollo Sostenible (ODS), que son un llamado urgente a la acción de todos los países, desarrollados y en desarrollo, en una asociación global. [ONU, 2015]

Todos, los ciudadanos de este mundo, invierten muchos recursos hacia el ambicioso objetivo de hacer del mundo un lugar mejor, pero no se detienen a analizar si estos esfuerzos realmente los están acercando al logro de las metas.

Por esto, para poder contribuir a un futuro mejor, es un deber conocer los problemas que enfrenta el mundo. Para ello, es necesario recolectar datos cuidadosamente para que

todos vean el estado del mundo actual y rastreen dónde se está progresando y dónde no. Mediante el correcto análisis de datos es posible ver cómo ha cambiado el mundo.

1.3. Idea Directriz de la Tesina

La agenda de SDG dispuesta por la ONU tiene como meta cumplir 17 retos que llevan a la mejora de la calidad de vida en el mundo. Los puntos en los que se basa este trabajo final de carrera para demostrar cómo se puede contribuir al cumplimiento o mejora de los mismos son:

- buena salud,
- educación de calidad,
- innovación y crecimiento en industrias.

Estos tres puntos se ven afectados en el contexto de una pandemia y es de suma importancia aplicar políticas de control, mitigación y crecimiento alrededor de las mismas. Una medida importante es controlar el avance de la enfermedad para aplicar de manera eficiente medidas que eviten el colapso del sistema de salud, que permitan que los alumnos puedan acceder educación de calidad y además que acompañen el crecimiento de las pequeñas y grandes empresas.

Para lograr estos retos en el contexto actual de una pandemia que azota a todos, es necesario gestión, inversión y lo más importante un fuerte control de los avances.

Por un lado, se tiene que pensar en una infraestructura que permita recolectar la mayor cantidad de datos sobre el avance de los cambios que se llevan adelante y por otro lado hay que sacar provecho de esos datos para saber el estado de los avances hasta el momento.

Pensando de esta manera se puede ver que los datos son el bien más importante y valuado en estos casos y por lo tanto es igual de importante seleccionar las mejores estrategias y herramientas para trabajar con ellos.

En base a estas necesidades y metas a cumplir se establece la necesidad de que los organismos tengan herramientas e información sobre las herramientas de análisis de datos para así poder optar por la mejor que sea afín al caso de uso y a los datos con los que tienen que trabajar.

1.4. Hipótesis de Trabajo

La hipótesis a verificar es que bajo el conjunto de métodos y métricas del Framework MyFEPS complementado con benchmarks de rendimiento y test unitarios del libro 'High Performance Python', la librería PySpark ejecutándose en único Nodo (haciendo uso de una única máquina) presenta ventajas de rendimiento y prestaciones en comparación a la librería Pandas.

1.5. Objetivo General y específicos

1.5.1. Objetivo general.

El objetivo principal de este trabajo final de carrear es llevar a cabo todo el análisis, diseño y construcción de dos scripts para el procesamiento de un DataFrame del Gobierno de la Nación Argentina con casos de covid19, con el fin de presentar las diferencias de rendimiento entre PySpark y Pandas.

1.5.2. Objetivos específicos

Los objetivos específicos de este trabajo final de carrera se enumeran a continuación:

1. Desarrollar el marco conceptual con el fin de presentar los conceptos fundamentales del trabajo.

2. Relevar y definir las características/parámetros principales para poder comparar dos herramientas de análisis de datos.
3. Relevar y definir el contexto para poder comparar las herramientas de análisis de datos.
4. Desarrollar los scripts (uno por herramienta a comparar) que cumpla con los parámetros previamente identificados.
5. Armar un Jupyter Notebook interactivo con la finalidad de comparar los scripts y presentar las métricas.
6. Plantear las conclusiones sobre cual herramienta presenta mejor rendimiento en base el contexto y características a comparar.

1.5.3. Objetivos personales

Los objetivos personales de este trabajo final de carrera están relacionados con la integración y puesta en práctica de los conocimientos aprendidos durante la cursada de la carrera de Ingeniería en Informática en la Facultad de Tecnología de la Universidad de Belgrano.

Se pretende implementar los conceptos aprendidos de las siguientes materias:

- Base de datos 1
- Base de datos 2
- Lenguajes de programación
- Ingeniería de software IV

1.6. Metodología

Se aplican Scrum y Kanban cómo metodologías para gestionar el proyecto. Se utiliza el modelo MyFEPS (Metodologías y Framework para la Evaluación de Productos de Software) para la evaluación y comparación objetiva de las herramientas de análisis de datos, Pandas y PySpark.

1.7. Justificación del Trabajo

El proyecto surge a raíz de una necesidad personal y social para lograr determinar de manera profesional las diferencias entre las principales herramientas de análisis de datos que se presentan en la lista de herramientas del entorno Big Data.

Hoy en día los datos toman un grado de criticidad e importancia sin precedentes en el ámbito. El caso más cercano que demuestra esto es la pandemia que azota al mundo desde finales de 2019.

Es importante elegir de manera correcta la herramienta de análisis de datos para ofrecer el mejor tratamiento de datos a la mayor velocidad posible para así poder, por ejemplo, hacer inteligencia de cómo avanza la pandemia y como contrarrestar su poder de contagio e informar a la comunidad de lo que está pasando y de lo que viene.

Dicha comparación permite plantear las bases fundacionales de Big Data, cuales son los retos que presenta, la necesidad mundial actual de obtener y hace uso correcto de datos, cuales son los parámetros a tener en cuenta para comparar herramientas de análisis de datos y determinar cuál de las dos herramientas presenta mejor rendimiento bajo ciertos ejes de comparación y métricas definidas.

1.8. Delimitaciones o alcance de la Tesina

Los límites que propone este trabajo final de carrera abarcan la investigación de las herramientas PySpark y Pandas, el desarrollo de la evaluación y comparativa técnica mediante el procesamiento de un DataFrame, tomando como referencia conjuntos de métodos o métricas del Framework MyFEPS complementado con benchmarks de rendimiento y test unitarios descritos en ‘High Performance Python’.

Dicha comparación entre las librerías se demuestra por medio de un programa realizado en un archivo Jupyter Notebook tras procesar el DataFrame escogido de los casos de covid en Argentina ofrecido por el Gobierno de la Nación conformando el entregable del trabajo final de carrera.

2. MARCO CONCEPTUAL

2.1. Big Data

En la última década hubo un salto en las tecnologías y en la cantidad de datos que estas generan. Esto lleva a la generación de aproximadamente 30.000 gigabytes de datos por segundo, y la tasa de creación de datos sigue aumentando. [WARREN, 2015]

Los datos con los que se lidia día a día son diversos. Por ejemplo, los usuarios crean contenido como publicaciones de blogs, tweets, interacciones de redes sociales y fotos. Todo esto se registra en servidores y de alguna manera hay que hacer uso de estos datos.

El crecimiento de la cantidad de datos ha afectado profundamente a las organizaciones, dejando a los sistemas de bases de datos tradicionales, como las bases de datos relacionales, en el límite de su capacidad. Cada vez se ven más sistemas de datos tradicionales obsoletos bajo la presión de Big Data. Esto se debe a que las tecnologías asociadas a los sistemas relacionales no pueden escalar a la demanda que ejerce Big Data. [WARREN, 2015]

De la mano de la constante y abrumadora cantidad de datos que se generan a diario, nacieron nuevas tecnologías para poder hacer frente a los desafíos que se presentan entorno a Big Data. Muchas de estas nuevas tecnologías se han agrupado bajo el término NoSQL. De alguna manera, son más complejas que las bases de datos tradicionales y, de otra manera, son más simples.

Estos sistemas pueden escalar a conjuntos de datos mucho más grandes, pero el uso efectivo de estas tecnologías requiere un conjunto de técnicas nuevas. Estas herramientas vienen a cumplir un rol específico dentro del entorno Big Data, por lo cual

es necesario tener un profundo conocimiento de cada una para poder seleccionar, según el caso, la que mejor cumpla con nuestros requerimientos.

Big Data se divide en 4 grandes capas, las cuales cumplen un rol a la hora de cumplir con los desafíos: [WARREN, 2015]

- La **capa de datos** es el backend de todo el sistema en la que se almacenan los datos, en cualquier formato, sin procesar provenientes de diferentes fuentes, incluidos sistemas transaccionales, sensores, archivos, datos analíticos; etc. [BOARD INFINITY, 2020]
- La **capa de ingestión o integración** surge a partir de la necesidad de integrar la capa de datos con la capa de procesamiento. Los datos sin procesar ubicados en la capa de datos por su naturaleza y características pueden no estar aptos para ser consumidos directamente por la capa de procesamiento. Por lo tanto, la ingestión se encarga de transformar los datos de manera que se puedan procesar utilizando herramientas y tecnologías específicas desde la capa de procesamiento. [BOARD INFINITY, 2020]
- La **capa de procesamiento** es una de las capas más importantes dentro del stack de tecnología de Big Data, ya que en ella ocurre el procesamiento y análisis de los datos. En esta capa, se producen todas las transformaciones, limpieza y filtro de los datos para finalmente dejarlos en la capa de visualización. [BOARD INFINITY, 2020]
- La **capa de visualización** es la final del stack de tecnologías, que es donde se produce la generación de información y su visualización a través de reportes, gráficos y KPIS. [BOARD INFINITY, 2020]

Figura 1 - Ecosistema Big Data.

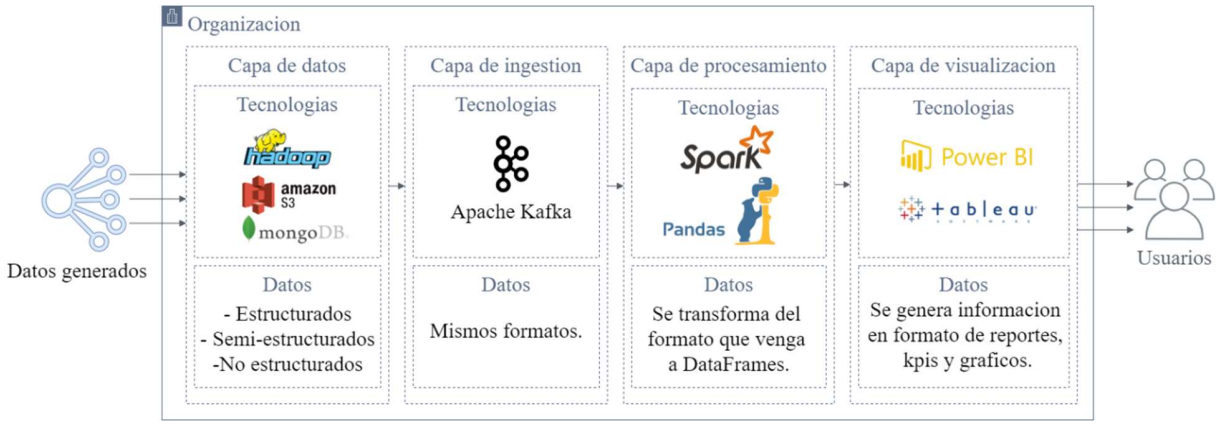


Figura 1: diagrama de las distintas capas en el entorno Big Data, cuales son algunas de las tecnologías que las integran, y que tipos de datos se manejan en cada una. Autoría propia.

2.2. Análisis de datos

El análisis de datos se refiere a procesos de limpieza, transformación y modelado de datos para descubrir información útil para la toma de decisiones. Todo esto se da mayormente en la capa de procesamiento dentro del ecosistema Big Data. El propósito del análisis de datos es extraer información útil de los datos y tomar una decisión basada en el resultado del análisis de datos.

Siempre al tomar alguna decisión se hace en base a lo que pasó la última vez o en lo que pasará al elegir esa decisión en particular. Teniendo en cuenta esto, para cualquier decisión que se tome se debe analizar el pasado y/o futuro.

Sin un correcto análisis de los datos se ven afectadas todas las decisiones que se toman. Para mejorar, innovar o cambiar algo que viene mal es necesario aplicar análisis de datos de la mejor manera.

Para todo análisis de datos se deben seguir con rigurosidad las siguientes fases:

1. **Recopilación de requisitos de datos:** El primer paso es definir el porqué del análisis de datos. Se debe determinar el propósito u objetivo que se quiere cumplir. Se plantea qué analizar y cómo medirlo, porque se está llevando a cabo y cuáles son las métricas a utilizar. [GURU99, 2020]

2. **Recopilación de datos**: Una vez definidos los requisitos y la idea sobre el análisis, se debe recopilar u obtener los datos necesarios en función de los requisitos. [GURU99, 2020]

3. **Limpieza de datos:** Los datos recopilados pueden presentar impurezas o errores que hay que limpiar. Estas impurezas pueden ser registros duplicados, valores vacíos, valores y/o formatos erróneos, etc. Es vital llevar a cabo esta fase de manera correcta ya que si un dato erróneo pasa a la fase de análisis puede afectar el resultado final. [GURU99, 2020]

4. **Análisis de los datos:** Una vez que los datos se recopilan, limpian y procesan, están listos para el análisis. En esta fase se aplican todos los métodos de manipulación de datos como agregación, filtros, orden, agrupación, etc. Es posible que en esta fase se descubra la necesidad de mayor cantidad de datos o de nuevas fuentes de datos. [GURU99, 2020]

5. **Interpretación de datos:** Una vez terminado el análisis se debe obtener información a partir de los datos. En esta etapa se analizan los datos ya procesados en búsqueda de información. [GURU99, 2020]

6. **Visualización de datos:** Los datos se presentan de una forma gráfica y ordenada, orientada a historias, para una mejor comprensión y análisis de los profesionales que tomen las decisiones. [GURU99, 2020]

Figura 2 - Fases en el análisis de datos

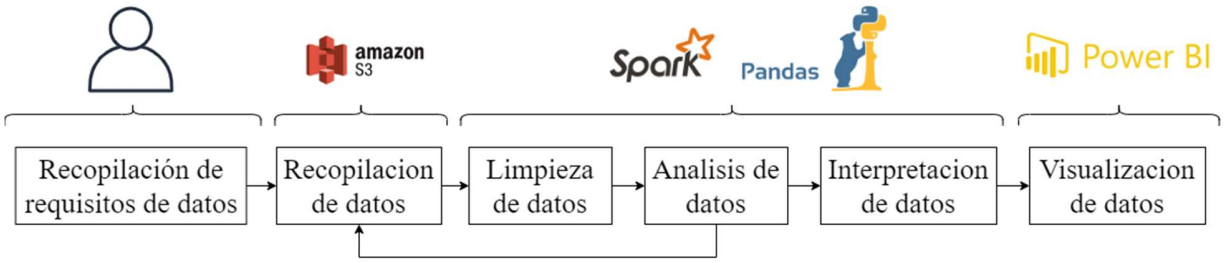


Figura 2: relación entre las distintas capas del análisis de datos y las herramientas que se utilizan en cada una. Autoría propia.

2.3. Dato vs Información

Por un lado, están los datos que son la materia prima usada en Big Data. Se generan a partir de hechos, observaciones, eventos, números, caracteres, etc. Y siempre toman el estado de crudeza (no están procesados, se los toma tal cual se generaron por separado).

Por otro lado, la información surge a partir del procesamiento o tratamiento de un conjunto de datos. Este procesamiento de datos se trata de estructurar, agregar, ordenar y presentar en un contexto, para que tomen un significado y utilidad dado un contexto.

Tabla 1 - Comparativa entre dato e información.

Parámetros	Dato	Información
Descripción.	Variables cualitativas o cuantitativas que ayudan a desarrollar ideas o conclusiones.	Es un grupo de datos que conlleva novedades y significado.

Formato	Los datos están en forma de números, letras o un conjunto de caracteres.	Ideas e inferencias.
Representación	Puede ser estructurado, tabular, gráfico, árbol de datos, etc.	Lenguaje, ideas y pensamientos basados en los datos proporcionados.
Significado	Los datos no tienen ningún propósito específico.	Tiene un significado que se le ha asignado interpretando datos.
Interrelación	Información que se recopila.	Información que se procesa.
Características	Los datos son una sola unidad y están sin procesar. Por sí solo no tiene ningún significado.	La información es el producto y grupo de datos que, conjuntamente, tienen un significado lógico.
Dependencia	Nunca depende de la información.	Depende de los datos.
Soporte para la toma de decisiones	No se puede utilizar para la toma de decisiones.	Es muy utilizado para la toma de decisiones.
Privacidad	Los datos son propiedad de una organización y no están disponibles para la venta al público.	La información está disponible para la venta al público.

Tabla 1: comparativa entre dato e información. [GURU99, 2020].

2.4. Tipos de datos

El enfoque principal de este trabajo final de carrera es sobre los datos estructurados. Para comprender bien que son los datos estructurados, primero hay que conocer que otros dos tipos existen y cuáles son sus principales diferencias.

Figura 3 - Ejemplo formatos de datos.

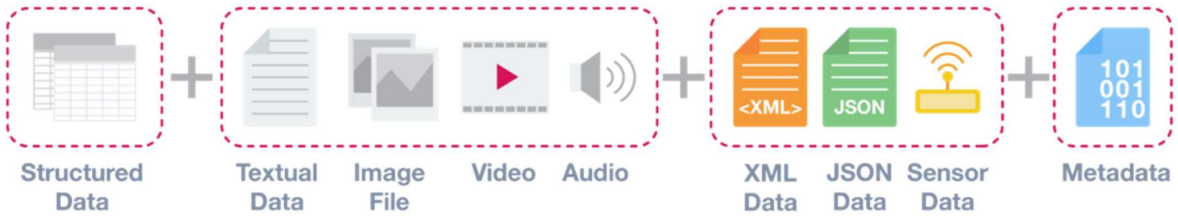


Figura 3: distintos formatos de datos que existen y como se agrupan en distintos conjuntos [BIGDATAFRAMEWORK, 2019]

Datos estructurados: se ajustan a un formato tabular con relación entre las diferentes filas y columnas. Tienen un esquema de datos para comprender que representa cada columna/fila y que tipo de datos acepta en cada una de ellas, y así poder trabajar de una manera organizada y segura sobre cada una de las columnas o filas. Los ejemplos comunes de datos estructurados son archivos de Excel o bases de datos SQL. [BIGDATAFRAMEWORK, 2019]

Tabla 2 - Datos estructurados.

fecha	provincia	confirmados	mujeres_fallecidas	hombres_fallecidos
2020-09-19	Capital Federal	683	2	4
2020-09-19	Buenos Aires	3877	19	13

Tabla 2: ejemplo de datos estructurados para dar a entender lo que el autor pretende. [MINISTERIO DE SALUD, 2020]

Tabla 3 - Representación de esquema para datos estructurados.

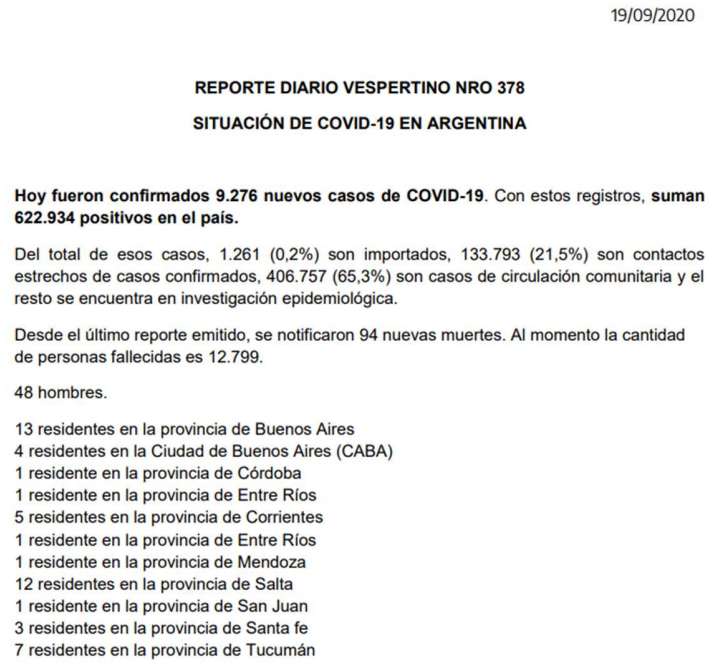
Columna	Tipo de dato
fecha	date
provincia	string
confirmados	integer
mujeres_fallecidas	integer
hombres_fallecidos	integer

Tabla 3: ejemplo del esquema de datos para la Tabla 2. [MINISTERIO DE SALUD, 2020]

Datos no estructurados: son información que no tiene un esquema predefinido o no está organizada de una manera predefinida. Esto da como resultado irregularidades y

ambigüedades que dificultan la comprensión y el uso de los mismos en comparación con los datos estructurados. Los ejemplos comunes de datos no estructurados incluyen archivos de audio, video o bases de datos NoSQL. [BIGDATAFRAMEWORK, 2019]

Figura 4 – Datos no estructurados.



19/09/2020

REPORTE DIARIO VESPERTINO NRO 378

SITUACIÓN DE COVID-19 EN ARGENTINA

Hoy fueron confirmados 9.276 nuevos casos de COVID-19. Con estos registros, **suman 622.934 positivos en el país.**

Del total de esos casos, 1.261 (0,2%) son importados, 133.793 (21,5%) son contactos estrechos de casos confirmados, 406.757 (65,3%) son casos de circulación comunitaria y el resto se encuentra en investigación epidemiológica.

Desde el último reporte emitido, se notificaron 94 nuevas muertes. Al momento la cantidad de personas fallecidas es 12.799.

48 hombres.

13 residentes en la provincia de Buenos Aires
4 residentes en la Ciudad de Buenos Aires (CABA)
1 residente en la provincia de Córdoba
1 residente en la provincia de Entre Ríos
5 residentes en la provincia de Corrientes
1 residente en la provincia de Entre Ríos
1 residente en la provincia de Mendoza
12 residentes en la provincia de Salta
1 residente en la provincia de San Juan
3 residentes en la provincia de Santa fe
7 residentes en la provincia de Tucumán

Figura 4: ejemplo de un caso de datos no estructurados obtenidos de un archivo formato PDF. [MINISTERIO DE SALUD, 2020]

Datos semiestructurados: Los datos semiestructurados son una forma de datos estructurados que no se ajusta a la estructura formal de los modelos de datos asociados con bases de datos relacionales u otras formas de tablas de datos, pero que contienen etiquetas u otros marcadores para separar elementos semánticos y hacer cumplir las jerarquías de filas y campos dentro de los datos. Los ejemplos de datos semiestructurados incluyen JSON y XML. [BIGDATAFRAMEWORK, 2019]

La razón por la que existe esta tercera categoría (entre datos estructurados y no estructurados) es porque los datos semiestructurados son considerablemente más fáciles de analizar que los datos no estructurados. Muchas soluciones y herramientas de Big Data tienen la capacidad de leer y procesar JSON o XML. Esto reduce la complejidad de analizar datos estructurados, en comparación con datos no estructurados.

Figura 5 - Datos semiestructurados:

```
'2020-09-19': [
   {
      'provincia': 'Capital Federal',
      'confirmados': 683,
      'mujeres_fallecidas': 2,
      'hombres_fallecidos': 4
   },
      {
      'provincia': 'Buenos Aires',
      'confirmados': 3877,
      'mujeres_fallecidas': 19,
      'hombres_fallecidos': 13
   }
]
```

Figura 5: ejemplo de la estructura y sintaxis de los archivos JSON. Autoría propia.

El mayor porcentaje de los conjuntos de datos que hoy en día se conocen, se puede transformar a una forma estructurada que sea más adecuada para el análisis y el modelado de datos. Por ejemplo, una colección de reportes epidemiológicos puede procesarse en una tabla para extraer datos relevantes por país (casos positivos, casos negativos, recuperados, densidad poblacional, etc), que luego podría usarse para realizar un análisis de velocidad de contagios según densidad poblacional.

Tabla 4 - Comparativa entre tipos de datos.

Parámetros	Datos estructurados	Datos No estructurados	Datos semi-estructurados
Presencia de esquema.	Si	No	Si
Tipo de archivos.	CSV, Excel, tablas SQL.	PDF, imágenes, videos.	JSON, XML.
Dificultad en manipulación.	Bajo	Alto	Medio

Tabla 4: comparativa entre los distintos tipos de datos que existen que dan a entender lo que el autor pretende. Autoría propia.

2.5. DataFrame

Un DataFrame es una API estructurada que representa una tabla de datos con columnas, filas y un esquema. El esquema es la lista de columnas con los tipos de datos correspondientes.

Figura 6 - Representación de DataFrame

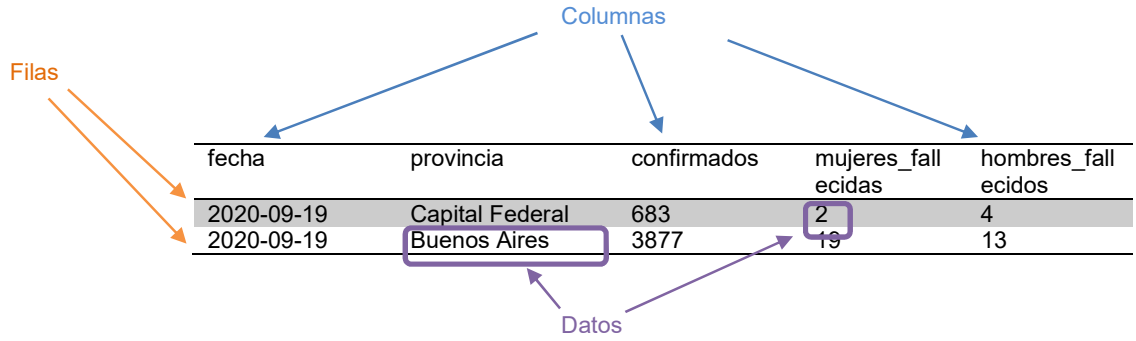


Figura 6: columnas, filas y datos dentro de un DataFrame. Autoría propia.

La API de DataFrame trae como beneficios:

1. **Detección de errores**: en los DataFrames se detectan errores de sintaxis en tiempo de compilación (lo que ahorra tiempo y costos al desarrollador). Por otro lado, no detectara la no existencia de una columna hasta el momento de ejecución.

Tabla 5 - Detección de errores SQL vs DataFrames

	SQL	DataFrames
Error sintaxis	Ejecución	Compilación
Error de análisis	Ejecución	Ejecución

Tabla 5: presenta en que momento de la ejecución de un programa SQL y DataFrames hacen la detección de errores. Autoría propia.

2. **Abstracción de alto nivel en datos estructurados y semiestructurados**: permite generar DataFrames (datos estructurados) a partir de datos semiestructurados (ejemplo JSON) como de otros formatos de datos estructurados (ejemplo queries SQL).
3. **Operaciones/expresiones de alto nivel de abstracción**: filtros, mapas, agregación, promedios, sumas, consultas SQL, acceso a columnas y uso de funciones lambda en datos semiestructurados.

El concepto de DataFrame se encuentra en varios lenguajes de programación como R, Python y Scala. Gracias a esto, es posible convertir un DataFrame de Pandas (Python) a un DataFrame de Spark (PySpark) y viceversa. [DATABRICKS, 2020]

Se hace uso de las API's de DataFrame tanto en PySpark como en Pandas para realizar la comparativa que plantea este trabajo final de carrera.

2.6. Propiedades de un DataFrame

Por lo general, en bases de datos relacionales se actualiza y sumariza constantemente la información para reflejar el estado actual del mundo; no hay necesidad de preocuparse por la inmutabilidad o la perpetuidad de los datos. Este enfoque limita las preguntas que

se pueden responder con esos datos y no logra disuadir firmemente los errores y la corrupción de datos.

Al hacer cumplir estas propiedades en el mundo de Big Data, se logra un sistema más robusto:

Crudeza: Un DataFrame responde preguntas sobre datos que se fue adquiriendo en el tiempo, ósea sobre datos del pasado. A la propiedad de mantener exactamente como están los eventos o datos que se recolectaron en el tiempo se lo conoce como crudeza. De esta manera, cuanto más crudos se almacenen los datos más preguntas se podrán contestar en el futuro. [WARREN, 2015]

Inmutabilidad: En el mundo de las bases de datos relacionales, y también en la mayoría de las otras bases de datos, la actualización es una de las operaciones fundamentales. Pero, para cumplir con la inmutabilidad, no es posible actualizarlos ni eliminarlos, sino que solo se deben agregar más. [WARREN, 2015]

Esta propiedad ofrece dos ventajas:

- **Tolerancia a fallos humanos**: evitando actualizaciones o eliminación de datos y solo agregando más datos, se logra la tolerancia a fallos humanos. Esto permite tener un DataFrame fuente, al cual solo se le agregan nuevos datos a través del tiempo con la tranquilidad de que no se va corromper.
- **Simplicidad**: se evita la dificultad de los modelos de datos, solo permitiendo agregar nuevos datos.

Perpetuidad: La inmutabilidad genera como consecuencia que cada dato es verdadero por siempre. Es decir, un dato, una vez verdadero, debe serlo siempre. [WARREN, 2015]

Es importante tener en cuenta estas propiedades de los DataFrames para saber cómo funcionan y cuál es la razón de ser de los mismos.

Como ejemplo se puede analizar el caso de uso del DataFrame que ofrece el Gobierno de la Nación Argentina (el cual es la base del análisis comparativo). En este caso la propiedad de crudeza del DataFrame se ve en su misma existencia, cada nuevo registro que se carga no se transforma sobre el mismo DataFrame, sino que cualquier análisis que se quiera inferir a partir de él se hace en una segunda instancia sin modificarlo.

La inmutabilidad y perpetuidad se da cuando para un mismo caso se agrega por cada actualización un nuevo registro. Por ejemplo, si hay un caso sospechoso para cierta provincia, y luego se determina que es un caso confirmado, en vez de actualizar el registro que decía sospechoso se genera uno nuevo con mismo id (para poder identificar que se trata del mismo caso) y con una fecha de diagnóstico más reciente (para saber que es la novedad ultima).

2.7. Introducción herramientas a comparar

2.7.1.Apache Spark

Comenzó en UC Berkeley en 2009 como el proyecto de investigación Spark, que se publicó por primera vez al año siguiente en un artículo titulado "Spark: Cluster Computing with Working Sets" de Matei Zaharia, Mosharaf Chowdhury, Michael Franklin, Scott Shenker e Ion Stoica. del AMPlab de UC Berkeley. [ZAHARIA, 2018]

En esa época el motor de programación en paralelo vigente era Hadoop MarReduce. Gracias a Hadoop MapReduce se empezó a ver el potencial de la computación en clusters, pero también lo desafiante e ineficiente que era programar grandes modelos con MapReduce. [ZAHARIA, 2018]

El proyecto de investigación Spark empezó tratando de resolver los problemas de MapReduce y aprovechando en su máximo potencial la computación en paralelo. Mediante librerías estándares y externas, más un motor de computación unificado logro con el paso de los años y el aporte de la comunidad ser una de las principales herramientas de analytics a considerar en la actualidad. [ZAHARIA, 2018]

Apache Spark es un motor informático unificado y un conjunto de bibliotecas para el procesamiento de datos en paralelo en clústeres de computadoras o de forma local (único nodo). Actualmente Spark es el motor de código abierto más desarrollado para esta tarea, lo que lo convierte en una herramienta estándar para cualquier desarrollador o científico de datos interesado en Big Data. [ZAHARIA, 2018]

En torno a Spark se construyeron varias API's para ofrecer soporte a los lenguajes de programación más utilizados (Python, Java, Scala y R), incluye bibliotecas para diversas tareas que van desde SQL hasta transmisión y aprendizaje automático, y se ejecuta en cualquier lugar, desde una computadora portátil hasta un grupo de miles de servidores. Esto hace que sea un sistema fácil de comenzar y escalar hasta el procesamiento de Big Data o una escala increíblemente grande. [ZAHARIA, 2018]

Figura 7 - Set de herramientas en Spark.

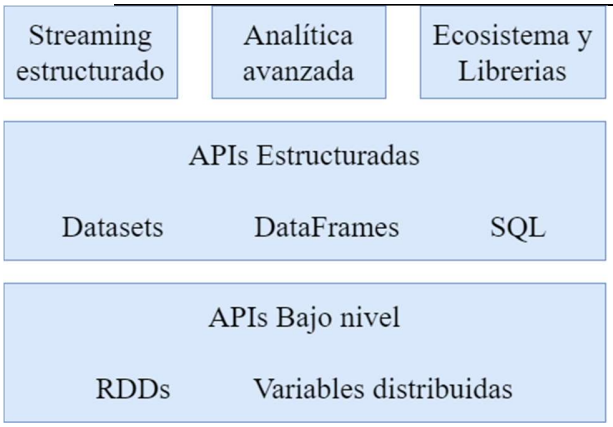


Figura 7: presenta las distintas áreas o grupo de herramientas que conforman el universo de Spark. [ZAHARIA, 2018]

2.7.1.1. Filosofía de Apache Spark

Componentes claves que sostiene la definición teórica de Apache Spark:

1. Unificado

La idea principal detrás de este concepto es el objetivo de que las tareas de análisis de datos del mundo real, ya sean análisis interactivos en una herramienta como Jupyter Notebook o desarrollo de software, tiendan a combinar muchos tipos de procesamiento y librerías diferentes.

Su principal objetivo es ofrecer una plataforma central para realizar cualquier tipo de procesamiento de datos. Está diseñado para admitir una amplia gama de tareas de análisis de datos, que van desde la carga de datos simple y las consultas SQL hasta el aprendizaje automático, sobre el mismo motor informático y con un conjunto consistente de API.

2. Motor de computación

Spark no está hecho para el almacenamiento permanente de los datos. Spark maneja la carga de datos en memoria desde distintas fuentes y realiza los cálculos sobre ellos, pero no los almacena como si fuese una base de datos.

Los sistemas relacionales ofrecen el almacenamiento y poder de computación en el mismo servicio, por ende, no pueden hacer frente a la creciente cantidad de datos existentes. Con este fuerte acoplamiento de dos funcionalidades tan distintas, una limita a la otra imposibilitando la escalabilidad.

Hoy en día existen sistemas de almacenamientos en la nube como Amazon S3, Azure Storage, sistemas de archivos distribuidos como Apache Hadoop, bases de datos clave-valor como Apache Cassandra y brokers de eventos como Apache Kafka, que se encargan de facilitar y brindar un excelente servicio para el almacenamiento de datos a costos mínimos y con la posibilidad de un crecimiento sin límite.

3. Bibliotecas

El ultimo componente son sus bibliotecas, que se basan en su diseño como motor unificado para proporcionar una API unificada para tareas relevantes al análisis de datos.

Admite las bibliotecas estándares que están integradas en el motor, y bibliotecas externas desarrolladas por la comunidad. Spark incluye bibliotecas para SQL y datos estructurados (Spark SQL), aprendizaje automático (MLlib), procesamiento de transmisión (Spark Streaming) y análisis de gráficos (GraphX).

2.7.1.2. Arquitectura de Apache Spark

Por las necesidades y retos presentes en Big Data, como variedad, velocidad y cantidad de datos a procesar, Spark presenta un marco de trabajo distribuido, administrando y coordinando la ejecución de tareas de datos en un grupo de computadoras.

Para poder ejecutar trabajos distribuidos, Spark necesita tener distintos integrantes con funciones específicas y esenciales:

1. El controlador (driver)

Es el controlador de la ejecución de una aplicación Spark y mantiene todo el estado y las tareas de los ejecutores del clúster Spark. Debe interactuar con el administrador del clúster para obtener recursos físicos y ejecutar a los ejecutores.

Por lo tanto, solo es un proceso en una máquina física que es responsable de mantener el estado de la aplicación que se ejecuta en el clúster.

2. Los ejecutores (executors)

Son los procesos que realizan las tareas asignadas por el controlador. Los ejecutores tienen una responsabilidad: tomar las tareas asignadas por el controlador, ejecutarlas e informar sobre su estado (éxito o fracaso) y resultados.

3. Aplicación Spark

Una aplicación Spark es la tarea en si misma que se ejecuta dentro de la arquitectura de Spark. Por ejemplo, una Aplicación Spark es código de usuario que lee un archivo CSV con datos, los guarda dentro de un DataFrame, lo limpia, ordena y luego los escribe en un nuevo archivo CSV.

Para que esta Aplicación Spark pueda llevarse a cabo debe intervenir un proceso controlador y un conjunto de procesos ejecutores.

Figura 8 - Aplicación Spark: Interacción entre partes

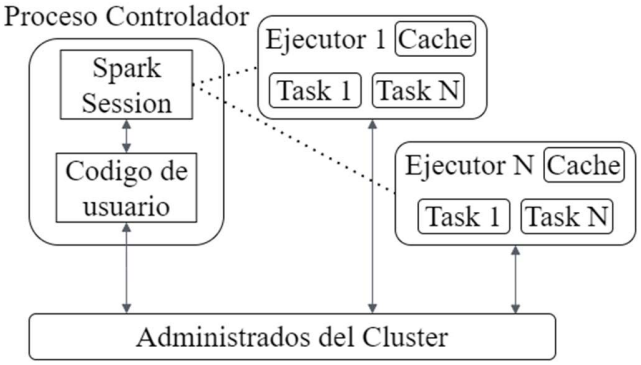


Figura 8: se observa el controlador a la izquierda y dos ejecutores a la derecha. La línea punteada representa que los dos ejecutores se instancian a partir de SparkSession. Las líneas llenas representan la dirección en la comunicación. Autoría propia.

4. El administrador (manager)

El controlador y los ejecutores no pueden existir sin un administrador. El administrador del clúster es responsable de mantener un clúster de máquinas que ejecutarán las Aplicaciones Spark.

Un administrador de clúster tendrá sus propias abstracciones de “maestro” y “trabajador”. La diferencia principal es que están vinculados a máquinas físicas en lugar de procesos.

Figura 9 - Nodos en un cluster

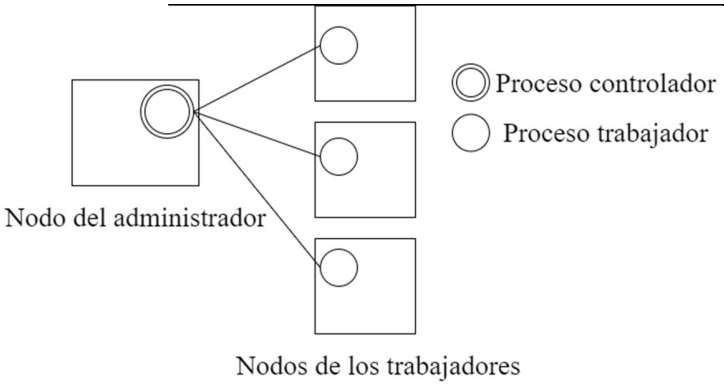


Figura 9: se observa una configuración de clúster básica. La máquina a la izquierda es el nodo maestro donde se encuentra el proceso del controlador. A la derecha se observan tres nodos trabajadores donde, cada uno tiene un proceso del trabajador. Aún no se está ejecutando ninguna Aplicación Spark; estos son solo los procesos del administrador del clúster. Autoría propia.

Cuando llega el momento de ejecutar una aplicación Spark, se solicita recursos al administrador del clúster para ejecutarla. Durante el transcurso de la ejecución de la Aplicación Spark, el administrador del clúster será responsable de administrar las máquinas subyacentes en las que se ejecuta la misma.

2.7.1.3. Modos de ejecución

Antes de configurar Spark y desarrollar cualquier tarea, el primer paso es seleccionar el modo de ejecución. Como se desarrolló con anterioridad, Spark fue pensado desde su concepción para ejecutarse de manera distribuida en un clúster.

Sin embargo, no se limita al modelo de clúster distribuido, sino que también adopta un modo de ejecución llamada Standalone que permite ejecutar aplicaciones Spark en un único nodo (maquina física). Este modo es el punto central de este trabajo final de carrera, ya que permite igualar en condiciones el terreno para comparar Pandas y PySpark.

1. Clúster

En el modo de clúster, un usuario envía un JAR, un script de Python o un script de R pre compilado a un administrador de clúster. Luego, el administrador del clúster inicia el proceso del controlador en un nodo trabajador dentro del clúster, además de los procesos ejecutores.

Figura 10 - Alocando controlador y ejecutores en nodos dentro del clúster

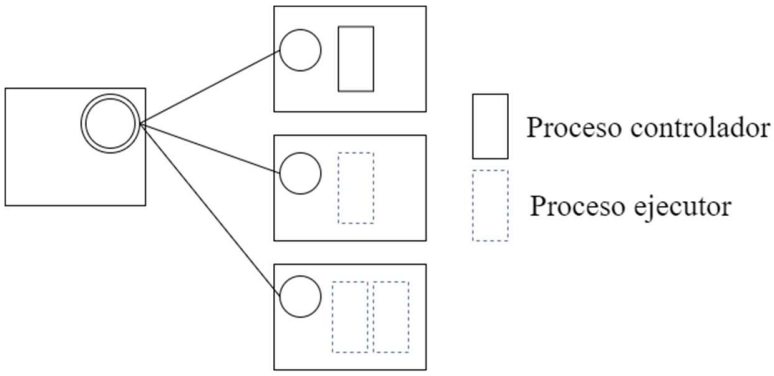


Figura 10: se observa que el administrador del clúster colocó el controlador en un nodo trabajador y los ejecutores en otros nodos trabajadores. Autoría propia.

2. Cliente

El modo de cliente es casi el mismo que el modo de clúster, excepto que el controlador Spark permanece en la máquina cliente que envió la solicitud.

Figura 11 - Alocando controlador y ejecutores modo cliente

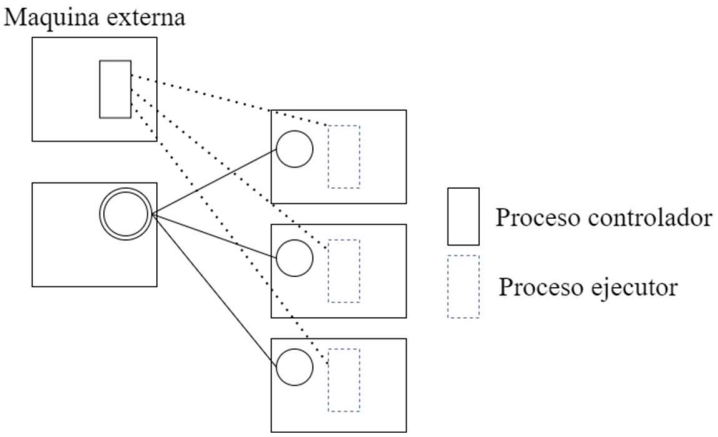


Figura 11: se observa que el controlador se está ejecutando en una máquina fuera del clúster, pero que los trabajadores están ubicados en las máquinas del clúster. Autoría propia.

3. Standalone

El modo local ejecuta toda la aplicación Spark en una sola máquina. Logra el paralelismo a través de hilos en esa única máquina. Esta es una forma común de aprender Spark, probar sus aplicaciones o experimentar iterativamente con el desarrollo local. El modo local ya viene por defecto dentro de Apache Spark.

En modo local, el controlador y los ejecutores se ejecutan (como subprocesos) en la computadora individual en lugar de ejecutarse en un grupo de máquinas. Para cada subproceso se asigna una cantidad de memoria y núcleos del CPU. La cantidad de nodos es en base a la cantidad de núcleos que estén disponibles.

Figura 12 - Modo Standalone

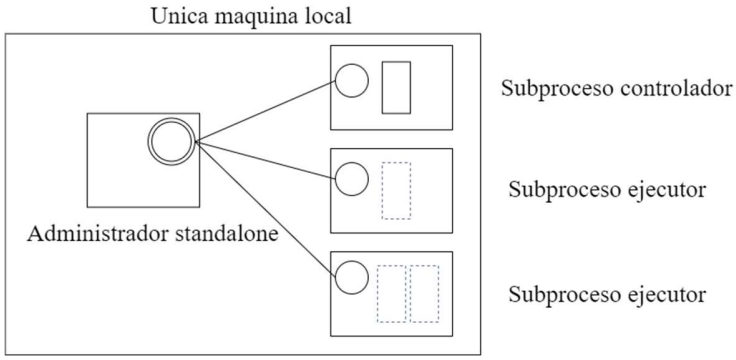


Figura 12: se observa que el modelo es parecido al modo clúster, pero con la diferencia que todo sucede en una misma máquina, y que cada nodo es un subproceso con CPU y memoria asignada.

2.7.1.4. Ciclo de vida de una Aplicación Spark

Una aplicación Spark es la tarea a ejecutar, que en el caso de ejecución en un único nodo requiere un controlador y ejecutores.

A continuación, se presentan las etapas que se llevan a cabo a la hora de ejecutar una Aplicación en un único nodo. Se detalla cómo se comporta la infraestructura, quienes son los participantes, cuales son los pasos y como se comunican.

1. Requerimiento del cliente

Desde la maquina local del cliente se realiza la solicitud al nodo administrador. En este paso, se están solicitando explícitamente recursos para alocar el controlador.

2. Lanzamiento

Ahora que el proceso del controlador se ha colocado en un nodo trabajador, es posible comenzar a ejecutar el código de usuario. La tarea del usuario debe incluir una SparkSession que es la encargada de solicitar al nodo administrador los recursos según la cantidad de ejecutores que tenga definido.

3. Ejecución

El controlador y los ejecutores se comunican entre ellos, ejecutando código y moviendo datos. El controlador programa tareas para cada ejecutor y cada ejecutor responde con el estado de esas tareas y el éxito o el fracaso.

4. Terminación

Cuando el proceso controlador finaliza con éxito o falla (según la respuesta de todos los ejecutores), el administrador apaga o interrumpe los nodos trabajadores donde residen los ejecutores.

Figura 13 - Diagrama de secuencia ciclo de vida Aplicación Spark

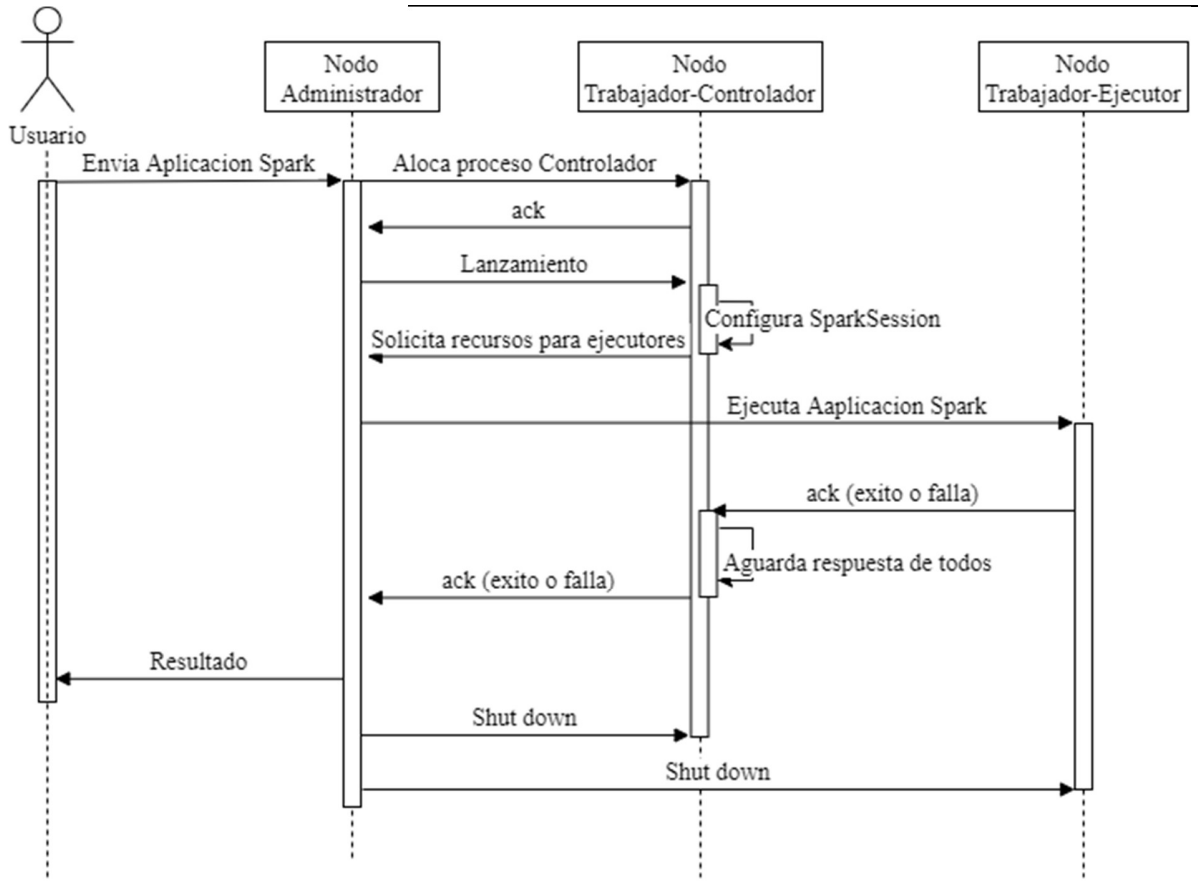


Figura 13: diagrama de secuencia desde que el usuario envía una Aplicación Spark al clúster Standalone hasta que recibe una respuesta con el resultado de la ejecución. Autoría propia.

2.7.2.Pandas

El desarrollo de Pandas se inició en 2008 por Wes McKinney; y se concedió como código abierto en 2009. Es una biblioteca de Python que contiene herramientas y estructuras de datos de alto nivel que se han creado para ayudar a los programadores de Python a realizar análisis de datos de gran alcance. [MCKINNEY, 2012]

El propósito último de Pandas es descubrir rápidamente información en los datos brindando un conjunto de APIS para adaptarse rápidamente a los desafíos del requerimiento.

Pandas se diseñó inicialmente teniendo en cuenta las finanzas, específicamente por la necesidad de manipular datos de series de tiempo y procesar información histórica de acciones. [MCKINNEY, 2012]

Algunas de las características más relevantes que presenta Pandas y se harán uso para el desarrollo de la comparativa son:

- DataFrame rápidos y eficientes para la manipulación de datos con indexación integrada;
- Manejo integrado de datos faltantes;
- La capacidad de procesar datos almacenados en muchos formatos comunes como CSV, Excel, HDF5 y JSON;
- Las columnas se pueden insertar y eliminar de las estructuras de datos para la mutabilidad de tamaño;
- Agregar o transformar datos con una potente función de agrupación de datos para dividir, aplicar y combinar en conjuntos de datos;
- Fusión y unión de conjuntos de datos;
- Indexación jerárquica que facilita el trabajo con datos de alta dimensión en una estructura de datos de menor dimensión;
- Altamente optimizado para el rendimiento, con rutas de código críticas escritas en los lenguajes de programación Python o C.

El amplio y solido conjunto de funciones, combinado con su perfecta integración con Python y otras herramientas dentro del ecosistema de Python más el uso e integración de los cuadernos de Jupyter, le da a Pandas una amplia adopción en muchos dominios. Se utiliza en finanzas, neurociencias, economía, estadísticas, publicidad y analítica web.

Gracias al rendimiento, productividad y capacidad de colaboración, se ha convertido en una de las herramientas preferidas por los científicos de datos para representar datos para su manipulación y análisis. [REFERENCIA A USO POR LA COMUNIDAD]

2.7.2.1. Arquitectura de Pandas

Se compone de varias librerías dispuestas de forma jerárquica:

1. **pandas/core**: contiene archivos para estructuras de datos fundamentales como Series / DataFrames y funcionalidades relacionadas.
2. **pandas/src**: contiene código Cython y C para implementar algoritmos fundamentales.
3. **pandas/io**: contiene herramientas de entrada/salida (como archivos planos, Excel, SQL, etc.).
4. **pandas/tools**: contiene algoritmos de datos auxiliares que combinan y unen rutinas, concatenación, tablas dinámicas y más.
5. **pandas/sparse**: contiene versiones dispersas de Series, DataFrame y más.
6. **pandas/stats**: contiene regresión lineal, y regresión de ventana móvil.
7. **pandas/util**: contiene utilidades, desarrollo y herramientas de prueba.
8. **pandas/rpy**: contiene la interfaz RPy2 para conectarse a R.

Cada librería a su vez se compone de sub módulos (scripts de Python) que se encargan de ofrecer todas las funcionalidades necesarias para leer, manipular y escribir datos.

Este trabajo final de carrera no incluye dentro de sus límites analizar en profundidad los sub módulos integrantes de cada librería, sino estrictamente de la comparativa funcional entre PySpark y Pandas.

2.8. Ejes de comparación

Para realizar la comparativa entre ambas herramientas se definen los siguientes ejes de comparación:

Tabla 6 - Ejes de comparación

Ejes de comparación	Resumen
Runtime	JVM vs Cython.
Lenguaje de programación	Scala vs Python para Spark.
Modos de ejecución	Perezoso vs Ansioso.
Escalabilidad de procesamiento	Comportamiento y requerimientos en sistemas Distribuidos vs Locales.
Escalabilidad de datos	Comportamiento y requerimientos en sistemas Distribuidos vs Locales.
Ecosistema de aplicaciones	En IDEs, Machine Learning y Visualizaciones.
Soporte y comunidad	Apoyo y movimiento en GitHub de los repositorios fuente de cada herramienta.
Instalación	Pasos y configuraciones necesarias para realizar la instalación.
Manipulación CSV	Funciones y métodos para la manipulación de archivos con formato CSV.
Manipulación DataFrames	Funciones y métodos para la manipulación de DataFrames.
Transformaciones	Funciones y métodos para realizar transformaciones en los datos.

Tabla 6: breve resumen de los ejes que se evalúan y comparan. Autoría propia.

2.8.1.Runtime

Pandas

Pandas en está escrito en una mezcla de Python y Cython (con soporte de escritura estática, que transforma el código Python en C y luego compila el código C para brindar un rendimiento similar a C), con todas las ventajas y desventajas del lenguaje interpretado y tipado dinámicamente. Pero dado que la mayoría de las transformaciones finalmente se realizan mediante un código C optimizado y de bajo nivel, el rendimiento es suficiente para la mayoría de los casos de uso.

Spark

Spark se implementa en el lenguaje de programación Scala, que apunta a la máquina virtual Java (JVM). A diferencia de Python, Scala es un lenguaje compilado y tipado estáticamente. Spark no se basa en el código C / C ++ optimizado de bajo nivel, sino que todo el código se optimiza sobre la marcha durante la ejecución por el compilador Java Just-in-Time (JIT).

2.8.2.Lenguaje de programación

Al comparar Spark y Pandas, hay que incluir una comparación de los lenguajes de programación compatibles con cada herramienta.

Pandas solo se puede usar con Python, pero Spark permite usar varios lenguajes de programación como Scala y Python.

Python vs Scala

La diferencia principal entre Scala y Python es que Scala es un lenguaje fuertemente tipado, es decir, cada variable y parámetro tiene un tipo fijo y Scala inmediatamente arroja un error si intenta usar un tipo incorrecto, en cambio Python se tipea dinámicamente, es decir, una sola variable o parámetro puede aceptar cualquier tipo de dato - aunque el código puede asumir tipos específicos y por lo tanto fallar más tarde durante la ejecución.

Esta diferencia tiene un gran impacto:

1. Modelo de ejecución

Python es un lenguaje interpretado, lo que significa que Python puede ejecutar inmediatamente cualquier código, siempre que sea una sintaxis válida de Python. No se requiere ningún paso de compilación.

Esto hace que Python sea una excelente opción para el trabajo interactivo, ya que Python puede ejecutar el código inmediatamente a medida que lo escribe.

Scala, por otro lado, es un lenguaje compilado, lo que significa que un compilador de Scala primero necesita transformar el código de Scala en el llamado código de bytes de Java para la JVM (que a su vez se traduce al código de máquina nativo durante la ejecución).

Este enfoque de tres pasos (escribir, compilar, ejecutar) a menudo dificulta los experimentos de código, ya que los tiempos de entrega son mayores.

2. Curva de aprendizaje

Python es muy simple de aprender: fue diseñado específicamente para ser así con un fuerte enfoque en la legibilidad. El sistema de tipo dinámico de Python es adecuado para principiantes, que nunca han tenido contacto con un lenguaje de programación.

Scala, por otro lado, tiene una curva de aprendizaje mucho más pronunciada y, a diferencia de Python, el código puede volverse rápidamente difícil de leer para los principiantes.

3. Robustez del código

Los lenguajes de escritura dinámica tienen una gran desventaja sobre los lenguajes de escritura estática: el uso de un tipo incorrecto solo se detecta durante el tiempo de ejecución y no antes (durante el tiempo de compilación).

Esto significa que, si se llama a una función con un tipo de datos incorrecto en algunas condiciones muy raras, es posible que solo se note cuando sea demasiado tarde (en producción).

En cambio, un lenguaje compilado y tipado estáticamente evita que se entregue el código con errores a producción. Los errores se detectan a tiempo y se deben solucionar antes de que el compilador pueda terminar su trabajo.

2.8.3.Modelo de ejecución

Pandas

Pandas implementa un modelo de ejecución ansioso, lo que significa que cualquier transformación a un DataFrame que aplique se ejecutará de inmediato. Este enfoque para el usuario es excelente para el trabajo interactivo (como suelen hacer los científicos de datos), pero también tiene algunas desventajas: Pandas no puede crear un plan de ejecución optimizado fusionando múltiples operaciones, ya que cada transformación se realizará de inmediato.

Spark

Una de las grandes ventajas que presenta Spark es la evaluación perezosa o Lazy Evaluation en su lenguaje de origen. Esto quiere decir que Spark espera hasta el último momento para ejecutar las instrucciones o funciones de cálculo sobre los datos. En lugar de modificar los datos inmediatamente cuando lee alguna operación, crea un plan de transformaciones que considera más conveniente a aplicar a los datos de origen.

Internamente, Spark utiliza las siguientes etapas:

1. Se crea un plan de ejecución lógica a partir de las transformaciones especificadas por el desarrollador.
2. Luego, se deriva un plan de ejecución analizado donde se verifican y resuelven todos los nombres de las columnas y las referencias externas a las fuentes de datos
3. Luego, este plan se transforma en un plan de ejecución optimizado mediante la aplicación iterativa de las estrategias de optimización disponibles. Por ejemplo, las operaciones de filtrado se envían lo más cerca posible de las fuentes de datos para reducir el número de registros lo antes posible.

4. Finalmente, el resultado del paso anterior se transforma en un plan de ejecución físico donde las transformaciones se canalizan juntas en las llamadas etapas.

Este enfoque general es muy similar a todas las bases de datos relacionales, que también optimizan las consultas SQL con estrategias similares antes de ejecutarlas.

2.8.4.Escalabilidad de procesamiento

Pandas

El procesamiento en sí es de un solo subproceso y no se puede distribuir a diferentes máquinas. Por muy malo que parezca, Pandas sigue siendo muy útil y puede ser muy rápido, ya que la mayor parte del trabajo se realiza dentro de un backend C / C ++ altamente optimizado.

Spark

Spark es intrínsecamente de múltiples subprocesos y puede hacer uso de todos los núcleos de su máquina. Además, Spark fue diseñado desde el principio para realizar su trabajo en grandes clústeres con posiblemente cientos de máquinas y miles de núcleos.

Al dividir la cantidad total de trabajo en tareas individuales, que luego se pueden procesar de forma independiente en paralelo, Spark hace un uso muy eficiente de los recursos del clúster disponibles.

2.8.5.Escalabilidad de datos

Pandas

Pandas requiere que todos sus datos quepan en la memoria principal de la computadora local. No puede persistir datos en el disco ni puede distribuir datos dentro de un grupo de máquinas.

Spark

Spark también escala muy bien con grandes cantidades de datos. No solo se escala a través de múltiples máquinas en un clúster, sino que además brinda la capacidad de persistir resultados intermedios en el disco. Por lo tanto, Spark casi nunca está limitado por la cantidad total de memoria principal, sino solo por la cantidad total de espacio disponible en disco.

Esto deriva de que, al seguir perezosamente los planes de ejecución, el conjunto total de datos de trabajo nunca se materializa completamente en la RAM en ningún momento. Todos los datos (incluidos los datos de entrada y los resultados intermedios) se dividen en pequeños fragmentos, que se procesan de forma independiente e incluso los resultados se almacenan eventualmente en pequeños fragmentos y esos nunca se guardan en la RAM a la vez.

2.8.6.Ecosistema de aplicaciones

Hoy en día el éxito de un lenguaje de programación no está ligado solamente a su sintaxis o sus conceptos, sino a su ecosistema.

Para todo proyecto de análisis de datos se requiere un IDE (Integration Development Environment) interactivo, completo y fácil de usar, herramientas o modelos de estadística y aprendizaje automático para poder aplicar transformaciones y obtener información, y por ultimo librerías para poder graficar los resultados.

Tabla 7 - Ecosistema Pandas vs PySpark

Áreas	Pandas	PySpark
IDE	• IPython • Jupyter Notebook	• Jupyter Notebook
Estadística y Machine Learning	• Pandas-tfrecords • Statsmodels	• Machine Learning Library (MLlib)

	• Sklearn-pandas • Featuretools • Compose	
Visualización	• Altair • Bokeh • Seaborn • Plotnine • IPython vega • Plotly • Qtpandas • D-Tale • hvplot	• Bokeh • Seaborn • Plotnine • Plotly

Tabla 7: principales áreas a comparar y las herramientas que existen para satisfacer sus respectivas necesidades. Autoría propia.

Pandas puede hacer uso de todas las librerías que tenga el ecosistema de Python, por eso es una herramienta tan utilizada, para cada problema que se presente es factible encontrar la solución. [ECOSISTEMA PANDAS, 2020]

Spark, por otro lado, vive en un universo completamente diferente. Como participante del mundo JVM, puede utilizar todo tipo de bibliotecas Java, pero el enfoque de la mayoría de las bibliotecas Java son las redes, los servicios web y las bases de datos. Los algoritmos numéricos no están en el dominio central de Java. [ECOSISTEMA SPARK, 2020]

Al usar PySpark se puede hacer uso de librerías del ecosistema de Python, y así llegar a un espectro más amplio de aplicaciones. Aunque la integración llega más tarde que en el caso de Pandas, y hasta en algunas ocasiones no llega nunca.

Por este motivo PySpark se ve más limitado que Pandas en cuanto a la posibilidad de elección de distintas herramientas o soluciones que mejor se amolden a nuestro problema.

2.8.7.Soporte y comunidad

El soporte y la comunidad activa representan la aceptación y apoyo que tiene cada herramienta o tecnología open source. Para tener una visión clara del apoyo de cada herramienta se consulta el repositorio fuente, para ambos casos GitHub. [GITHUB PANDAS, 2020] [GITHUB SPARK, 2020]

Tabla 8 - Soporte y comunidad.

Tópico	Descripción	Pandas	Spark
Stars	Cantidad de usuarios a los que les gusta el proyecto y lo siguen activamente.	28.2k	28.6k
Fork	Cantidad de usuarios que mantienen una copia del proyecto en sus repositorios personales, pudiendo modificarlos sin afectar el proyecto principal.	11.7k	23.3k
Releases	Cantidad de versiones/paquetes entregados a la comunidad.	128	144
Contributors	Cantidad de usuarios que hicieron al menos un commit al proyecto.	2249	1610
Used by	Cantidad de repositorios o proyectos que dependen del repositorio actual.	354k	549

Tabla 8: tópicos principales a evaluar para comprender el soporte y la comunidad que mantiene cada herramienta hasta la fecha 15/02/2021. Autoría propia.

Otro aspecto a tener en cuenta es la licencia, la cual define los términos y condiciones para el uso y distribución del código fuente. [GITHUB PANDAS, 2020] [GITHUB SPARK, 2020]

Tabla 9 - Licencias.

Herramienta	Licencia	Permisos	Limitaciones

Pandas	BSD 3-Clause	• Uso comercial • Modificaciones • Distribución • Uso privado	• Responsabilidad • Garantía
Spark	Apache License 2.0	• Uso comercial • Modificaciones • Distribución • Uso patentado • Uso privado	• Uso de marcas comerciales • Responsabilidad • Garantía

Tabla 9: presenta los permisos y limitaciones dados en las licencias de cada herramienta hasta la fecha 15/02/2021. Autoría propia.

3. DESARROLLO DEL TRABAJO

3.1. Conceptos base para el desarrollo de la comparativa

3.1.1. Comma separated values (CSV)

Los archivos CSV (valores separados por comas) se usan generalmente para intercambiar datos tabulares entre sistemas usando texto sin formato. CSV es un formato de archivo basado en filas, lo que significa que cada fila del archivo es una fila en la tabla. Las columnas se diferencian a través del separador y las filas con los saltos de línea '\n'. [AIDA NGOM, 2020]

CSV contiene una fila de encabezado que contiene nombres de columna para los datos; de lo contrario, los archivos se consideran parcialmente estructurados. A diferencia de una base relacional, las conexiones de datos se establecen mediante varios archivos CSV, las claves externas se almacenan en columnas de uno o más archivos, pero el formato en sí no expresa las conexiones entre estos archivos. [AIDA NGOM, 2020]

El formato CSV no está completamente estandarizado, lo que significa que puede o no contener encabezado y además pueden usar separadores que no sean comas, como tabulaciones '\t', espacios '\e' o punto y coma ';'.

Figura 14 - Extracto CSV.

```
id_evento_caso,sexo,edad,edad_años_meses,residencia_pais_nombre,residencia_provincia_nombre,residencia_
departamento_nombre,carga_provincia_nombre,fecha_inicio_sintomas,fecha_apertura,sepi_apertura,fecha_intern
acion,cuidado_intensivo,fecha_cui_intensivo,fallecido,fecha_fallecimiento,asistencia_respiratoria_mecanica,carga
_provincia_id,origen_financiamiento,clasificacion,clasificacion_resumen,residencia_provincia_id,fecha_diagnostic
o,residencia_departamento_id,ultima_actualizacion
1000000,M,53,Años,Argentina,CABA,SIN ESPECIFICAR,Buenos
Aires,,6/1/2020,23,,NO,,NO,,NO,6,Privado,Caso Descartado,Descartado,2,6/9/2020,0,9/19/2020
1000002,M,21,Años,Argentina,Buenos Aires,La Matanza,Buenos Aires,,6/1/2020,23,,NO,,NO,,NO,6,Público,Caso
Descartado,Descartado,6,6/1/2020,427,9/19/2020
1000003,F,40,Años,Argentina,Córdoba,Capital,Córdoba,5/24/2020,6/1/2020,23,,NO,,NO,,NO,14,Privado,Caso
Descartado,Descartado,14,6/1/2020,14,9/19/2020
```

Figura 14: ejemplo de las primeras tres filas del CSV con casos covid Argentina. [MINISTERIO DE SALUD, 2020]

Ventajas

- Es legible por humanos y fácil de editar manualmente;
- Proporciona un esquema simple;
- Casi todas las aplicaciones existentes pueden procesar CSV;
- Tiene conectores para ser usado desde Pandas y PySpark;
- Es fácil de implementar y analizar.

Desventajas

- No admite valores nulos, estos se representan como valores vacíos;
- Sin soporte para tipos de columna. No hay diferencia entre las columnas numéricas y de texto;
- Soporte deficiente para caracteres especiales;
- Falta de un estándar universal.

A pesar de las limitaciones y los problemas, los archivos CSV son una opción popular para el intercambio de datos, ya que son compatibles con una amplia gama de aplicaciones científicas, comerciales y de consumo. [LUMINOUSMEN, 2020]

3.1.2.Jupyter Notebook

El proyecto Jupyter es un proyecto de código abierto sin fines de lucro, nacido del Proyecto IPython en 2014, que evolucionó para respaldar la ciencia de datos interactivos y la computación científica en todos los lenguajes de programación. Se desarrolla activamente en GitHub, a través del consenso de la comunidad de Jupyter. [JUPYTER ORG, 2020]

Jupyter Notebook es una aplicación que permite crear y compartir documentos, conocidos como Notebooks, que incluyen código, ecuaciones, visualizaciones y texto narrativo de manera interactiva. Se suele implementar para limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos, aprendizaje automático y mucho más.

Un Notebook es un documento único en el que se puede ejecutar código, mostrar el resultado y también agregar explicaciones, fórmulas, gráficos y hacer que el análisis sea más transparente, comprensible, repetible y compartible.

El uso de Notebooks es ahora una parte importante del flujo de trabajo de ciencia de datos en empresas de todo el mundo.

Figura 15 – Ejemplo Jupyter Notebook

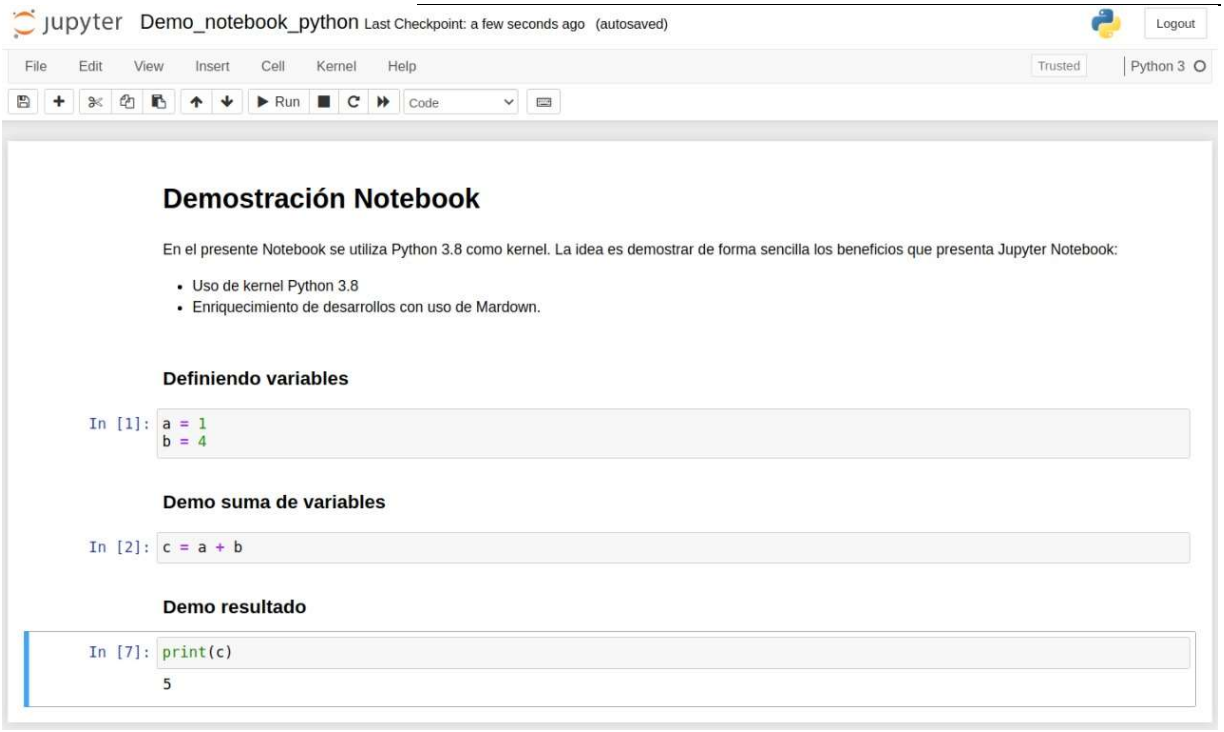


Figura 15: se observa un extracto para visualizar y comprender como es la estructura de un Notebook y como integra celdas de código, resultados interactivos y textos narrativos para enriquecer los análisis. Autoría propia.

Ventajas:

- Soporte para más de 40 lenguajes de programación, como Python, R y Scala;
- Posibilidad de compartir Notebooks a través de email, Dropbox, GitHub, etc;
- El código puede producir una salida rica e interactiva con el uso de HTML, imágenes, videos y LaTeX;
- Permite aprovechar las herramientas de Big Data en un mismo notebook, como Apache Spark, de Python, R y Scala, Pandas, scikit-learn, ggplot2, y/o TensorFlow.

3.1.3.Python para el análisis de datos

Desde su primera aparición en 1991, Python se ha convertido en uno de los lenguajes de programación más populares. Entre los lenguajes interpretados, Python se distingue por su amplia y activa comunidad informática científica. La adopción de Python para la computación científica tanto en aplicaciones industriales como en investigación académica ha aumentado significativamente desde principios de la década de 2000. [JET BRAINS, 2020]

Figura 16 - Popularidad del lenguaje de programación Python.

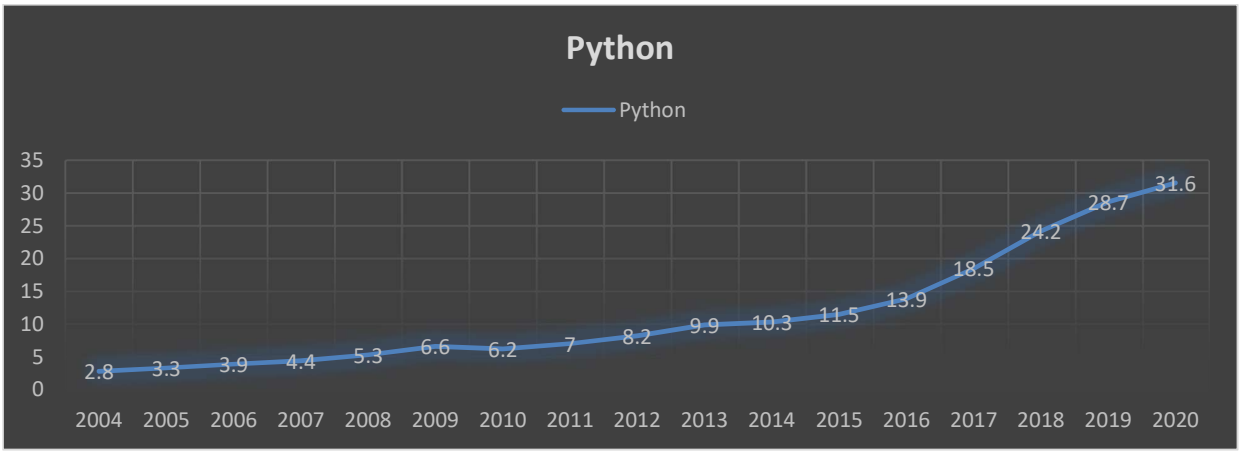


Figura 16: porcentaje de popularidad que ocupa en el mercado el lenguaje de programación Python. [PYPL, 2020]

En los últimos años, el soporte mejorado de las bibliotecas de Python, como por ejemplo Pandas, lo ha convertido en una alternativa sólida para las tareas de manipulación de datos. Combinado con la fortaleza de Python en la programación de propósito general, es una excelente opción como lenguaje único para crear aplicaciones centradas en datos.

Tabla 10 - Ranking de los principales lenguajes para el análisis de datos.

Ranking	Lenguaje de programación	Participación	Tendencia
1	Python	31.56 %	+2.9 %
7	R	4.08 %	+0.3 %
11	Matlab	1.65 %	-0.1 %
17	Scala	0.93 %	-0.2 %

Tabla 6: ranking de popularidad de los distintos lenguajes de programación, tomando como base un año atrás. [PYPL, 2020]

Cabe aclarar que no se trata de presentar a Python como el mejor lenguaje de programación, sino que se trata de justificar la razón de su elección para desarrollar los scripts presentes en este trabajo final de carrera.

Si bien Python es un entorno excelente para construir muchos tipos de aplicaciones de análisis de datos y sistemas de propósito general, hay una serie de usos para los que Python puede ser menos adecuado. Como Python es un lenguaje de programación interpretado, en general, la mayor parte del código Python se ejecutará más lento que el código escrito en un lenguaje compilado como Java o C ++.

En algunos proyectos y trabajos el tiempo del programador suele ser más valioso que el tiempo de la CPU, por este motivo es necesario hacer una compensación y determinar si se está dispuesto a perder tiempo en CPU y ganar tiempo a la hora del desarrollo. Para este trabajo final de carrera se opta por hacer un trade off, eligiendo el tiempo de desarrollo por sobre el tiempo de CPU.

3.1.4.MyFEPS

MyFEPS, de sus siglas Metodologías y Framework para la Evaluación de Productos de Software, cómo su nombre bien lo describe es un procedimiento, que detalla una serie

de procesos y subprocesos, para la evaluación de productos de software. Paralelamente al procedimiento, provee un marco de trabajo, artefactos y una serie de indicaciones para ejecutar la evaluación del producto de software.

En este trabajo final de carrera se evalúan las siguientes características con sus respectivas sub-características:

- Eficiencia - En los tiempos de respuesta.

Para evaluar esta sub-característica se toman tres muestras de tiempo por cada funcionalidad a comparar. Con la ayuda de la librería time en Python se toman los tiempos de ejecución, luego se guardan en una lista y por último se calcula la media de los resultados, tal cómo se define en MyFEPS.

Dado que se evalúan tiempos de ejecución para comparar dos herramientas, se exceptúa la definición de MOEF (Minutos de operación Estándar) y el resultado de la Valoración.

Función definida:

```
# Tiempos de respuesta
# Cantidad de iteraciones
n = 3

# Inicializar en cero sumDTP
sumDTP = 0

# Lista con Minutos hombre por cada iteración
MHO = []

# For loop para recorrer cada valor de MHO y sumarlo
for i in range(n):
        sumDTP = MHO[i] + sumDTP

# Media de tiempos
MTO = sumDTP / n
```

- Eficiencia - En la utilización de memoria interna.

Mediante el uso de la librería psutil de Python, se toma la cantidad de Memoria usada por la Función en Carga Alta (CMF(x)) y la Cantidad de Memoria del sistema (CMS). Luego se calcula la valoración cómo 1 - (CMF/CMS).

Se repite tres veces este procedimiento y luego se determina el valor final cómo la media de la valoración parcial.

Función definida:

```
# Memoria interna utilizada
# Cantidad de memoria del sistema
CMS = p.memory_full_info()[7]

# For loop para calcular la media de memoria utilizada.
for key in result[herramienta]['memoria']:
        # Inicializar en cero CMF
        CMF = 0
        for value in result[herramienta]['memoria'][key]:
                CMF = CMF + value

        # Media de memoria
        CMF = CMF / len(result[herramienta]['memoria'][key])

        # Valoracion
        valoracion = 1 - (CMF/CMS)
```

- Eficiencia - En la utilización de CPU.

Mediante el uso de la librería psutil de Python, se toma Porcentaje de CPU para la Función en Carga Alta (Carga(x)). Luego se calcula la valoración cómo 1 - (Carga/100).

Se repite tres veces este procedimiento y luego se determina el valor final cómo la media de la valoración parcial.

Función definida:

```
# CPU utilizado
# Porcentaje total de uso por CPU
cpu = 100

# For loop para calcular la media del porcentaje de CPU utilizado.
for key in result[herramienta]['cpu']:

        # Inicializar en cero porcentaje inicial de CPU usado.
        carga = 0
        for value in result[herramienta]['cpu'][key]:
                carga = carga + value

        # Media del CPU utilizado
        carga = carga / len(result[herramienta]['cpu'][key])

        # Valoracion
        valoracion = 1 - (carga / cpu)
```

- Instalable - Primera instalación.

Se define el óptimo número de horas hombre para la primera instalación en entorno de uso (ONHHPI), partiendo de la base de tener Python previamente instalado y configurado.

Con la ayuda de un cronómetro se mide el número de horas hombre para la primera instalación en entorno de uso (NHHPI), lo cual conlleva la instalación y configuración de la herramienta hasta poder realizar un import desde un notebook de Jupyter Notebook.

Por último, se aplica la siguiente fórmula para ponderar la herramienta: M = ((NHHPI - ONHHPI) / ONHHPI) - 1

Esta prueba se lleva a cabo por un usuario con previo conocimiento sobre el proceso de instalación, así se puede evaluar el tiempo real que toma cada instalación.

- Instalable - Actualizaciones.

Se define el óptimo número de horas hombre para la primera instalación en entorno de uso (ONHHPI), partiendo de la base de tener Python y la herramienta en cuestión previamente instalados y configurados.

Con la ayuda de un cronómetro se mide el número de horas hombre para la instalación de una nueva versión en entorno de uso (NHHPI), lo cual conlleva la actualización de la herramienta y luego poder realizar un import desde un notebook de Jupyter Notebook.

Por último, se aplica la siguiente fórmula para ponderar la herramienta: M = ((NHHPI - ONHHPI) / ONHHPI) - 1

Esta prueba se lleva a cabo por un usuario con previo conocimiento sobre el proceso de actualización, así se puede evaluar el tiempo real que toma cada instalación.

Para información en detalle sobre el significado y método de evaluación de las características y sub-característica definidas consultar **Anexo II: MyFEPS**.

3.1.5.Parámetros a analizar, comparar y ponderar

Para evaluar y luego comparar herramientas de software es necesario definir los parámetros y métricas para los requerimientos tanto funcionales como no funcionales.

Dentro de los parámetros se define:

- Jupyter Notebook como herramienta base de desarrollo de código y visualización de resultados;
- Lenguaje de programación Python por su rápida curva de aprendizaje y perfecta integración con Jupyter Notebook, Pandas y PySpark;
- Lectura y escritura de valores en formato CSV, por ser un estándar en el manejo de datos y por ser el formato elegido para reportar los casos de covid;

- Los requerimientos funcionales se evalúan en torno del procesamiento y tratamiento de un DataFrame, ya que este se encuentra presente en ambas herramientas y es un tipo de esquema de datos estructurados ampliamente utilizado;
- Se hará uso de MyFEPS como fuente de Metodologías y Framework para la implementación de la evaluación y comparativa.

Funcionalidades:

- **Manipulación CSV:**
  - Lectura CSV;
  - Escritura CSV;

- **Manipulación DataFrames:**
  - Creación;
  - Select;
  - Agregar columnas;
  - Renombrar columnas;
  - Eliminar columnas;
  - Cambio de tipo de columnas;
  - Filtro de filas;
  - Filtro de filas únicas;
  - Orden de filas;

- **Transformaciones:**
  - Joins;
  - Groupby;
  - Count;
  - Min;
  - Max;

- Avg;

3.1.6.Instalación y actualizaciones

Tanto Spark como Pandas van a ejecutarse en Python (PySpark y Pandas).
Para evaluar los tiempos de la primera instalación y la actualización de versiones, se parte de la base de tener los primeros 3 pasos del **Anexo III: Guía de instalación y configuración del entorno de trabajo**.

Además, se toma como parte de la primera instalación de cada herramienta la instalación de Jupyter Notebook y los subsiguientes pasos hasta llegar a importar la herramienta en un nuevo notebook. Esto es así ya que se considera a Jupyter Notebook cómo un complemento necesario para poder iniciar a trabajar.

3.1.6.1. Primera instalación

Para ambas herramientas se establece como óptimo número de horas hombre para la primera instalación en entorno de uso (ONHHPI) en 0.17 hs (~10 minutos).

PySpark con Apache Spark 2.4.7
La instalación de PySpark requiere de un previo conocimiento sobre las dependencias que requiere y las versiones de las mismas.

Tabla 11 - Versiones y dependencias para Spark.

Spark	Scala	Java	Python
2.4.7	2.11	8	3.7
3.0.1	2.12	8	3.8

Tabla 11: dependencias y sus respectivas versiones necesarias para poder corres Spark. Autoría propia.

Los pasos desarrollados para esta primera instalación comprenden los pasos 4, 5, 6, 8, 9, 10, 11, 12 y 13 del **Anexo III: Guía de instalación y configuración del entorno de trabajo**.

Ponderación:
ONHHPI = 0.17
NHHPI = 0.11
M = ((NHHPI - ONHHPI) / ONHHPI) - 1
M = ((0.11 - 0.17) / 0.17) - 1
M = -1.35

Pandas

No es necesario configurar ninguna variable de entorno ni instalar dependencias por separado. El controlador de paquetes pip se encarga de evaluar que es necesario para que la herramienta funcione e lo instala de ser necesario.

Los pasos desarrollados para esta primera instalación comprenden los pasos 7, 8, 10, 11, 12 y 13 del **Anexo III: Guía de instalación y configuración del entorno de trabajo**.

Ponderación:
ONHHPI = 0.17
NHHPI = 0.055
M = ((NHHPI - ONHHPI) / ONHHPI) - 1
M = ((0.055 - 0.17) / 0.17) - 1
M = -1.68

3.1.6.2. Actualización de versiones

Para ambas herramientas se establece como óptimo número de horas hombre para la instalación de una nueva versión en entorno de uso (ONHHPI) en 0.08 hs (~5 minutos).

PySpark

Se actualiza de la versión 2.4.7 a la 3.0.1

Ponderación:

ONHHPI = 0.08

NHHPI = 0.042

M = ((NHHPI - ONHHPI) / ONHHPI) - 1

M = ((0.042 - 0.08) / 0.08) - 1

M = -1.48

Pandas última versión

Se actualiza de la versión 1.0.0 a la 1.2.3

Ponderación:

ONHHPI = 0.08

NHHPI = 0.005

M = ((NHHPI - ONHHPI) / ONHHPI) - 1

M = ((0.005 - 0.08) / 0.08) - 1

M = -1.94

3.1.7.Manipulación CSV

3.1.7.1. Lectura

Dado la manera de ejecución de cada herramienta, perezosa en PySpark y ansiosa en Pandas, la prueba de lectura se dificulta. Cuando se leen archivos CSV con Pandas los lee y persiste en una variable de manera automática. Por otro lado, cuando se leen archivos CSV con PySpark no los está trayendo realmente hasta que se aplique alguna función de transformación.

Por este motivo, la lectura de los archivos CSV se deja como un paso necesario para poder completar las siguientes evaluaciones.

Pandas
Se hace uso de la funcionalidad ‘read_csv’.

Parámetros utilizados:
pandas.read_csv(filepath, sep=',', header='infer')

Código:
pd_250 = pd.read_csv(f'data/250mb.csv', sep=',', header='infer')

PySpark
Se hace uso de la funcionalidad ‘read’.

Parámetros utilizados:
spark.read.option("header",True).option("sep", ",").option("inferSchema", "true").csv(filepath)

Código

```
# Creating Spark session
appName = f"Test"
master = 'local'
spark = SparkSession.builder \
    .master(master) \
    .appName(appName) \
    .getOrCreate()

ps_250 = spark.read.option("header",True).option("sep", ",").option("inferSchema",
"true").csv(f'data/250mb.csv')
```

3.1.7.2. Escritura

Ambas herramientas ofrecen distintos formatos para persistir en disco los datos. En este trabajo final de carrera se evalúa la posibilidad y los tiempos que toma la escritura en formato CSV.

Pandas

Se hace uso de la funcionalidad ‘to_csv’.

Parámetros utilizados:

DataFrame.to_csv(path_or_buf=None, sep=',', header=True, index=True)

Código:

```
df_pd = pandas[key].to_csv(f'./output/{key}_escritura_pandas.csv', sep=',',
header=True, index=False)
```

PySpark

Se hace uso de la funcionalidad 'write'.

Parámetros utilizados:

DataFrame.write.mode('overwrite').option("header","true").csv(filepath)

Código:

```
df =
spark_dict[key].write.mode('overwrite').option("header","true").csv(f'./output/{key}_escritura_pyspark.csv')
```

Resultados

Tiempos por iteración:

```
{
   'pandas': {
      '600mb': [22.94665789604187, 17.637106895446777, 22.826310873031616],
      '200mb': [9.05095100402832, 9.148581743240356, 9.377422332763672],
      '1800mb': [57.284857988357544, 57.26997494697571, 53.530219316482544]
   },
   'pyspark': {
      '600mb': [21.589548110961914, 20.989014387130737, 21.4204843044281],
      '200mb': [9.733651161193848, 8.7040433883667, 8.814811944961548],
      '1800mb': [66.81619429588318, 65.39513492584229, 66.19224190711975]
   }
}
```

Figura 17 - Promedio tiempo escritura

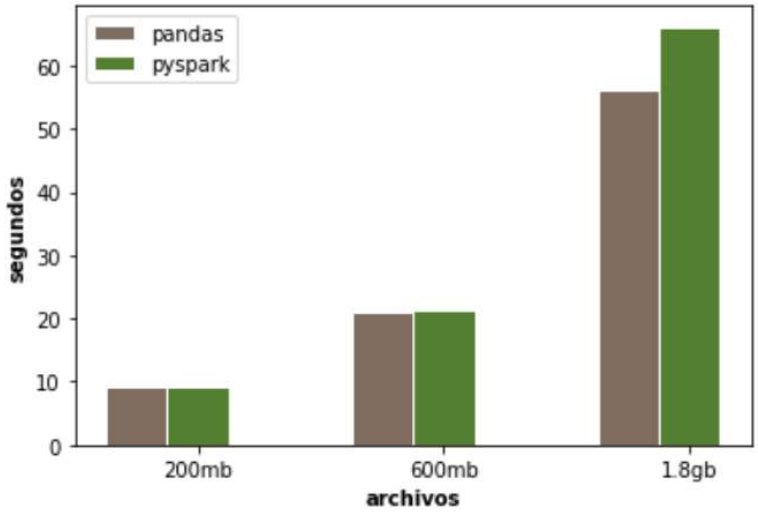


Figura 17: tiempo promedio de escritura por archivo para cada herramienta. Autoría propia.

3.1.8.Manipulación DataFrames

3.1.8.1. Creación

Un DataFrame se puede crear a partir de la lectura de un archivo con datos (cualquier formato aceptado por las herramientas) o a partir de un arreglo de datos (listas, diccionarios, etc).

En este apartado se evalúa la posibilidad de crear un DataFrame a partir de un arreglo de datos y los tiempos que lleva.

Pandas

Se hace uso de la funcionalidad 'DataFrame'.

Parámetros utilizados:

pandas.DataFrame(np.random.randint(0,100,size=(numero_filas,
numero_columnas), columns = array(strings))

Código:

df_pd = pd.DataFrame(np.random.randint(0,100,size=(filas, 4)),
columns=columnas_list)

PySpark

Se hace uso de la funcionalidad 'createDataFrame'.

Parámetros utilizados:

RandomRDDs.uniformVectorRDD(spark, numero_filas,
numero_columnas).map(lambda a : a.tolist()).toDF(array(strings))

Código:

df_ps = RandomRDDs.uniformVectorRDD(spark, filas,4).map(lambda a :
a.tolist()).toDF(columnas_list)

Resultados

Tiempos por iteración

{
'pandas': {
'200': [0.00039267539978027344, 0.0002186298370361328,
0.0002105236053466797],
'600': [0.0003914833068847656, 0.0003910064697265625,
0.0004012584686279297],
'1800': [0.0005419254302978516, 0.0004405975341796875, 0.00042724609375]

```
    },
    'pyspark': {
        '200': [0.4421522617340088, 0.5197796821594238, 0.5292215347290039],
        '600': [0.6251654624938965, 0.5315561294555664, 0.5684871673583984],
        '1800': [0.5495848655700684, 0.5422930717468262, 0.48700499534606934]
    },
}
```

Figura 18 - Promedio tiempo creación

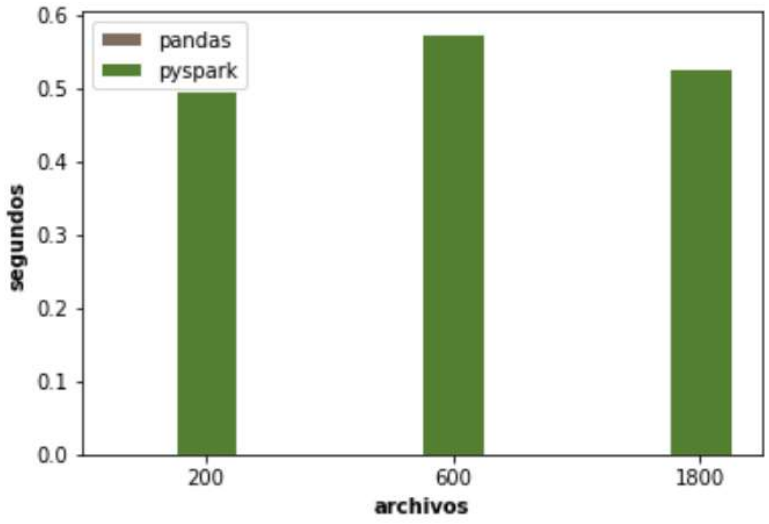


Figura 18: tiempo promedio de creación de un DataFrame a partir de un arreglo de datos. Autoría propia.

3.1.8.2. Select

Cuando se hace un Select se está realizando la selección de un sub conjunto de datos de un DataFrame. Este sub conjunto puede ser a partir de las columnas y/o de las filas.

Se realiza la evaluación y se calculan los tiempos para un Select donde se filtra un sub conjunto de filas y columnas.

Pandas

Se hace uso de la funcionalidad 'head' para filtrar filas y '[array(strings)]' para filtrar las columnas.

Parámetros utilizados:

Dataframe[array(strings)].head(integer)

Código:

```
df_pd = pandas[key][
        ["id_evento_caso","sexo","edad","fecha_inicio_sintomas","clasificacion_res
umen"]
    ].head(5)
```

PySpark

Se hace uso de la funcionalidad 'limit' para filtrar filas y 'select' para filtrar las columnas.

Parámetros utilizados:

DataFrame.select(array(strings)).limit(integer)

Código:

```
df = spark_dict[key]\
        .select("id_evento_caso","sexo","edad","fecha_inicio_sintomas","clasificaci
on_resumen")\
        .limit(5)
```

Resultados

Tiempo por iteración

{
'pandas': {
'600mb': [0.07107877731323242,0.06836247444152832,0.05698847770690918],
'200mb': [0.07350850105285645, 0.037667036056518555,
0.024095296859741211],
'1800mb': [0.20477557182312012, 0.2550787925720215, 0.19771575927734375]
},
'pyspark': {
'600mb': [0.1770784854888916,0.16184711456298828,0.1427624225616455],
'200mb': [0.8215694427490234, 0.11728906631469727, 0.12237215042114258],
'1800mb': [0.18016576766967773, 0.16768598556518555,
0.19508910179138184]
}
}

Figura 19 - Promedio tiempo Select

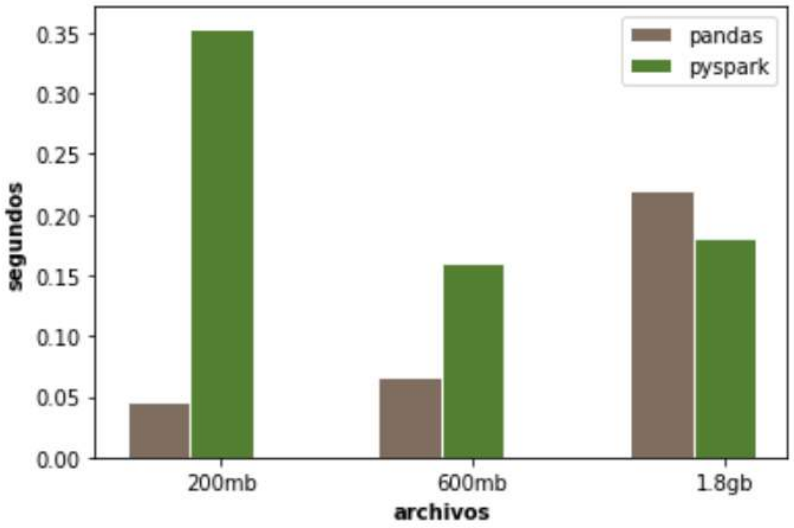


Figura 19: tiempo promedio del Select por archivo para cada herramienta. Autoría propia.

3.1.8.3. Agregar columnas

Agregar columnas en un DataFrame previamente creado.

Pandas

Se hace uso de la funcionalidad 'DataFrame[column]'.

Parámetros utilizados:

DataFrame[column]

Código:

df_pd['nueva_columna'] = df_pd['clasificacion_resumen']

PySpark

Se hace uso de la funcionalidad 'withcolumn'.

Parámetros utilizados:

DataFrame.withColumn(column, value)

Código:

```
df = spark_dict[key].withColumn("nueva_columna", col("clasificacion_resumen"))
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [ 3.3827855587005615, 3.380985975265503, 3.4345271587371826],
      "200mb": [0.448502779006958, 0.3274075984954834, 0.3381004333496094],
      "600mb": [0.7599363327026367, 0.7525498867034912, 0.7272038459777832]
   },
   "pyspark": {
      "1800mb": [ 0.11200284957885742, 0.13601112365722656,
0.1160581111907959],
      "200mb": [0.11784148216247559, 0.1291182041168213, 0.07892560958862305],
      "600mb": [0.13703203201293945, 0.07977008819580078,
0.11626291275024414]
   }
}
```


Figura 20 - Promedio tiempo Agregar columnas

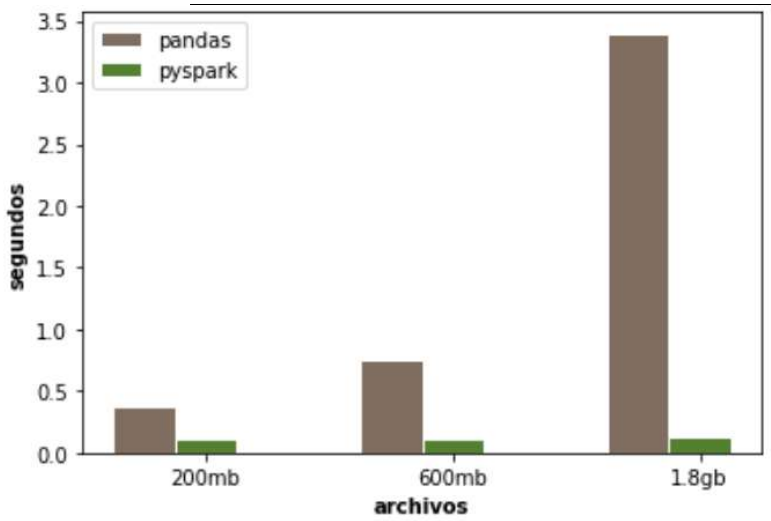


Figura 20: tiempo promedio para agregar columnas por archivo para cada herramienta. Autoría propia.

3.1.8.4. Renombrar columnas

Renombrar columnas en un DataFrame previamente creado.

Pandas

Se hace uso de la funcionalidad 'rename'.

Parámetros utilizados:

DataFrame.rename(columns={old_column: new_column})

Código:

```
df_pd = df_pd.rename(columns={'sexo': 'genero'})
```

PySpark

Se hace uso de la funcionalidad 'withColumnRenamed'.

Parámetros utilizados:

DataFrame.withColumnRenamed(old_column, new_column)

Código:

```
df = spark_dict[key].withColumnRenamed('sexo', 'genero')
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [5.758801698684692, 4.6929309368133545, 4.540667295455933],
      "200mb": [0.37262821197509766, 0.413724422454834, 0.4118027687072754],
      "600mb": [1.1210901737213135, 1.200824499130249, 1.1797425746917725]
   },
   "pyspark": {
      "1800mb": [0.11611795425415039, 0.10382437705993652,
0.12794804573059082],
      "200mb": [0.19290566444396973, 0.08384823799133301,
0.08106732368469238],
      "600mb": [0.1631782054901123, 0.06534385681152344, 0.11467599868774414]
   }
}
```


Figura 21 - Promedio tiempo Renombrar columnas

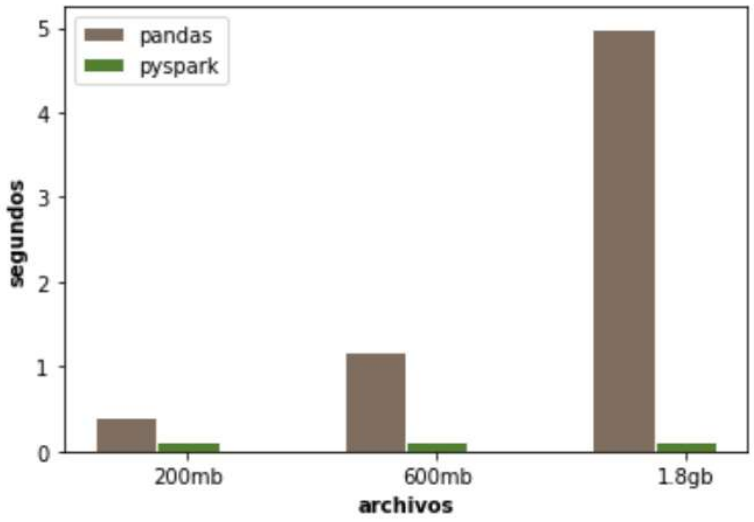


Figura 21: tiempo promedio para Renombrar columnas por archivo para cada herramienta. Autoría propia.

3.1.8.5. Eliminar columnas

Eliminar columnas en un DataFrame previamente creado.

Requerimientos:

columnas = [
"edad_años_meses","carga_provincia_nombre","sepi_apertura","asistencia_respir
atoria_mecanica"
]

Pandas

Se hace uso de la funcionalidad ‘drop’.

Parámetros utilizados:

DataFrame.drop(columns=array(string))

Código:

df_pd = df_pd.drop(columns=columnas)

PySpark

Se hace uso de la funcionalidad 'drop'.

Parámetros utilizados:

DataFrame.drop(*array(string))

Código:

df = spark_dict[key].drop(*columnas)

Resultados

Tiempos por iteración

{
"pandas": {
"1800mb": [6.168460845947266, 3.6317429542541504, 3.5335872173309326],
"200mb": [0.8436994552612305, 0.4017674922943115, 0.411268949508667],
"600mb": [1.8375482559204102, 1.0420050621032715, 1.0081591606140137]
},
"pyspark": {
"1800mb": [0.09990572929382324, 0.1000680923461914, 0.1120309829711914],
"200mb": [0.16148090362548828, 0.10410809516906738,
0.11986708641052246],

"600mb": [0.14039134979248047, 0.12366938591003418,
0.11996269226074219]
}
}

Figura 22 - Promedio tiempo Eliminar columnas

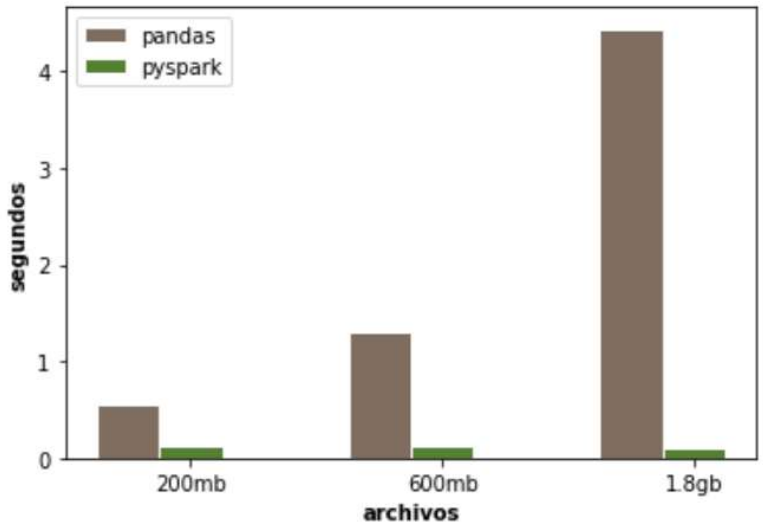


Figura 22: tiempo promedio para Eliminar columnas por archivo para cada herramienta. Autoría propia.

3.1.8.6. Filtro de filas

En este apartado se evalúa y calculan los tiempos para el filtrado de un DataFrame previamente creado dado una cláusula de condición.

Pandas

Se hace uso de la funcionalidad ‘loc’.

Parámetros utilizados:

DataFrame.loc[(condition)][column_to_show]

Código:

df_pd = df_pd.loc[(df_pd['sexo']=='F')]['edad']

PySpark

Se hace uso de la funcionalidad 'filter'.

Parámetros utilizados:

DataFrame.select(column_to_show).filter(condition)

Código:

df = spark_dict[key].select(spark_dict[key].edad).filter(spark_dict[key]['sexo'] ==
"F")

Resultados

Tiempos por iteración

{
"pandas": {
"1800mb": [1.259523868560791, 1.2432262897491455, 1.2898695468902588],
"200mb": [0.5778446197509766, 0.14796686172485352, 0.15366768836975098],
"600mb": [0.40010738372802734, 0.4011831283569336, 0.3515622615814209]
},
"pyspark": {

"1800mb": [0.06848812103271484, 0.07136058807373047,
0.04822087287902832],
"200mb": [0.2968177795410156, 0.06409668922424316, 0.06996560096740723],
"600mb": [0.0740816593170166, 0.05631566047668457, 0.06476640701293945]
}
}

Figura 23 - Promedio tiempo Filtro de filas

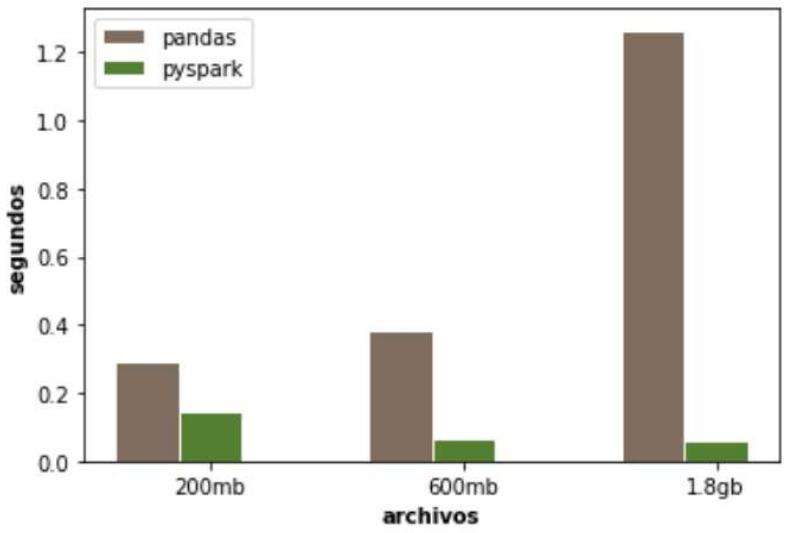


Figura 24: tiempo promedio para filtro de filas por archivo para cada herramienta. Autoría propia.

3.1.8.7. Filtro de filas únicas

En este apartado se evalúa y calculan los tiempos para el filtrado de únicas filas de un DataFrame.

Pandas

Se hace uso de la funcionalidad 'drop_duplicates'.

Parámetros utilizados:

DataFrame.drop_duplicates()

Código:

df_pd = df_pd.drop_duplicates()

PySpark

Se hace uso de la funcionalidad 'drop_duplicates'.

Parámetros utilizados:

DataFrame.drop_duplicates()

Código:

df = spark_dict[key].drop_duplicates()

Resultados

Tiempos por iteración

{
"pandas": {
"1800mb": [20.35026240348816, 18.993958473205566, 19.02502465248108],
"200mb": [1.7495057582855225, 1.8004539012908936, 1.73382568359375],
"600mb": [4.6009461879730225, 4.747037887573242, 4.778027772903442]
},
"pyspark": {

"1800mb": [74.51453709602356, 76.96151828765869, 61.16250681877136],
"200mb": [9.70053768157959, 8.005526781082153, 8.013718605041504],
"600mb": [25.1970694065094, 19.703218936920166, 20.32112216949463]
}
}

Figura 24 - Promedio tiempo Filtro de filas únicas

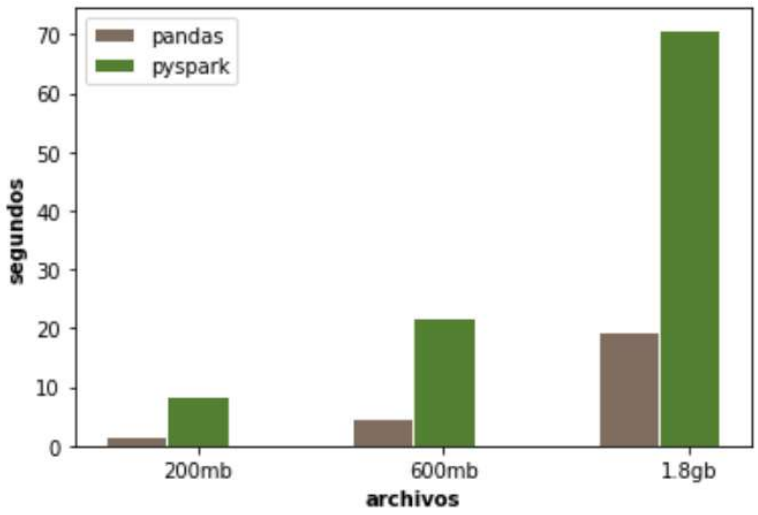


Figura 25: tiempo promedio para Filtro de filas únicas por archivo para cada herramienta. Autoría propia.

3.1.8.8. Cambio de tipo de columnas

Siempre es recomendable trabajar las columnas de fechas con tipo DateTime y no String. Se evalúa y calculan los tiempos para transformar una columna tipo String a DateTime para un DataFrame previamente creado.

Pandas

Se hace uso de la funcionalidad ‘to_datetime’.

Parámetros utilizados:

pandas.to_datetime(column, format='%Y-%m-%d')

Código:

```
df_pd['fecha_apertura'] =  pd.to_datetime(df_pd['fecha_apertura'], format='%Y-%m-
%d')
```

PySpark

Se hace uso de la funcionalidad ‘to_date’.

Parámetros utilizados:

DataFrame.withColumn(old_column, to_date(column, format))

Código:

```
df = spark_dict[key].withColumn('fecha_apertura',
                to_date(col('fecha_apertura'), 'yyyy-MM-dd'))
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [4.328090667724609, 2.331166982650757, 2.468911647796631],
      "200mb": [0.8346021175384521, 0.3562641143798828, 0.27077293395996094],
      "600mb": [1.3028481006622314, 0.6397833824157715, 0.6051347255706787]
```

},
"pyspark": {
"1800mb": [0.10786032676696777, 0.12007927894592285,
0.11194276809692383],
"200mb": [0.23809409141540527, 0.11025333404541016,
0.12401127815246582],
"600mb": [0.14397954940795898, 0.10800886154174805,
0.10800552368164062]
}
}

Figura 25 - Promedio tiempo Cambio tipo de columnas

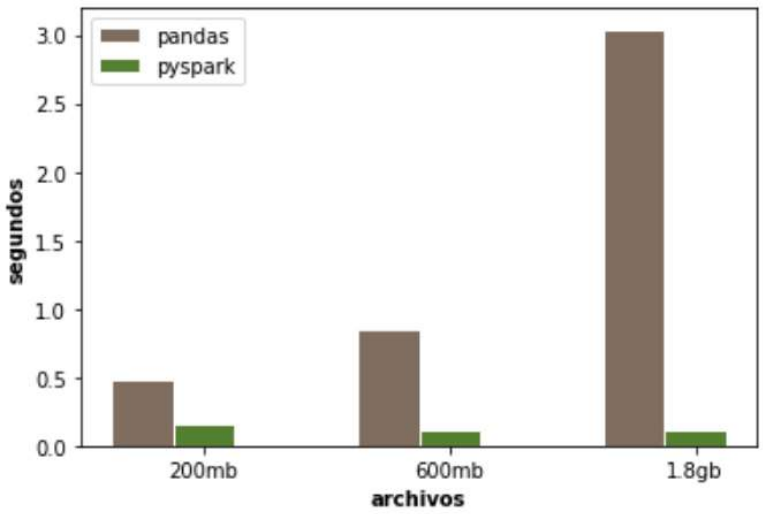


Figura 23: tiempo promedio para cambio tipo de columnas por archivo para cada herramienta. Autoría propia.

3.1.8.9. Orden de filas

Ordenar filas según los valores de las columnas en un DataFrame previamente creado.

Pandas

Se hace uso de la funcionalidad 'sort_values'.

Parámetros utilizados:

DataFrame.sort_values(by=column, ascending=True)

Código:

```
df_pd = pandas[key].sort_values(by='id_evento_caso',ascending=True)
```

PySpark

Se hace uso de la funcionalidad 'orderBy'.

Parámetros utilizados:

DataFrame.orderBy(column.asc())

Código:

```
df = spark_dict[key].orderBy(col("id_evento_caso").asc())
```

Resultados

Tiempos por iteración

{
"pandas": {
"1800mb": [1.768789291381836, 3.224034547805786, 1.8821392059326172],
"200mb": [0.5369458198547363, 0.18391633033752441, 0.18428373336791992],

```
    "600mb": [0.4986586570739746, 0.5498032569885254, 0.5407295227050781]
  },
  "pyspark": {
    "1800mb": [35.926474809646606, 35.56608605384827, 35.87405323982239],
    "200mb": [7.431195497512817, 5.081226587295532, 5.414663314819336],
    "600mb": [14.945232391357422, 11.839983463287354, 11.969556331634521]
  }
}
```


Figura 26 - Promedio tiempo Orden de filas

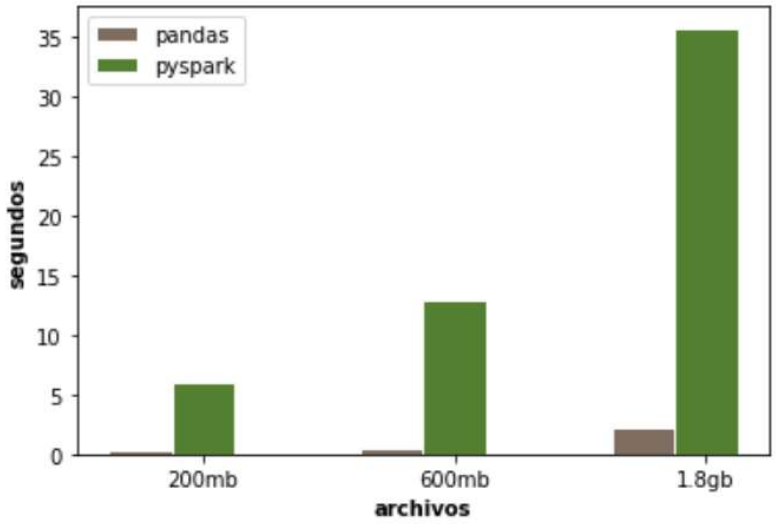


Figura 26: tiempo promedio para el orden de filas por archivo para cada herramienta. Autoría propia.

3.1.9.Transformaciones

3.1.9.1. Joins

Agregar valor al DataFrame mediante la unión con otro DataFrame.

Para realizar esta prueba es necesario el tratamiento previo de dos DataFrame. Para la medición de los tiempos no se toma en cuenta el procesamiento previo de ambos DataFrame.

Pandas

Se hace uso de la funcionalidad ‘merge’.

Parámetros utilizados:

pd.merge(df_1, df_2, how='inner', on=column_name)

Requisitos previos:

```
# Dataset con coordenadas de las provincias
provincias_temp = pd.read_csv(archivo_provincias, sep=',', header='infer')
provincias_temp = provincias_temp.assign(
    provincia=[*zip(provincias_temp.centroide_lat, provincias_temp.centroide_lon)]
)
provincias = provincias_temp[['provincia', 'iso_nombre']]
provincias['iso_nombre'] = provincias['iso_nombre'].replace('Ciudad Autónoma de Buenos Aires','CABA')
# Dataset casos covid Argentina.
casos = pandas[key]
casos_argentina = casos[casos.residencia_pais_nombre == 'Argentina']
casos_argentina = casos_argentina.drop(columns='residencia_pais_nombre')
```

Código:

```
 df_join = pd.merge(
    casos_argentina,
    provincias,
    left_on='residencia_provincia_nombre',
    right_on=['iso_nombre']
)
```

PySpark

Se hace uso de la funcionalidad ‘join’.

Parámetros utilizados:

df_1.join(df_2, on=column_name, how='inner')

Requisitos previos:

```
# Dataset con coordenadas de las provincias

provincias_temp = spark.read.option("header",True) \

        .option("sep", ",") \

        .option("inferSchema", "true") \

        .csv(archivo_provincias)

provincias_temp = provincias_temp.withColumn('provincia', F.struct("centroide_lat", "centroide_lon"))

provincias_temp = provincias_temp.withColumn('provincia', provincias_temp.provincia.cast("string"))

provincias = provincias_temp.select('provincia', 'iso_nombre')

provincias = provincias.withColumn('iso_nombre', \

                F.when(provincias['iso_nombre'] == 'Ciudad Autónoma de Buenos Aires', 'CABA') \

                .otherwise(provincias['iso_nombre']))

# Dataset casos covid Argentina.

casos = spark_dict[key]

casos_argentina = casos.filter(casos['residencia_pais_nombre'] == 'Argentina')

casos_argentina = casos_argentina.drop('residencia_pais_nombre')
```

Código:

```
df_join = casos_argentina.join(
    provincias,
    casos_argentina.residencia_provincia_nombre == provincias.iso_nombre,
    how='inner'
)
```

Resultados

Tiempos por iteración


```
{
    "pandas": {
        "tiempos": {
            "1800mb": [24.487361431121826,24.874932527542114,24.388399839401245],
            "200mb": [2.5114572048187256, 2.364752769470215, 2.291337728500366],
            "600mb": [7.035664081573486, 7.2420737743377686, 7.181816339492798]
        }
    },
    "pyspark": {
        "tiempos": {
            "1800mb": [0.276069641113281, 0.30893301963806, 0.271385192871093],
            "200mb": [1.5281364917755127,0.4237527847290039,0.30540037155151367],
            "600mb": [0.3890213966369629, 0.2917914390563965, 0.2902817726135254]
        }
    }
```

}

Figura 27 - Promedio tiempo Joins

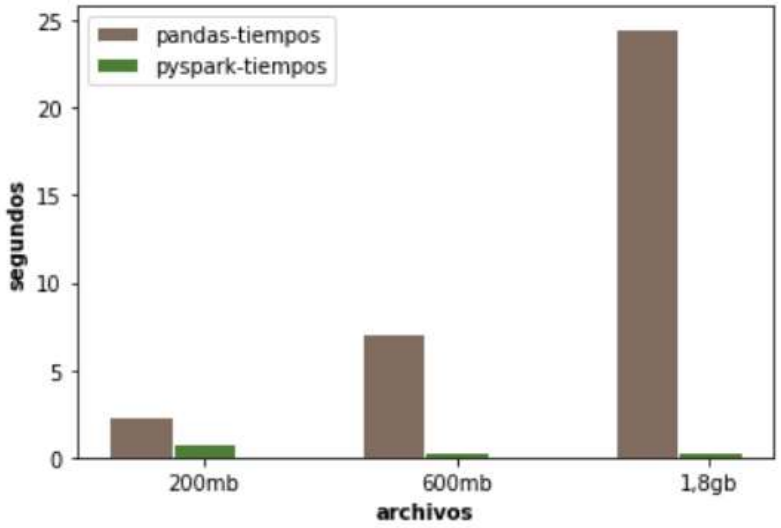


Figura 27: tiempo promedio para la unión de DataFrames por archivo para cada herramienta. Autoría propia.

3.1.9.2. GroupBy

Agrupación de datos para obtener información. En este caso se calcula cuantos casos se reportaron agrupados por sexo y edad.

Pandas

Se hace uso de la funcionalidad ‘groupby’.

Parámetros utilizados:

DataFrame.groupby([columns])

Código:

```
df_pd = pandas[key].groupby(['sexo', 'edad']).agg("count")
```

PySpark

Se hace uso de la funcionalidad ‘groupBy’.

Parámetros utilizados:

DataFrame.groupBy([columns])

Código:

```
df = spark_dict[key].groupBy(['sexo', 'edad']).count()
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [7.566366910934448, 6.339615821838379, 6.3550124168396],
      "200mb": [0.8940138816833496, 1.0097546577453613, 1.0623648166656494],
      "600mb": [2.2354416847229004, 2.1332554817199707, 2.080054759979248]
   },
   "pyspark": {
      "1800mb": [19.695650100708008, 20.076890468597412, 19.277278423309326],
      "200mb": [3.6845386028289795, 2.9755241870880127, 3.1389870643615723],
      "600mb": [7.151000499725342, 6.708099365234375, 6.602245569229126]
```

```
    }
}
```

Figura 28 - Promedio tiempo GroupBy

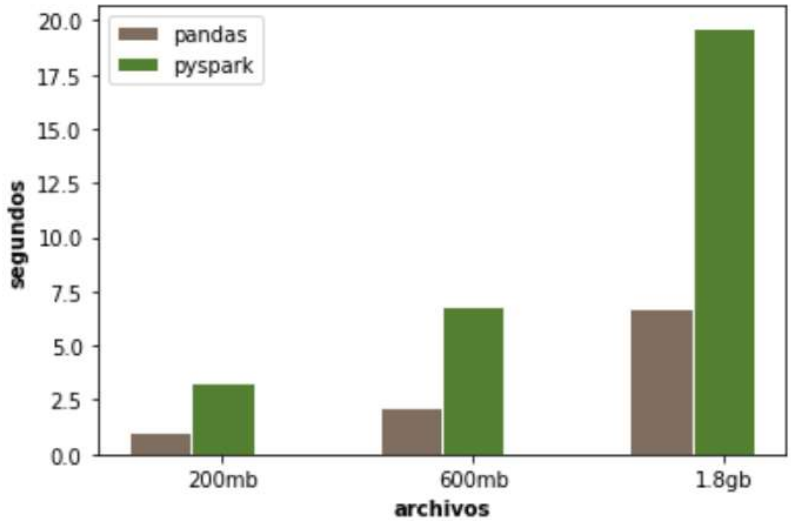


Figura 28: tiempo promedio para GroupBy por archivo para cada herramienta. Autoría propia.

3.1.9.3. Count

Contar apariciones de un valor en un DataFrame. En este caso se hace un conteo de casos de personas con sexo Femenino que sean mayores de 80 años.

Pandas

Se hace uso de la funcionalidad ‘count’.

Parámetros utilizados:

DataFrame[condition].count()

Código:

```
df_pd = pandas[key][(pandas[key].sexo == 'F') & (pandas[key].edad >= 80)].count()
```

PySpark

Se hace uso de la funcionalidad ‘count’.

Parámetros utilizados:

DataFrame.filter(condition).count()

Código:

```
df = spark_dict[key].filter((spark_dict[key].sexo == 'F') & (spark_dict[key].edad >=
80)).count()
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [0.611236572265625, 0.5967025756835938, 0.5944716930389404],
      "200mb": [0.14993691444396973, 0.09185981750488281,
0.09154415130615234],
      "600mb": [0.21300244331359863, 0.20921587944030762,
0.21022915840148926]
   },
   "pyspark": {
      "1800mb": [16.729555368423462, 16.680155992507935, 16.67951250076294],
      "200mb": [2.2228639125823975, 2.007298469543457, 1.9932050704956055],
      "600mb": [5.468553066253662, 5.396849155426025, 5.4417126178741455]
   }
```

}

Figura 29 - Promedio tiempo Count

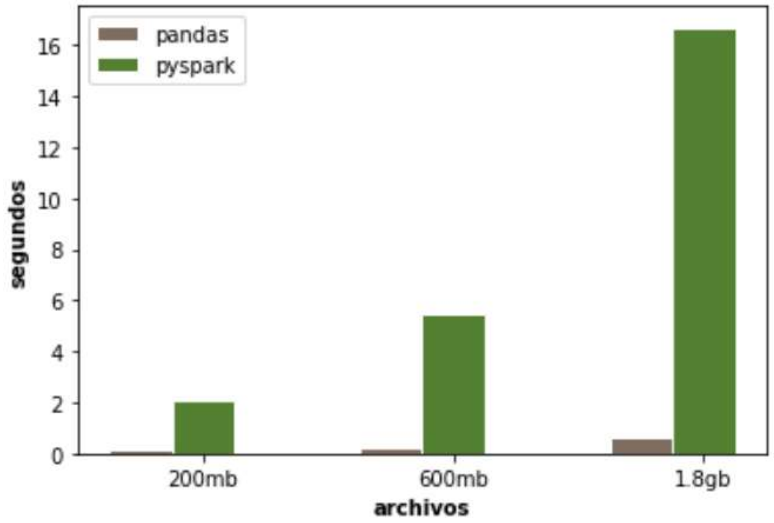


Figura 29: tiempo promedio para el Count por archivo para cada herramienta. Autoría propia.

3.1.9.4. Min

Calcular el mínimo valor de una columna. En este caso se calcula la mínima fecha reportada.

Pandas

Se hace uso de la funcionalidad ‘min’.

Parámetros utilizados:

DataFrame[column].min()

Código:

```
min_fecha_apertura = df['fecha_apertura'].min()
```

PySpark

Se hace uso de la funcionalidad ‘min’.

Parámetros utilizados:

```
DataFrame.select(min(column)).collect()
```

Código:

```
df = spark_dict[key].withColumn("fecha_apertura",
                        to_timestamp(col("fecha_apertura"),
                                "yyyy-MM-dd"))min_fecha_apertura =
df.select(min('fecha_apertura')).collect()
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [0.03207254409790039, 0.031485795974731445,
0.031978607177734375],
      "200mb": [0.003328561782836914, 0.0029718875885009766,
0.0029468536376953125],
      "600mb": [0.010400533676147461, 0.009948253631591797,
0.010161638259887695]
   },
   "pyspark": {
```

"1800mb": [26.768272399902344, 26.594033002853394, 27.248106956481934],
"200mb": [3.08001708984375, 3.231893301010132, 3.3722362518310547],
"600mb": [8.303431749343872, 8.328128099441528, 8.22463321685791]
}
}

Figura 30 - Promedio tiempo Min

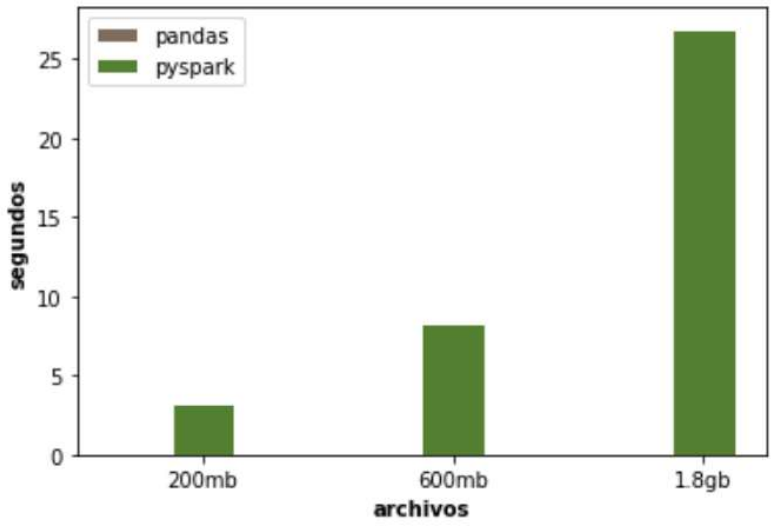


Figura 30: tiempo promedio para Min por archivo para cada herramienta. Autoría propia.

3.1.9.5. Max

Calcular el máximo valor por columna. En este caso se calcula la máxima fecha reportada.

Pandas

Se hace uso de la funcionalidad ‘max’.

Parámetros utilizados:

DataFrame[column].max()

Código:

```
max_fecha_apertura = df['fecha_apertura'].max()
```

PySpark

Se hace uso de la funcionalidad ‘max’.

Parámetros utilizados:

DataFrame.select(max(column)).collect()

Código:

```
df = spark_dict[key].withColumn("fecha_apertura",
                        to_timestamp(col("fecha_apertura"),
                                "yyyy-MM-dd"))
max_fecha_apertura = df.select(max('fecha_apertura')).collect()
```

Resultados

Tiempos por iteración


```
{
   "pandas": {
      "1800mb": [0.04286813735961914, 0.038877010345458984,
0.0392916202545166],
```

```
      "200mb": [0.0032324790954589844, 0.0028841495513916016,
0.0028679370880126953],
      "600mb": [0.01017618179321289, 0.00993967056274414,
0.009997844696044922]
   },
   "pyspark": {
      "1800mb": [25.758188247680664, 26.170207262039185, 26.811375856399536],
      "200mb": [3.486179828643799, 3.279587745666504, 3.354332447052002],
      "600mb": [8.533971071243286, 8.860496044158936, 8.362021923065186]
   }
}
```


Figura 31 - Promedio tiempo Max

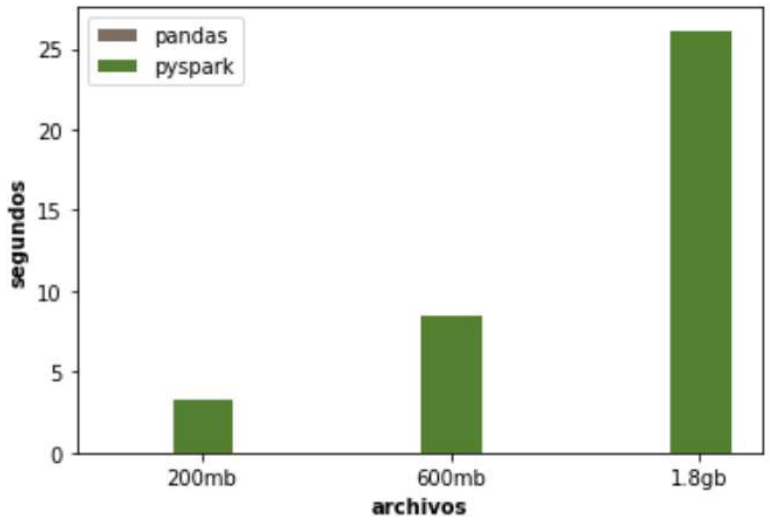


Figura 31: tiempo promedio para Max por archivo para cada herramienta. Autoría propia.

3.1.9.6. AVG

Calcula el promedio de los valores dado una columna en particular. En este caso se calcula el promedio de edad para los casos reportados.

Pandas

Se hace uso de la funcionalidad 'mean'.

Parámetros utilizados:

DataFrame.column.mean()

Código:

```
df_pd = pandas[key].edad.mean()
```

PySpark

Se hace uso de la funcionalidad 'avg'.

Parámetros utilizados:

DataFrame.groupBy().avg(column).collect()

Código:

```
df = spark_dict[key].groupBy().avg('edad').collect()
```

Resultados

Tiempos por iteración

{
  "pandas": {

```
    "1800mb": [0.027553081512451172, 0.027036428451538086,
0.0272519588470459],
    "200mb": [0.004982709884643555, 0.004792451858520508,
0.004315853118896484],
    "600mb": [0.008927583694458008, 0.008768320083618164,
0.008788585662841797]
  },
  "pyspark": {
    "1800mb": [16.992757320404053, 18.50647735595703, 17.6005277633667],
    "200mb": [2.1381494998931885, 2.069664478302002, 2.075735092163086],
    "600mb": [5.538867235183716, 5.4920079708099365,5.498332500457764]
  }
}
```


Figura 32 - Promedio tiempo AVG

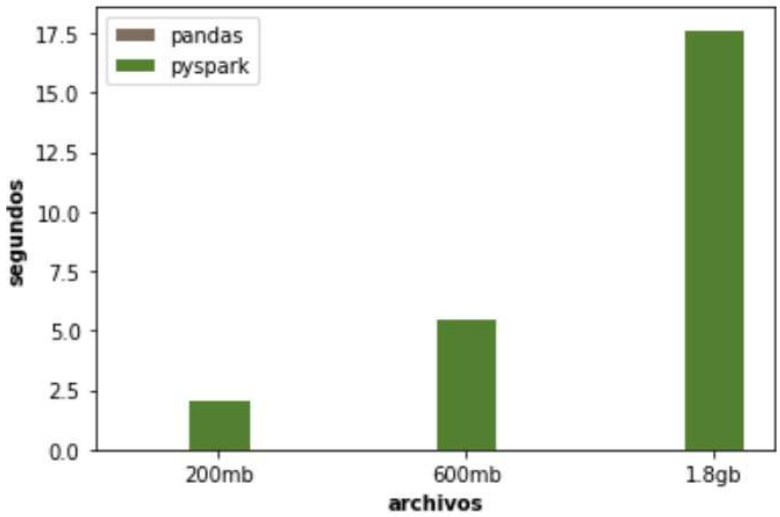


Figura 32: tiempo promedio para AVG por archivo para cada herramienta. Autoría propia.

3.1.10. Proceso integrador

Con el fin de realizar una prueba integradora, se desarrollan dos scripts, uno por herramienta, los cuales comprenden un procesamiento de datos que generar las siguientes métricas agrupadas por provincia y por fecha:

- Cantidad de casos en cuidados intensivos por sexo.
- Cantidad de casos en cuidados intensivos por origen de financiamiento.
- Cantidad de casos nuevos fallecidos por sexo.
- Cantidad de casos nuevos fallecidos por origen de financiamiento.
- Cantidad de casos nuevas internaciones por sexo.
- Cantidad de casos nuevas internaciones por origen de financiamiento.
- Cantidad de casos por clasificación y sexo.
- Promedio edad de casos en cuidados intensivos.
- Promedio edad de casos nuevos fallecidos.
- Promedio edad de casos nuevas internaciones.
- Promedio edad por clasificación.

Además, con el apoyo de un DataFrame que contiene las provincias de Argentina y sus coordenadas, se enriqueció el DataFrame final para poder ubicar las métricas en un mapa.

De esta manera, el DataFrame resultante permite visualizar por fecha las métricas antes mencionadas ubicadas en la provincia a la que corresponden con la opción de visualizarlas en un gráfico de tipo mapa.

Pandas Python script

pandas_script.py

PySpark Python script

pyspark_script.py

Resultados


```
{
   "pandas": {
      "cpu": {
         "1800mb": [97.3, 90.0, 90.5],
         "200mb": [94.1, 100.0, 99.9],
         "600mb": [99.7, 100.0, 99.8]
      },
      "memoria": {
         "1800mb": [4181356544, 4185292800, 4184739840],
         "200mb": [3786842112, 3787161600, 3787350016],
         "600mb": [3801456640, 3802783744, 3803095040]
      },
      "tiempos": {
         "1800mb": [49.78736853599548, 54.94438600540161, 52.66531229019165],
         "200mb": [6.641046524047852, 6.052762031555176, 6.8448967933654785],
         "600mb": [15.744794368743896, 15.658109426498413, 14.82665205001831]
      }
   },
   "pyspark": {
      "cpu": {
```

```
        "1800mb": [110.1, 109.6, 109.5],
        "200mb": [165.1, 125.8, 122.4],
        "600mb": [115.3, 112.8, 112.0]
      },
      "memoria": {
        "1800mb": [1716740096, 1761599488, 1666371584],
        "200mb": [1382391808, 1565278208, 1474015232],
        "600mb": [1585565696, 1578606592, 1593544704]
      },
      "tiempos": {
        "1800mb": [520.9318192005157, 525.9829268455505, 540.0702137947083],
        "200mb": [141.91835808753967, 129.51602411270142, 121.70634126663208],
        "600mb": [201.95885372161865, 202.88529872894287, 202.81581020355225]
      }
    }
}
```


Figura 33 - Promedio uso memoria caso integrador

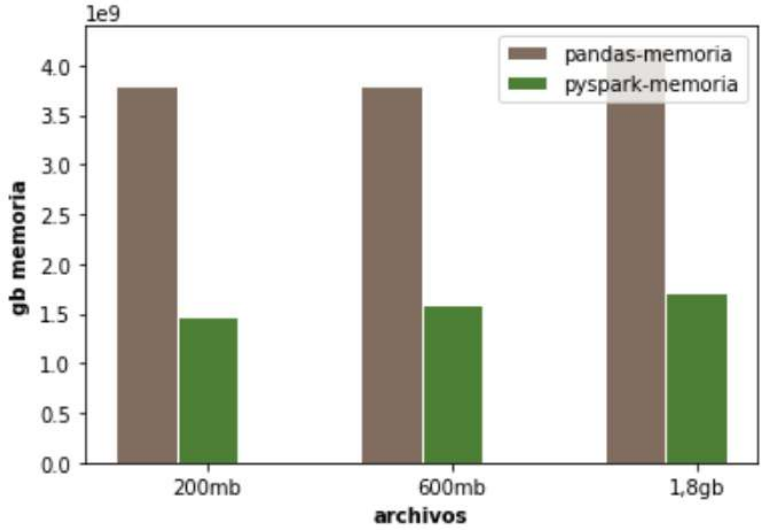


Figura 33: promedio de la memoria utilizada para los casos integradores por archivo para cada herramienta. Autoría propia.

Figura 34 - Promedio uso CPU caso integrador

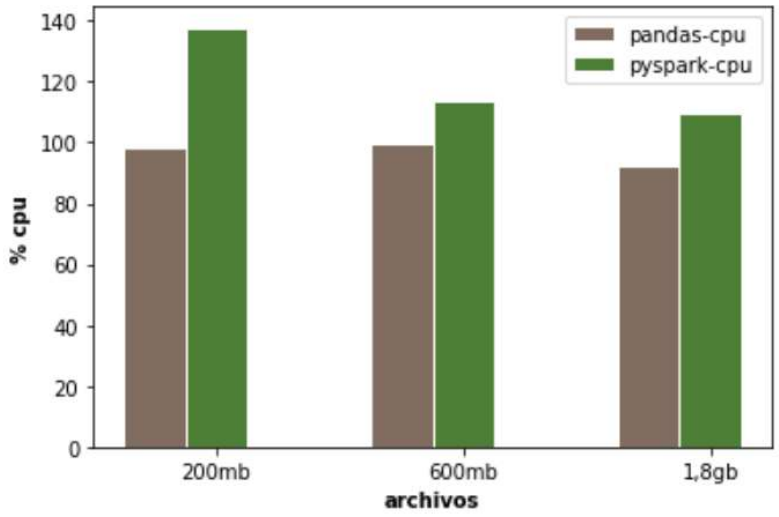


Figura 34: promedio del CPU utilizado para los casos integradores por archivo para cada herramienta. Autoría propia.

Figura 35 - Promedio tiempos caso integrador

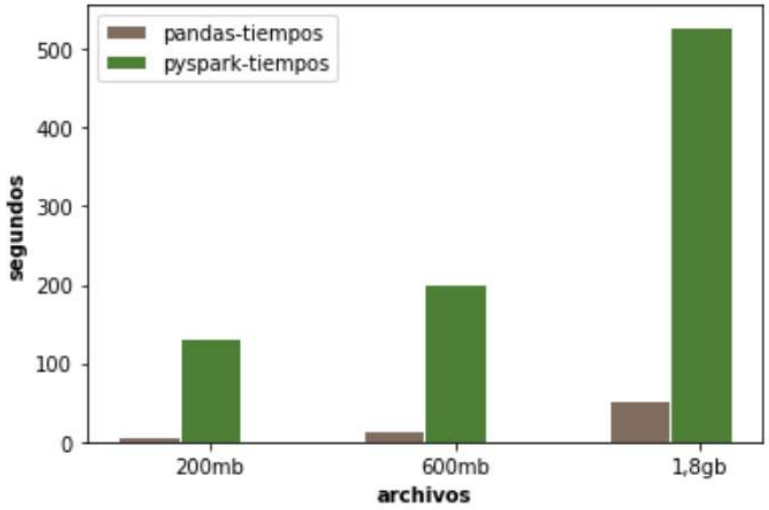


Figura 35: tiempo promedio para los casos integradores por archivo para cada herramienta. Autoría propia.

Resultados de las métricas de MyFEPS

Tabla 12 - Valoración memoria Pandas

Archivo	Memoria utilizada (bytes)	Formula (%)	Resultado
200mb	3787117909.333	1 - (3787117909.333/1666371584)	-1.273
600mb	3802445141.333	1 - (3802445141.333/1666371584)	-1.282
1800mb	4183796394.667	1 - (4183796394.667/1666371584)	-1.511

Tabla 12: muestras de la memoria utilizada por cada proceso, la fórmula para calcular la valoración del uso de memoria y el resultado del mismo. Autoría propia.

Tabla 13 - Valoración memoria PySpark

Archivo	Memoria utilizada (bytes)	Formula (%)	Resultado
200mb	1473895082.667	1 - (1473895082.667/1666371584)	0.116
600mb	1585905664.000	1 - (1585905664.000/1666371584)	0.048
1800mb	1714903722.667	1 - (1714903722.667/1666371584)	0.029

Tabla 13: muestras de la memoria utilizada por cada proceso, la fórmula para calcular la valoración del uso de memoria y el resultado del mismo. Autoría propia.

Tabla 14 - Valoración CPU Pandas

Archivo	CPU utilizado (%)	Formula (%)	Resultado
200mb	98.000	1 - (98.0/100)	0.020
600mb	99.833	1 - (99.833/100)	0.002
1800mb	92.600	1 - (92.600/100)	0.074

Tabla 14: muestras del CPU utilizado por cada proceso, la fórmula para calcular la valoración del uso de CPU y el resultado del mismo. Autoría propia.

Tabla 15 - Valoración CPU PySpark

Archivo	CPU utilizado (%)	Formula (%)	Resultado
200mb	137.767	1 - (137.767/100)	-0.378
600mb	113.367	1 - (113.367/100)	-0.134
1800mb	109.733	1 - (109.733/100)	-0.097

Tabla 15: muestras del CPU utilizado por cada proceso, la fórmula para calcular la valoración del uso de CPU y el resultado del mismo. Autoría propia.

4. CONCLUSIONES

4.1. Discusión – Resumen final

Las herramientas de análisis de datos son esenciales para nuestra era, la Era de Los Datos. Son necesarias para lograr encontrar respuestas en los datos que se recolectan segundo a segundo, y así según sea el caso poder potenciar una sociedad o mitigar problemas.

¿Por qué ni Spark ni Pandas son mejores que el otro? Mejor dicho, hay que elegir siempre la herramienta adecuada para el trabajo en cuestión.

- Evaluación del uso de Pandas por sobre Spark:

Pandas es fácil de usar y posee mucha información valiosa en internet gracias a su numerosa comunidad. Realiza todas sus operaciones de forma rápida siempre y cuando el volumen de datos sea reducido. Al ser parte del ecosistema de Python goza de muchas librerías numéricas, estadísticas y de aprendizaje automático. Además, ofrece simplicidad a la hora de instalarlo y flexibilidad en cuanto a los proyectos a los que se puede adaptar.

A su vez presenta limitantes a la hora de escalar, ya que no escala en absoluto. No hace uso de varios CPU y necesita que todo el volumen de datos quepa en la totalidad de memoria RAM de la maquina donde se ejecuta. Por el lado del desarrollo de código, presenta debilidades a la hora de generar código robusto ya que pertenece a un lenguaje tipado dinámicamente.

- Conclusiones sobre Pandas:

Pandas puede ser utilizado en proyectos de análisis de datos donde sea requerido realizar de forma rápida y dinámica la exploración y experimentos de datos, con disponibilidad de mucha memoria RAM para poder procesar grandes volúmenes de datos. No se debería implementar desarrollos escritos en Pandas en ambientes de

producción ya que por ser de tipado dinámico hay errores que no hay forma de detectarlos hasta que pasen.

- Evaluación del uso de Spark por sobre Pandas:

Spark desde su concepción escala con la cantidad de CPU, la cantidad de máquinas y/o en función de la cantidad de datos disponibles. Aunque requiere tener un grado de conocimiento mayor sobre las dependencias y versiones necesarias para poder instalarlo, los tiempos que esto conlleva son muy pequeños. No presenta limitantes en cuanto a la cantidad de datos a procesar con una cantidad limitada de recursos, exceptuando el tiempo que toma. Es posible desarrollar código Spark tanto con Scala como con Python, ofreciendo así un espectro más amplio de soluciones y librerías presentes en cada uno de ellos.

Como fue creado para procesar grandes cantidades de datos sin limitantes, en ciertos casos sigue siendo más lento en conjuntos de datos pequeños, en los que Pandas los procesa en cuestión de segundos. Aunque el volumen de su comunidad es similar a la de Pandas presenta menos integraciones y soluciones con otras librerías como por ejemplo las de aprendizaje automático.

- Conclusiones sobre Spark:

Spark es ideal para cargas de trabajo ETL (Extract, transform, Load) donde se debe definir un pipeline de transformación sobre un conjunto grande de datos en entornos productivos. En análisis de datos dinámicos donde se ejecutan celda por celda, Spark presenta mayores tiempos que en el caso de Pandas, pero cuando Spark tiene la opción de evaluar todas las funciones y pasos que debe ejecutar (caso integrador) optimiza el procesamiento reduciendo los tiempos, consumo de memoria RAM y CPU. En el caso de implementaciones para proyectos de aprendizaje automático se debe tener en cuenta como la segunda opción, por debajo de Pandas.

- Conclusiones final

No hay que reemplazar una herramienta con la otra, entre si se complementan y ambas presentan sus pros y sus contras. Como se comentó anteriormente, es necesario conocerlas a ambas para saber cuándo elegir cada una según el caso de uso.

Ambas herramientas manejan métricas parecidas en casi la totalidad de ejes de comparación definidos, con una gran diferencia en los modos de ejecución y los tiempos que toman en cada caso.

Por ende, en base a los resultados se definen los ambientes o casos en los que se pueden aplicar cada herramienta:

Para desarrollos relacionados a aprendizaje automático, análisis de datos y generación de métricas sobre un conjunto reducido o un sub conjunto de datos de manera ágil es recomendable hacer uso de Pandas.

Para procesos de ingeniería en ambientes productivos donde es necesario consumir grandes volúmenes de datos que pueden fluctuar requiriendo una posible escalabilidad, transformarlos y guardarlos es recomendable hacer uso de Spark.

Ambos se pueden usar en conjunto en todos los proyectos, donde primero se puede hacer uso de Pandas para un análisis ágil y certero sobre un sub conjunto de datos, y luego aplicar Spark para desarrollar una solución robusta y escalable que se implemente con el conjunto total de datos en un ambiente distribuido y productivo.

4.2. Futuras líneas de investigación

Este trabajo final puede ser continuado en varias líneas de investigación:

1. Incorporar el proyecto Koalas como alternativa de Pandas, para desarrollar procesamientos de datos con sintaxis de Pandas, pero ejecutados dentro de una sesión de Spark (distribuidos).
2. Incorporar el proyecto Dask como alternativa de Pandas, siendo una solución completa para el desarrollo de procesamientos de datos con sintaxis Pandas los cuales pueden ser ejecutados de forma distribuida, y ser orquestados como servicios.

5. BIBLIOGRAFÍA

ISAACSON W. (2014). Los Innovadores: Los genios que inventaron el futuro. Debate

WARREN J., Marz N. (2015). Big Data: Principles and best practices of scalable realtime data systems. Hanning

ZAHARIA, M., Chambers, B. (2018). Spark: The Definitive Guide. O'Reilly Media, Inc.

HEYDT M. (2015). Learning pandas. 2a. ed. Packt.

OZSVALD, I., Gorelick, M. (2014). High Performance Python. O'Reilly Media, Inc.

MCKINNEY, W. (2012). Python for Data Analysis. O'Reilly Media.

GITHUB PANDAS [en línea]. 2021. [consulta 20 de enero de 2021].
<https://github.com/pandas-dev/pandas>

ECOSISTEMA PANDAS [en línea]. 2021. [consulta 20 de enero de 2021].
<https://pandas.pydata.org/docs/ecosystem.html>

GITHUB SPARK [en línea]. 2021. [consulta 21 de enero de 2021].
<https://github.com/apache/spark>

ECOSISTEMA SPARK [en línea]. 2021. [consulta 21 de enero de 2021].
<https://www.edureka.co/blog/apache-spark-ecosystem/>

IDC. IDC's Global DataSphere Forecast Show Continued Steady Growth in the Creation and Consumption of Data [en línea]. 2020. [consulta 10 de septiembre de 2020].
<https://www.idc.com/getdoc.jsp?containerId=prUS46286020>

ONU. THE 17 GOALS. [en línea]. 2020. [consulta 15 de septiembre de 2020].
<https://sdgs.un.org/goals>

JUPYTER ORG. Some information about the Jupyter Project and Community. [en línea]. 2020. [consulta 03 de octubre de 2020].
<https://jupyter.org/about>

BOARD INFINITY. Different Types of Big Data Architecture Layers & Technology Stacks. [en línea]. 2020. [consulta 01 de septiembre de 2020].
<https://medium.com/boardinfinity/different-types-of-big-data-architecture-layers-technology-stacks-dd779bde392d>

BIGDATAFRAMEWORK. Data Types: Structured vs. Unustructured Data. [en línea]. 2019. [consulta 01 de septiembre de 2020].
<https://www.bigdataframework.org/data-types-structured-vs-unstructured-data/>

MINISTERIO DE SALUD. COVID-19, Casos registrados en la República Argentina. [en línea]. 2020. [consulta 25 de agosto de 2020].
<http://datos.salud.gob.ar/dataset/covid-19-casos-registrados-en-la-republica-argentina>

GURU99. Difference between Information and Data. [en linea]. 2020. [consulta 05 de septiembre de 2020]. <https://www.guru99.com/difference-information-data.html>

GURU99. What is Data Analysis? Types, Process, Method, Techniques. [en línea]. 2020. [consulta 05 de octubre de 2020].
<https://www.guru99.com/what-is-data-analysis.html>

PYPL. Popularity of Programming Language. [en línea]. 2020. [consulta 27 de agosto de 2020].
<http://pypl.github.io/PYPL.html>

JET BRAINS. The State of Developer Ecosystem 2020: Python. [en línea]. 2020. [consulta 04 de septiembre de 2020].
<https://www.jetbrains.com/lp/devecosystem-2020/>

DATABRICKS. DataFrame. [en línea]. 2020. [consulta 15 de septiembre de 2020].
<https://databricks.com/glossary/what-are-dataframes#:~:text=Back%20to%20glossary%20A%20DataFrame,a%20spreadsheet%20with%20named%20columns>

AIDA NGOM. [en línea]. 2020. [consulta 05 de octubre de 2020].
<https://www.adaltas.com/en/2020/07/23/benchmark-study-of-different-file-format/>

LUMINOUSMEN. Big Data file formats. [en línea]. 2020. [consulta 04 de octubre de 2020].
<https://luminousmen.com/post/big-data-file-formats>

6. ANEXOS

6.1. Anexo I: Glosario de Acrónimos y Términos

Analytics (Análisis de datos). Enfoque que implica el análisis de datos (Big Data en particular) para sacar conclusiones.

API. Application Programming Interface. Hace referencia a los procesos, las funciones y los métodos que brinda una determinada biblioteca de programación a modo de capa de abstracción para que sea empleada por otro programa informático.

Aplicación. Tipo de software que permite al usuario realizar uno o más tipos de trabajos.

Backend. Es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione.

Benchmarks. punto de referencia para realizar comparaciones.

Cassandra. Es una base de datos distribuida NoSQL de código abierto. La escalabilidad lineal y la tolerancia a fallas probada en hardware básico o infraestructura en la nube la convierten en la plataforma perfecta para datos de misión crítica.

Clusters. Conjunto de computadoras interconectadas por medio de una red de alta velocidad, las cuales operan como si fueran una única computadora.

Código. Conjunto de líneas de texto con los pasos que debe de seguir la computadora para ejecutar un programa.

CPU. Central Processing Unit. Es la parte de una computadora en la que se encuentran los elementos que sirven para procesar datos.

Framework (Entorno de trabajo). Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Hadoop MapReduce. Es un marco de software para escribir fácilmente aplicaciones que procesan grandes cantidades de datos (conjuntos de datos de varios terabytes) en paralelo en grandes grupos (miles de nodos) de hardware básico de una manera confiable y tolerante a fallas.

Hardware. Parte física de una computadora o sistema informático.

IDE. Integrated Development Environment. Es una aplicación de software que proporciona a los programadores informáticos instalaciones completas para el desarrollo de software. Normalmente consta de al menos un editor de código fuente, herramientas de automatización de compilación y un depurador.

Import. Invocar funcionalidades de librerías externas dentro del código que se está desarrollando.

JAR. Java Archive**.** Es un formato de archivo de paquete que se usa normalmente para agregar muchos archivos de clase Java y metadatos y recursos asociados en un archivo para su distribución.

Java. Es un lenguaje de programación y la primera plataforma informática creada por Sun Microsystems en 1995. Es la tecnología subyacente que permite el uso de programas punteros, como herramientas, juegos y aplicaciones de negocios. Java se

ejecuta en más de 850 millones de computadoras personales de todo el mundo y en miles de millones de dispositivos, como dispositivos móviles y aparatos de televisión.

JSON. JavaScript Object Notation. Es un formato ligero de intercambio de datos. Es fácil para los humanos leer y escribir. Es fácil para las máquinas analizar y generar. Se basa en un subconjunto del estándar de lenguaje de programación JavaScript ECMA-262 3.ª edición - diciembre de 1999. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluido C, C ++, C #, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen de JSON un lenguaje de intercambio de datos ideal.

Kafka. Es una plataforma de transmisión de eventos distribuida de código abierto utilizada por miles de empresas para canalizaciones de datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica.

Kanban. Es un método para administrar y mejorar el trabajo en los sistemas humanos. Este enfoque tiene como objetivo administrar el trabajo equilibrando las demandas con la capacidad disponible y mejorando el manejo de los cuellos de botella a nivel del sistema.

KPI. Key Performance Indicator. Son métricas que indican el nivel de desempeño en base a los objetivos que se fijaron.

Lenguaje de programación. Lenguaje formal (un lenguaje con reglas gramaticales bien definidas) que le proporciona al programador la capacidad de escribir o programar una serie de instrucciones o secuencias de ordenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de una computadora.

Log. Usado para referirse a la grabación secuencial en un archivo de todos los acontecimientos que afectan a un proceso particular.

Paralelismo. Función que realiza el procesados para ejecutar varias tareas al mismo tiempo en distintos procesos.

NoSQL. Una base de datos NoSQL proporciona un mecanismo para el almacenamiento y recuperación de datos que se modela en medios distintos de las relaciones tabulares utilizadas en las bases de datos relacionales.

Open Source. Es un software de computadora que se publica bajo una licencia en la que el titular de los derechos de autor otorga a los usuarios los derechos para usar, estudiar, cambiar y distribuir el software y su código fuente a cualquier persona y para cualquier propósito.

Procesador. Componente electrónico donde se realizan los procesos lógicos.

Queries (consulta a una base de datos)**.** Es una solicitud de datos o información de una tabla de base de datos o una combinación de tablas. Estos datos pueden generarse como resultados devueltos por Structured Query Language (SQL) o como imágenes, gráficos o resultados complejos.

Robusto. Es la capacidad de un sistema informático para hacer frente a los errores durante la ejecución y hacer frente a la entrada errónea.

Scala. Es un lenguaje de programación de propósito general de tipo estático que admite tanto la programación orientada a objetos como la programación funcional.

Scrum. Es un marco que utiliza una mentalidad ágil para desarrollar, entregar y mantener productos en un entorno complejo, con un énfasis inicial en el desarrollo de software, aunque también se utiliza en otros campos que incluyen investigación, ventas, marketing y tecnologías avanzadas.

Scripts. En informática un guion, archivo de órdenes o archivo de procesamiento por lotes, vulgarmente referidos con el barbarismo script, es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano. Los guiones son casi siempre interpretados, pero no todo programa interpretado es considerado un guion. El uso habitual de los guiones es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. Por este uso es frecuente que los shells sean a la vez intérpretes de este tipo de programas.

Sistema Operativo. Es un programa o conjunto de programas que en un sistema informático gestiona los recursos de hardware y provee servicios a los programas de aplicación, y corre en modo privilegiado respecto de los restantes.

Software. Se conoce como software al equipamiento lógico o soporte lógico de un sistema informático; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware.

Stack. Hace referencia a un conjunto de herramientas.

Streaming. Es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se descarga.

Testing (Pruebas de Software). Investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto.

Trade off (Solución de Compromiso). Decisión tomada en una situación conflictiva en la cual se debe perder, reducir cierta cualidad a cambio de otra cualidad.

Unit Test. Una prueba unitaria es una forma de probar una unidad: el fragmento de código más pequeño que se puede aislar lógicamente en un sistema.

6.2. Anexo II: MyFEPS

Tabla 16 - Descripción característica y sub-características - Eficiencia

Característica Básica (CB)	**Sub-característica (Sub-CB)**	***En qué medida***	***Evaluable en términos de***	***Comparable con***	**CONCATENADO**
Eficiencia	en los tiempos de respuesta	realizas sus funciones usando tiempos iguales o menores a los pre-establecidos	la medición de los tiempos de respuesta	tiempos de respuesta pre-establecidos	En qué medida realizas sus funciones usando tiempos iguales o menores a los pre-establecidos. Evaluable en términos de la medición de los tiempos de respuesta. Comparable con tiempos de respuesta pre-establecidos
	en la utilización de memoria interna	realizas sus funciones usando cantidades de memoria interna iguales o menores a los pre-establecidos	la medición del uso de la memoria interna usada	memoria interna pre-establecidos	En qué medida realizas sus funciones usando cantidades de memoria interna iguales o menores a los pre-establecidos. Evaluable en términos de la medición del uso de la memoria interna usada. Comparable con memoria interna pre-establecidos
	en la utilización de almacenaje externo	realizas sus funciones usando almacenaje externo en cantidades iguales o menores a los pre-establecidos	la medición del uso de almacenaje externo	almacenaje externo pre-establecidos	En qué medida realizas sus funciones usando almacenaje externo en cantidades iguales o menores a los pre-establecidos. Evaluable en términos de la medición del uso de almacenaje externo. Comparable con almacenaje externo pre-establecidos

	en la utilización del CPU	realizas sus funciones usando el CPU en cantidades iguales o menores a los pre-establecidos	la medición del uso del CPU	uso pre-establecido del CPU	En qué medida realizas sus funciones usando el CPU en cantidades iguales o menores a los pre-establecidos. Evaluable en términos de la medición del uso del CPU. Comparable con uso pre-establecido del CPU

Tabla 16: describe que se evalúa para cada sub-característica dentro de la característica Eficiencia. [MYFEPS, 2011]

Tabla 17 - Descripción características y sub-características - Instalación

Característica Básica (CB)	**Sub-característica (Sub-CB)**	***En qué medida***	***en el contexto de***	***de modo que***	***Evaluable en términos de***	***Comparable con***	**CONCATENADO**
Instalación	Primera instalación	se instala	la primera instalación	no son necesarios recursos adicionales a los pre-establecidos.	los recursos necesarios para la primera instalación	recursos pre-establecidos	En qué medida se instala en el contexto de la primera instalación de modo que no son necesarios recursos adicionales a los pre-establecidos. Evaluable en términos de los recursos necesarios para la primera instalación. Comparable con recursos pre-establecidos
	Upgrades	se re-instala	la instalación de nuevas versiones	no son necesarios recursos adicionales a los pre-establecidos.	los recursos necesarios para la instalación de nuevas versiones	recursos pre-establecidos	En qué medida se re-instala en el contexto de la instalación de nuevas versiones de modo que no son necesarios recursos adicionales a los pre-establecidos.

	Evaluable en términos de los recursos necesarios para la instalación de nuevas versiones. Comparable con recursos pre-establecidos

Tabla 17: describe que se evalúa para cada sub-característica dentro de la característica Eficiencia. [MYFEPS, 2011]

Tabla 18 - Descripción de Atributos y Métricas – Eficiencia

Atributo	**Métrica**
Tiempo de Respuesta de la Función	1. Obtener el MOEF (Minutos de operación Estándar) para la función tipo x 2. Obtener un conjunto "N" (Tal q N >= 3) de Operadores para las Mediciones 3. Para SumDTP=0 y i=1 Hasta "N", de a Uno hacer a. Medir los minutos hombre que llevó la operación de Muestra MHO b. SumDTP+=MHOi 4. Media de Tiempos MTO=SumDTP/N 5. Valoración = Si (MTO <= MOEF) entonces 1 sino MOEF/MTO
Memoria Interna Utilizada para ejecutar la Función Tipo X	Con una Aplicación Adhoc, Para todas las Funciones 1. Medir CMF(x) cantidad de Memoria usada por la Función en Carga Alta. 2. Obtener una Cantidad de Memoria del sistema: CMS. 3. Valoración = 1- (CMF/CMS) Valor Final= Media de Valoración Parcial.
CPU Utilizada para ejecutar la Función Tipo X	Con una Aplicación Adhoc, Para todas las Funciones 1. Medir Carga(x) Porcentaje de CPU usada por la Función en Carga Alta. 2. Valoración = 1- (Carga/100) Valor Final= Media de Valoración Parcial.

Tabla 18: describe por atributo a evaluar los pasos a seguir para obtener los valores de las métricas en el caso de la característica Eficiencia. [MYFEPS, 2011]

Tabla 19 - Descripción de Atributos y Métricas – Instalación

Atributo	**Métrica**

Esfuerzo en horas-hombre necesarios para su primer instalación en el entorno de uso	1. Establecer el óptimo número de horas hombre para la primera instalación en entorno de uso: ONHHPI 2. Medir el óptimo número de horas hombre para la primera instalación en entorno de uso: NHHPI M = ((NHHPI - ONHHPI) / ONHHPI) - 1
Esfuerzo en horas-hombre necesarios para la instalación de sus versiones en el entorno de uso	1. Establecer el óptimo número de horas hombre para la primera instalación en entorno de uso: ONHHPI 2. Medir el óptimo número de horas hombre para la primera instalación en entorno de uso: NHHPI M = ((NHHPI - ONHHPI) / ONHHPI) - 1

Tabla 19: describe por atributo a evaluar los pasos a seguir para obtener los valores de las métricas en el caso de la característica Instalación. [MYFEPS, 2011]

6.3. Anexo III: Guía de instalación y configuración del entorno de trabajo

Hardware utilizado para el desarrollo del presente trabajo final de carrera:

- Notebook o PC de escritorio
- 16gb RAM
- Procesador i7-4710HQ CPU 2.50GHz
- Sistema Operativo Ubuntu 20.04
- Conexión a internet

Configuración de entorno de trabajo:

1. Instalar Python versión 3.7

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt update
$ sudo apt install python3.7
$ python –version
$ sudo apt install python3-pip
$ pip3 -version
```

2. Instalar virtualenv 20.0.3

```
$ pip3 install 'virtualenv==20.0.3'
$ virtualenv --version
```

3. Crear entorno virtual con python3.7

```
$ virtualenv –python=/usr/bin/python3.7 <nombre a elección>
```

4. Instalar Apache Spark 2.4.7

```
$ wget https://apache.dattatec.com/spark/spark-2.4.7/spark-2.4.7-bin-hadoop2.7.tgz
$ tar -xvzf spark-2.4.7-bin-hadoop2.7.tgz
$ sudo mv spark-2.4.7-bin-hadoop2.7 /usr/local/spark-2.4.7
```

5. Cómo Spark está desarrollado en Java y utiliza la JVM para correr las aplicaciones se debe instalar JAVA:

```
$ sudo apt install default-jre
$ sudo apt install scala
$ pip3 install py4j==0.10.7
```

6. Instalar PySpark:

```
$ pip3 install pyspark
$ pip3 install findspark
```

7. Instalar Pandas versión 1.1.3

```
$ pip3 install 'pandas==1.1.3'
$ pandas --version
```

8. Instalar Jupyter Notebook

```
$ pip3 install jupyter
$ jupyter notebook --version
```

9. Configurar variables de entorno

```
$ source <nombre_mi_entorno_virtual>/bin/activate
$ # Variables de entorno
$ export SPARK_HOME='/usr/local/spark-2.4.7'
$ export PATH=$SPARK_HOME:$PATH
$ export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
$ export PYSPARK_DRIVER_PYTHON="jupyter"
$ export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
$ export PYTHONPATH=$SPARK_HOME/python/lib/py4j-0.10.7-
src.zip:$PYTHONPATH
$ export PYSPARK_PYTHON=python3.7
```

10. Iniciar nuevo servidor de Jupyter Notebook

```
$ jupyter notebook
```

Figura 36 - Servidor Jupyter Notebook corriendo en localhost

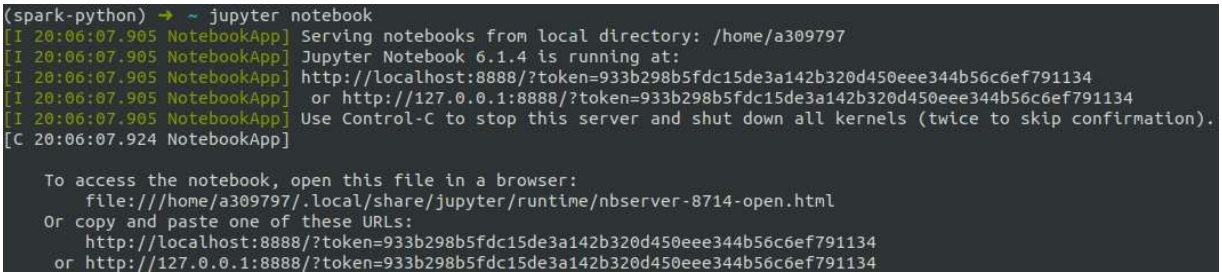


Figura 37: servicio Jupyter Notebook ejecutando en maquina local sobre localhost. Autoría propia.

11. Abrir interfaz de Usuario

Figura 37 - Interfaz de Usuario Jupyter Notebook.

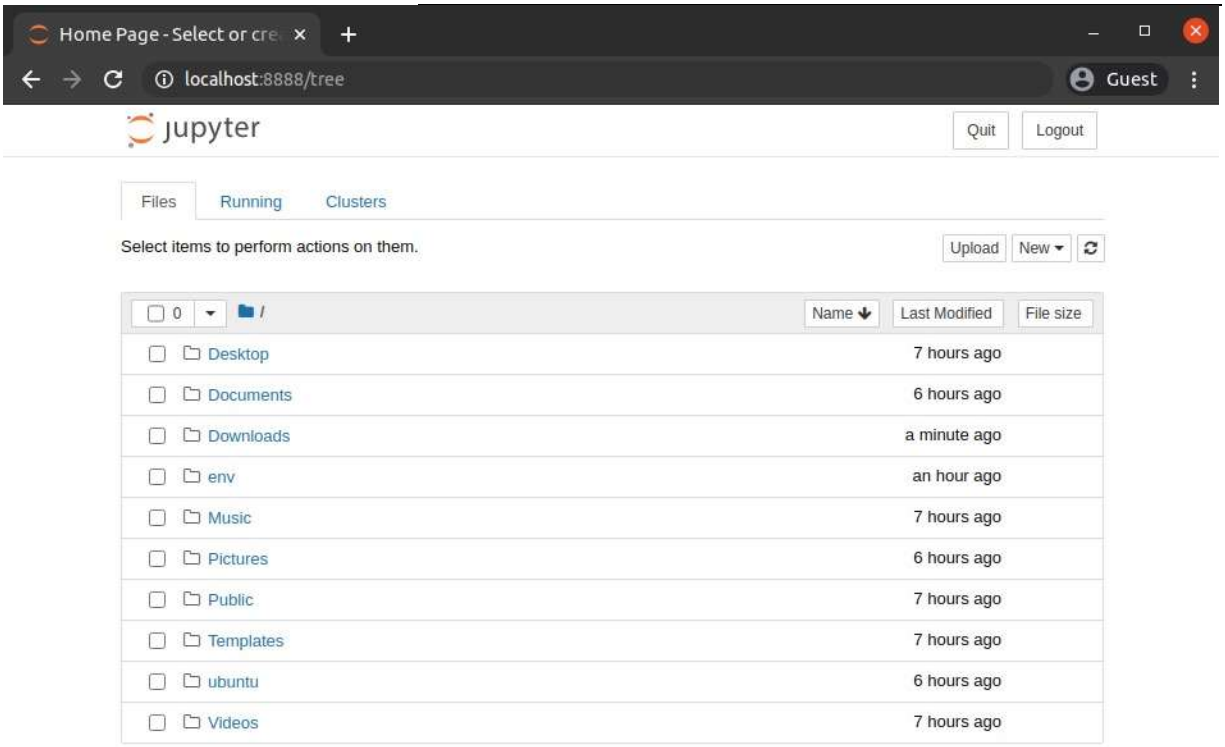


Figura 38: interfaz de usuario de Jupyter Notebook sobre explorador web. Autoría propia.

12. Crear nuevo notebook con kernel Python3.7

Figura 38 - Crear nuevo notebook con kernel python 3.7

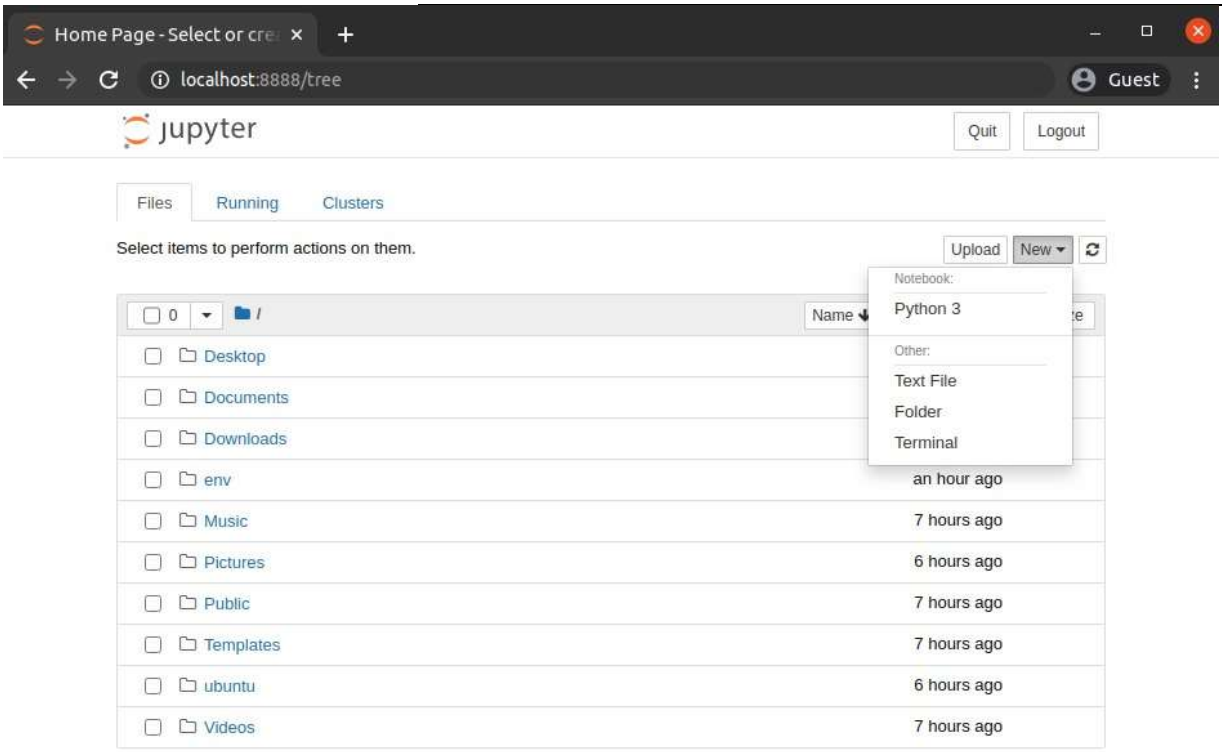


Figura 39: cómo crear un nuevo notebook seleccionando kernel Python 3.7. Autoría propia.

13. Importar librerías

Figura 39 - Importar librerías.

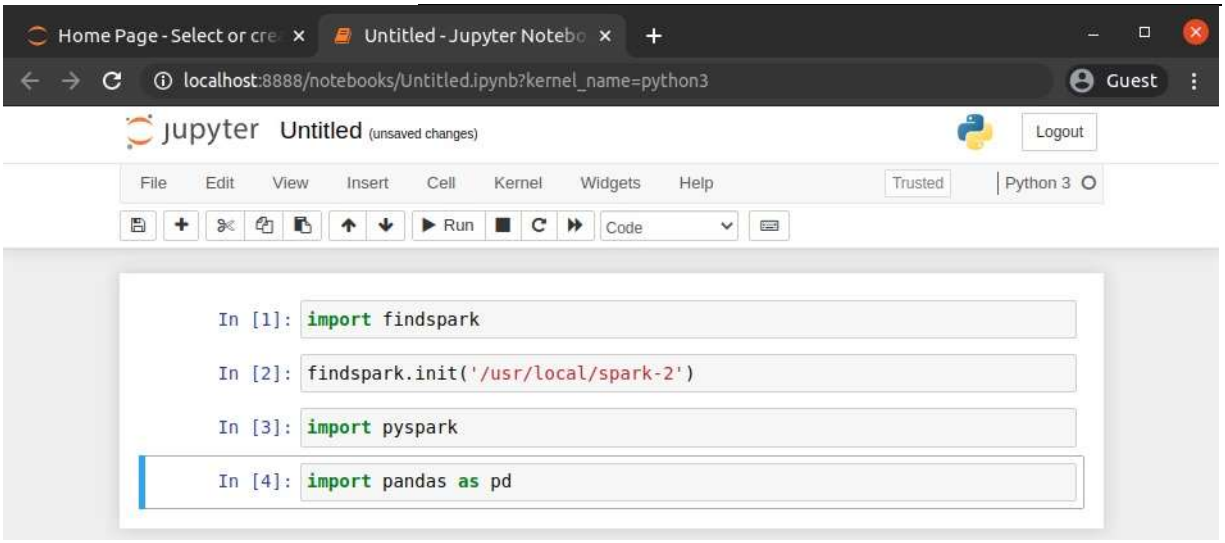


Figura 40: notebook con kernel python 3.7 y como se importan las librerías PySpark y Pandas. Permite comprobar que todo se ha realizado correctamente ya que en el caso de no ser así presentaría errores. Autoría propia.

14. Descargar DataFrame casos covid19 link.

6.4. Anexo IV: Archivos CSV casos Covid

Descripción: Los casos publicados como confirmados por criterio clínico epidemiológico con residencia en CABA corresponden a casos confirmados por laboratorio, cuyas muestras y resultados aún no han sido ingresados al SNVS-SISA. Dicha información será actualizada una vez listas las herramientas de interoperabilidad actualmente en desarrollo y dichos casos serán reclasificados automáticamente en función del algoritmo de laboratorio vigente.

Formato: CSV

Temas: COVID-19

Tabla 20 - Esquema de datos CSV casos covid

Título de la columna	Tipo de dato	Descripción
id_evento_caso	Número entero (integer)	Número de caso
sexo	Texto (string)	Sexo
edad	Número entero (integer)	Edad
edad_años_meses	Texto (string)	Edad indicada en meses o años
residencia_pais_nombre	Texto (string)	País de residencia
residencia_provincia_nombre	Texto (string)	Provincia de residencia
residencia_departamento_nombre	Texto (string)	Departamento de residencia
carga_provincia_nombre	Texto (string)	Provincia de establecimiento de carga
fecha_inicio_sintomas	Fecha ISO-8601 (date)	Fecha de inicio de síntomas
fecha_apertura	Fecha ISO-8601 (date)	Fecha de apertura del caso
sepi_apertura	Número entero (integer)	Semana Epidemiológica de fecha de apertura
fecha_internacion	Fecha ISO-8601 (date)	Fecha de internación
cuidado_intensivo	Texto (string)	Indicación si estuvo en cuidado intensivo
fecha_cui_intensivo	Fecha ISO-8601 (date)	Fecha de ingreso a cuidado intensivo en el caso de corresponder
fallecido	Texto (string)	Indicación de fallecido
fecha_fallecimiento	Tiempo ISO-8601 (time)	Fecha de fallecimiento en el caso de corresponder
asistencia_respiratoria_mecanica	Texto (string)	Indicación si requirió asistencia respiratoria mecánica
carga_provincia_id	Número entero (integer)	Código de Provincia de carga
origen_financiamiento	Texto (string)	Origen de financiamiento
clasificacion	Texto (string)	Clasificación manual del registro
clasificacion_resume	Texto (string)	Clasificación del caso
residencia_provincia_id	Número entero (integer)	Código de Provincia de residencia
fecha_diagnostico	Tiempo ISO-8601 (time)	Fecha de diagnóstico
residencia_departamento_id	Número entero (integer)	Código de Departamento de residencia
ultima_actualizacion	Fecha ISO-8601 (date)	Última actualización

Tabla 20: esquema de datos del DataFrame con casos covid. [MINISTERIO DE SALUD, 2021]

Creación de archivos:

Para realizar las pruebas correspondientes a la manipulación y transformación de DataFrames, se tomaron en consideración 3 tamaños de archivos a evaluar: 200, 600 y 1800 megabytes (MB).

El archivo original es de 1800 MB el cual fue extraído del sitio del Ministerio de Salud del Gobierno Argentino.

Para generar los otros dos ejemplares, se utilizó una técnica de selección de filas para reducir el tamaño del mismo y luego guardarlo cómo un nuevo archivo.

Código:

```
df = pd.read_csv(f'data/600mb.csv', sep=',', header='infer')
df_200_mb = df.iloc[:1100000].to_csv(f'./data/200mb.csv', sep=',', header=True, index=False)
df_600_mb = df.iloc[:3300000].to_csv(f'./data/200mb.csv', sep=',', header=True, index=False)
```

6.5. Anexo V: Funciones y Unit Test

Valoraciones MyFEPS:

```
def valoracion_cpu(herramienta):
    cpu = 100

    for key in result[herramienta]['cpu']:
        carga = 0
        for value in result[herramienta]['cpu'][key]:
            carga = carga + value
        carga = carga / len(result[herramienta]['cpu'][key])

        valoracion = 1 - (carga / cpu)

        print(f'% CPU utilizado archivo {key}: ', carga)
        print(f'Valoracion archivo {key}: ({carga}/100) = {valoracion}')
```

```
def valoracion_memoria(herramienta):
    CMS = p.memory_full_info()[7]

    for key in result[herramienta]['memoria']:
        CMF = 0
        for value in result[herramienta]['memoria'][key]:
            CMF = CMF + value
        CMF = CMF / len(result[herramienta]['memoria'][key])

        valoracion = 1 - (CMF/CMS)

        print(f'Memoria RAM utilizada archivo {key}:', CMF)
        print(f'Valoracion archivo {key}: ({CMF}/{CMS}) = {valoracion}')
```

Ploteo de resultados:

```
def ploting(promedio, axis=['200mb', '600mb', '1,8gb'], plot='tiempos'):
    # Definir ancho grafico
    barWidth = 0.25
    # Definir alto grafico
    if len(promedio[f'pandas-{plot}']) == 0:
        bars1 = [0.0, 0.0, 0.0]
    else:
        bars1 = promedio[f'pandas-{plot}']
    if len(promedio[f'pyspark-{plot}']) == 0:
        bars2 = [0.0, 0.0, 0.0]
    else:
        bars2 = promedio[f'pyspark-{plot}']
    # Definir posiciones en el eje X
    r1 = np.arange(3)
    r2 = [x + barWidth for x in r1]
    # Armar el grafico
    plt.bar(r1, bars1, color='#7f6d5f', width=barWidth, edgecolor='white', label=f'pan-
das-{plot}')
    plt.bar(r2, bars2, color='#557f2d', width=barWidth, edgecolor='white', la-
bel=f'pyspark-{plot}')
    plt.xlabel('archivos', fontweight='bold')
    if plot == 'tiempos':
        plt.ylabel('segundos', fontweight='bold')
    if plot == 'memoria':
        plt.ylabel('gb memoria', fontweight='bold')
    if plot == 'cpu':
        plt.ylabel('% cpu', fontweight='bold')
    plt.xticks([r + barWidth for r in range(len(bars1))], axis)
    # Crear leyendas de los ejes y Mostrar grafico
    plt.legend()
    plt.show()
```

Promedio resultados:

```
def resultados(axis=['200mb', '600mb', '1800mb']):
    promedio = {}
    pandas_keys = result['pandas'].keys()
    for pandas_key in pandas_keys:
        promedio[f'pandas-{pandas_key}'] = []
        try:
            for file in axis:
                promedio[f'pandas-{pandas_key}'].append(
                    sum(result['pandas'][f'{pandas_key}'][file]) / len(result['pandas'][f'{pan-
das_key}'][file])
                )
        except:
            pass
    pyspark_keys = result['pyspark'].keys()
    for pyspark_key in pyspark_keys:
        promedio[f'pyspark-{pyspark_key}'] = []
        try:
            for file in axis:
                promedio[f'pyspark-{pyspark_key}'].append(
                    sum(result['pyspark'][f'{pyspark_key}'][file]) / len(re-
sult['pyspark'][f'{pyspark_key}'][file])
                )
        except:
            pass
    return promedio
```

Pretty json output:

```
def pretty_json(result):
    pretty_json = json.dumps(result, indent=4, separators=(',', ': '), sort_keys=True)
    return pretty_json
```

Unit Tests:

Se hace uso de la librería pytest para realizar dos test por cada herramienta sobre el script correspondiente al caso completo. Los test evalúan el DataFrame resultante, haciendo foco en el orden de las columnas, nombre de las columnas y cantidad de columnas.

```
import pytest
import ipytest

ipytest.autoconfig()

columnas_esperadas = ['fecha', 'provincia', 'promedio_edad_Confirmado',
      'promedio_edad_Descartado', 'promedio_edad_internados_F',
      'promedio_edad_internados_M', 'promedio_edad_internados_NR',
      'promedio_edad_fallecidos_F', 'promedio_edad_fallecidos_M',
      'promedio_edad_fallecidos_NR', 'promedio_edad_cui_F',
      'promedio_edad_cui_M', 'promedio_edad_cui_NR',
      'cantidad_casos_F_Confirmado', 'cantidad_casos_F_Descartado',
      'cantidad_casos_M_Confirmado', 'cantidad_casos_M_Descartado',
      'cantidad_casos_NR_Confirmado', 'cantidad_casos_NR_Descartado',
      'cantidad_casos_cui_Privado', 'cantidad_casos_cui_Público',
      'cantidad_casos_cui_F', 'cantidad_casos_cui_M', 'cantidad_casos_cui_NR',
      'cantidad_casos_fallecidos_Privado',
      'cantidad_casos_fallecidos_Público', 'cantidad_casos_fallecidos_F',
      'cantidad_casos_fallecidos_M', 'cantidad_casos_fallecidos_NR',
      'cantidad_casos_internados_Privado',
      'cantidad_casos_internados_Público', 'cantidad_casos_internados_F',
      'cantidad_casos_internados_M', 'cantidad_casos_internados_NR']

def test_pyspark_schema():
   ps_df = pyspark_caso_real(
      archivo_casos='./data/200mb.csv',
      archivo_provincias='./data/provincias.csv',
      archivo_resultante=f'./output/caso_de_uso_real_pyspark_pytest.csv'
   )
   columnas_actuales = ps_df.columns
   assert len(columnas_actuales) == len(columnas_esperadas)
   assert all([a == b for a, b in zip(columnas_actuales, columnas_esperadas)])

def test_pandas_schema():
   pd_df = pandas_caso_real(
      archivo_casos='./data/200mb.csv',
      archivo_provincias='./data/provincias.csv',
      archivo_resultante=f'./output/caso_de_uso_real_pandas_pytest.csv'
   )
   columnas_actuales = pd_df.columns
   assert len(columnas_actuales) == len(columnas_esperadas)
   assert all([a == b for a, b in zip(columnas_actuales, columnas_esperadas)])
```