Parcial 1

Base de datos 2

Tema: Apache spark

Integrantes:

* Chilleti Emanuel
* Casermeiro Maria Silvia
* Vietto Santiago

Año: 2021

**Indice**

[**Apache Spark**](#_94c9ow7bthco)

**Introdución**

**Ecosistema Big Data**

**Qué es Apache spark**

**Cómo funciona?**

[**Conceptos**](#_wfpuy9obt5yr)

[**Concepto Apache Spark**](#_ktqfbyptd0ml)

[**Componentes**](#_1nkgloer8adj)

[**Material**](#_f1wkh8mveojp)

# **Apache Spark**

Big Data

(gráfico)

Big Data es la ciencia digital responsable de captar, ordenar, almacenar y analizar grandes volúmenes de información (peta-bytes/by day) provenientes de diversas fuentes. Con la finalidad de obtener insights para poder entender al cliente y su comportamiento y así ser capaz de proveer un servicio o producto específicamente dirigido.

Debido al gran volumen de información a manejar y a la velocidad de crecimiento, las infraestructuras Big Data nos permiten dar un orden a los datos y utilizarlos en cualquier formato (imagen, audio, vídeo o texto) en tiempo real.

Las cuatro V del Big Data:

* **Volumen**: Se refiere a la acumulación de gran cantidad de datos que se acumulan diariamente en una empresa. El verdadero reto no está en saber recopilar dicha información, sino en desarrollar un método continuo de recopilación y gestión del dato en directo.
* **Variedad**: Se refiere a las diferentes fuentes y tipos de datos tanto estructurados como no estructurados provenientes no solo de personas, sino de máquinas, sensores, software, etc. ~~Esta variedad en datos no estructurados crea problemas de almacenamiento, minería de datos y análisis de la información.~~
* **Velocidad**: Se refiere a la rapidez a la que se generan los datos en la actualidad. Y a su vez a lo rápido que estos pierden utilidad
* **Veracidad**: Se refiere a la calidad, la predictibilidad y la disponibilidad del dato. ~~Es la variable menos uniforme y menos sencilla de controlar, debido a la dificultad de cerciorarnos de que un dato es 100% fiable.~~

## Ecosistema Big Data

Debido a las características que tiene la Big Data (4 V’s) es necesario disponer de herramientas lo bastante potentes para procesar y almacenar grandes volúmenes de información. Y debido a que la tecnología avanza rápidamente, cada vez aparecen nuevas herramientas de diversas compañías que nos ofrecen nuevas suites para ello. Se presentará una sucinta descripción de cada uno de ellos.

## Historia

Big Data, surgieron de los motores de búsqueda en la década del 2000, principalmente de la mano de Google y Yahoo!. Los proveedores de motores de búsqueda fueron los primeros grupos de usuarios que se enfrentaron a los problemas de escalabilidad en Internet, principalmente en cómo almacenar e indexar todos los documentos en el universo de la Internet.En 2003, Google publicó un artículo llamado «The Google File System»(2003). Subsecuentemente, en 2004, publicó otro artículo titulado «MapReduce: Simplified Data Processing on Large Clusters». Al mismo tiempo, en Yahoo! Doug Cutting (creador inicial de Hadoop) estaba trabajando en un proyecto web de indexación llamado Nutch . Los artículos de Google inspiraron a Doug Cutting a tomar el trabajo que él había hecho hasta la fecha en su proyecto Nutch, e incorporar los principios de procesamiento y almacenamiento propuestos en dichas publicaciones. El producto resultante de esta fusión es lo que hoy se conoce como Hadoop. Muchas otras tecnologías comenzaban a surgir a la par de Hadoop, tales como:

* La rápida expansión del comercio electrónico (e-commerce)
* El nacimiento y el rápido crecimiento de la Internet móvil
* Redes sociales
* Blogs y sitios web basados en contenidos de usuarios.

Estas innovaciones en conjunto llevaron a lo que hoy es conocido como grandes volúmenes de datos. Esto aceleró la expansión del movimiento de Big Data y conllevó al nacimiento de Spark

Historia de Spark: nace en 2009 en la universidad de Berkeley y actualmente se encuentra en la versión 3.1.2. Nació de un paper de Google y a partir de ahí fue evolucionando, paso por los procesos MapReduce, en 2008 aproximadamente surgieron Hive y HBase, que como sabemos son dos herramientas del ecosistema de Hadoop, y a partir del año 2009-2010 salió la primera versión alfa Apache spark. Apache Spark se sitúa dentro del ecosistema de Apache Hadoop por lo cual es como un hijo de este.

## Qué es apache spark

Apache Spark es un framework de programación para procesamiento de datos distribuidos diseñado para ser rápido y de propósito general. Con propósito general, podemos decir que puede incluir desde soporte para análisis interactivo de datos con SQL a la creación de complejos pipelines de machine learning y procesamiento en streaming, todo usando el mismo motor de procesamiento y las mismas APIs.

Apache spark es un sistema de computación de datos basado en Hadoop MapReduce, donde principalmente su filosofía es de dividir o paralelizar el trabajo puesto que normalmente se instala en un cluster de computadoras. La idea es tener N maquinas, por ejemplo 10 computadoras, y cada una de esas instancias va a tener instalada una versión de Apache Spark, de esta manera cuando tengamos que procesar una gran cantidad de datos, imaginamos un fichero grande, podemos dividir ese fichero en 10 partes y cada máquina se encarga de una décima parte de ese fichero, y luego los juntaremos, con lo cual ganamos velocidad y esta velocidad es clave en el mundo de de Big Data.

## Algunos conceptos

Hadoop es una plataforma de software de código abierto para el almacenamiento distribuido (compartir discos duros en el almacenamiento de las computadoras que queramos) y el procesamiento distribuido de conjuntos de datos muy grandes (Big Data) en clusters de computadoras (varias computadoras juntas haciendo de una sola), construidos a partir de hardware de productos básicos, es decir, computadoras normales. Al compartir el procesamiento el almacenamiento podemos procesar mayores cantidades de datos ya que tenemos mayor poder y mayor almacenamiento, y es ahí donde entran los grandes conjuntos de datos o BIg Data, y esto lo hacemos en clusters de computadoras que como vimos antes son el conjunto de computadoras que forman Hadoop.

\_ Se usa Hadoop porque procesa Terabytes al fia, es extremadamente útil para poder hacer big data. Cuando usamos una computadora común y queremos hacer un escalado vertical, los tiempos de búsqueda se vuelven muy largos y eso es muy fastidioso y además podemos tener fallas de hardware, por lo que si se nos daña la computadora que está procesando los datos ya no tendremos manera de poder trabajarlo y los tiempos de procesamiento pueden ser muy grandes, y es por eso que se usa Hadoop, para poder compartir la carga del procesamiento entre muchas computadoras, y si una computadora se daña, no pasa nada. Hadoop hace el procesamiento por lotes y esto es lo que lo hace muy especial. Por otro lado, el escalamiento horizontal es lineal, lo que significa que podemos agregar la cantidad de computadoras que queramos a Hadoop sin problema, entonces, si tenemos un problema que requiere de un set de un millón de datos, un set de datos de 100 millones de personas, etc, podemos usar Hadoop en escalamiento horizontal y cuántas computadoras necesitemos para resolver el problema.

\_ En el mundo de Hadoop tenemos los siguientes elementos:

* Motores de Query
* Ecosistema de Hadoop: todos los programas que nos permiten trabajar con big data. Tenemos HDFS que es el almacenamiento distribuido de los datos, luego tenemos MapReduce que es el procesamiento distribuido de esos datos, es decir, cómo procesamos los datos para poder trabajarlos con diferentes queries, machine learning, o cualquier otro programa. Si queremos mostrar información usamos un Query que es como un software llamado Hive, si queremos hacer machine learning con la información que tenemos en nuestro set de datos podemos usar Mahout. Tenemos programas para poder controlar el ecosistema desde una interfaz gráfica, tenemos otras aplicaciones que nos permiten integrar los datos con aplicaciones como bancos o transacciones, y podemos usar NoSQL para guardar nuestros datos en una base de datos de forma distinta.
* Almacenamiento externo de datos

Hadoop MapReduce: este distribuye el procesamiento de datos del cluster o el grupo de computadoras. Divide los datos en particiones que son mapeadas (transformadas) y luego son reducidas (agregadas) por las funciones de mapeo y reducción que nosotros definamos. MapReduce es resistente a los fallos, donde un nodo master de aplicaciones monitorea los mapeos y reductores en cada partición. Entonces proveemos funciones en cualquier lenguaje de programación que podamos usar para poder transformar nuestra información y así agregarla, y de ahí el nombre MapReduce.

## Caracteristicas

Apache Spark es un **motor de procesamiento distribuido responsable de orquestar, distribuir y monitorear** aplicaciones que constan de múltiples tareas de procesamiento de datos sobre varias máquinas de trabajo, que forman un clúster..

Como ya hemos mencionado, es posible **leer los datos desde diferentes soluciones de almacenamiento persistente** como Amazon S3 o Google Storage, sistemas de almacenamiento distribuido como HDFS, sistemas key-value como Apache Cassandra, o buses de mensajes como Kafka.

A pesar de ello, Spark n**o almacena datos** en sí mismo, sino que tiene el foco puesto en el procesamiento. Este es uno de los puntos que lo diferencian de Hadoop, que incluye tanto un almacenamiento persistente (HDFS) como un sistema de procesamiento (MapReduce) de una manera muy integrada.

Es importante hablar de la velocidad de procesamiento: la clave es la posibilidad que ofrece Spark para realizar el **procesamiento en memoria**. Esto, y la extensión del popular MapReduce para permitir de manera eficiente otros tipos de operaciones: Queries interactivas y Procesamiento en Streaming.

## Funciones de Apache Spark

1. Está diseñado para cubrir una amplia gama de cargas de trabajo
2. Realiza procesamiento batch, algoritmos iterativos, queries interactivas, procesamiento streaming
3. Ofrece una serie de APIs que permiten a usuarios con diferentes backgrounds poder utilizarlo
4. Permite trabajar con datos más o menos estructurados (RDDs, dataframes, datasets)
5. Se integra de manera muy cómoda con otras herramientas Big Data,
6. Puede ejecutarse en clusters Hadoop y acceder a los datos almacenados en HDFS y otras fuentes de datos de Hadoop (Cassandra, Hbase, Kafka…).

## Más conceptos

* *API’s* (Interfaz de programación de aplicaciones
  + *RDD* Es una abstracción de estructura de datos usada por Spark. Desglosando su nombre podemos decir que son:

Resilientes (resilientes) Con la ayuda de un grafo acíclico dirigido (DAG), los RDD son tolerantes a fallas y capaces de re-computarse ante la caída de un nodo del clúster.

Distribuidos (distributed) La información contenida no está centralizada en un sólo nodo, los datos están distribuidos, en múltiples nodos del clúster. Esto se logra a partir de técnicas de particionado.

Conjunto de datos (dataset) Es una colección particionada de datos formada por valores primitivos o valores de valores (por ejemplo, tuplas de objetos que representan registros de información).

* + *Dataframes* DataFrames is a distributed collection of rows under named columns. In simple terms, it looks like an Excel sheet with Column headers, or you can think of it as the equivalent to a table in a relational database
  + *Dataset* is a strongly-typed, immutable collection of objects that are mapped to a relational schema
* *DAG* (Directed Acyclic Graph): es un grafo dirigido que no tiene ciclos, es decir, para cada nodo del grafo no hay un camino directo que comience y finalice en dicho nodo. ... Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado cluster.

## Componentes

\_ Tenemos 5 componentes principales:

Spark Core: es la base o el conjunto de librerías donde se apoyan el resto de los otros cuatro componentes. Este es el núcleo y la base del framework.

Spark SQL: es un módulo para el procesamiento de datos estructurados y semiestructurados. Podemos transformar los RDD y los data frame, para poder realizar operaciones sobre ellos. Módulo únicamente para el procesamiento de los datos.

Spark Streaming: nos permite la ingesta y procesamiento de datos en tiempo real, por lo cual vamos a tener una fuente, por ejemplo Twitter, y luego vamos a tener nuestro módulo con el que vamos a ingestar los datos de esa fuente y los vamos a volcar a un destino.

Spark MLLib: es una librería bastante completa que contiene numerosos algoritmos de machine learning.

Spark Graph: este nos va a permitir procesar grafos. Aca añadimos el DAG que es el grafo ciclico dirigido, donde nos va a permitir crear operaciones con los grafos con sus nodos y aristas, para realizar operaciones.

Ventajas

1. Velocidad: su diseño se ha enfocado en optimizar el rendimiento en el procesamiento de datos a gran escala, aprovechando conceptos como el procesamiento en memoria y otras optimizaciones. Es 100 veces más rápido que Hadoop y además consiguió el récord mundial de clasificación de datos a gran escala almacenados en disco.
2. Facilidad de uso: dispone de APIs sencillas de utilizar para trabajar con grandes conjuntos de datos. Tiene más de 100 operadores para transformarlos y manipular datos semiestructurados.
3. Motor unificado: viene empaquetado con bibliotecas de nivel superior, que incluyen soporte para consultas SQL, transmisión de datos, aprendizaje automático y procesamiento de gráficos. Estas bibliotecas estándar aumentan la productividad del desarrollador y se pueden combinar sin problemas para crear flujos de trabajo complejos.

## Arquitectura general

Una aplicación Spark está compuesta por el controlador (driver) y trabajadores (executors) que pueden correr localmente en una única JVM (Java Virtual Machine) o usando recursos de un clúster que son administrados por un administrador (de clúster)

(gráfico)

* *Administrador del clúster (cluster manager)* Es el encargado de coordinar la ejecución intermediando entre el controlador y el clúster. Se encarga de lanzar ejecutores a computar tareas. El controlador (driver) solicita al administrador del clúster los recursos necesarios para poder correr la aplicación. La tarea principal del administrador del clúster es ofrecer un servicio externo a las diferentes aplicaciones que requieran cómputos, dividiendo los recursos entre ellas para que puedan ejecutarse. También despacha trabajos al clúster. Las prestaciones de un administrador de clúster son en un modo FIFO (First in, First out)
* *Contexto de Spark (Spark context)* Representa la conexión, o punto de entrada, al motor de Spark. Actúa como un cliente del entorno de ejecución de Spark y cumple el rol de maestro de la aplicación (no confundir con el significado de maestro antes explicado).
* *Nodos trabajadores (worker nodes)* Un nodo, puede verse como una máquina: PC de escritorio, laptop, teléfono móvil, o unidad de cómputo independiente, interconectado con otros nodos en el clúster. Normalmente un nodo cumple el rol de trabajador (worker) es decir, ser capaz de recibir tareas, procesarlas y devolverlas al controlador. Los ejecutores, son los procesos host alojados en el nodo ejecuto, en los cuales las tareas del DAG son computadas. Los ejecutores reservan CPU y memoria y están dedicados a una aplicación Spark específica y liberados cuando la aplicación termina.
* *Controlador (Driver)* Es el proceso donde el proceso main() o punto de entrada de la aplicación es ejecutado. Representa el proceso corriendo el código de usuario que crea un Contexto de Spark (SparkContext), crear RDDs, y realizar cómputos sobre los mismos.

Ejecutan dos tareas:

Convertir el programa de usuario en tareas (tasks) El controlador es responsable de convertir el código de usuario en unidades de ejecución física llamadas tareas.

Planificar las tareas en los ejecutores (executors): El controlador de Spark debe coordinar y planificar todas las tareas a lo largo de todos los ejecutores disponibles

## Ejemplos

1. [ATP](https://atp.io/): utiliza Apache Spark para el desarrollo y ejecución de modelos predictivos y algoritmos de aprendizaje automático para mejorar la relevancia de las campañas de marketing programático.
2. [PanTera](https://pantera.io/): herramienta para explorar grandes conjuntos de datos. Emplea Spark para crear diagramas de dispersión geográfica de miles a millones de puntos.
3. Apache Spark es la solución líder en el campo del análisis y procesamiento de datos en Big Data y Machine Learning, desplazando a Hadoop, con la cual se integra totalmente al estar basado en ella.

## Herramientas

Anaconda: es un ambiente de trabajo para la ciencia de datos que permite hacer funcionar aplicaciones y administrar paquetes. A su vez abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python y R.

\_ Existen varias maneras de instalar Apache Spark:

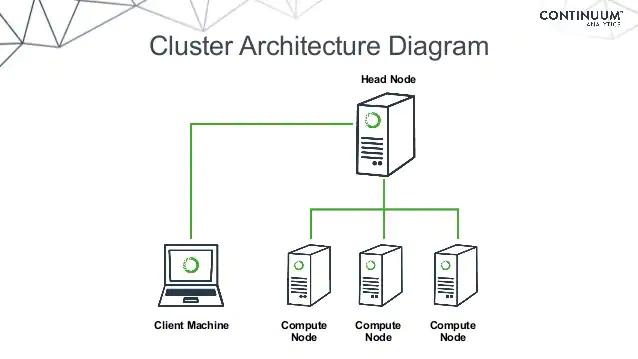
Modo Standalone: simplemente tenemos el HDFS que es el sistema de ficheros distribuidos de Hadoop y por encima tendríamos Apache Spark. Este es la manera más simple, ya que solo descargamos una instancia o versión compilada de Apache Spark y se la añadimos a cada uno de los nodos del cluster

Modo Hadoop V1 (SIMR): tenemos el HDFS, por encima un MapReduce y luego Apache Spark.

Modo Hadoop V2 (YARN): es el más interesante ya que tenemos el HDFS, por encima tenemos un gestor de recursos del cluster como YARN o Mesos y por arriba tenemos Apache Spark. Si lo hacemos sobre Mesos, este se va a encargar del reparto de tareas, es decir hace el papel de cluster Master y lo mismo con YARN que es la instalación más habitual.

## Conceptos

Cluster computacional: dada la capacidad de que una sola máquina nos amplia la necesidad cómputo y muchas aplicaciones en ingeniería de investigación, hacen entonces la idea de crear los clusters computacionales. Entonces un cluster computacional es un conjunto de máquinas las cuales se encuentran conectadas por medio de un switch en red, donde estas máquinas tienen la característica de poder conectarse entre sí, y principalmente son manejadas por una máquina que se les llama el Head Node o el nodo master. El nodo master lo que hace es ocultar la complejidad o el número de máquinas que hay detrás del sistema, y hacer de todo el sistema como si fuera uno solo para el usuario. Los clusters pueden estar conformados por estaciones de trabajo o como computadores personales los cuales se conectan a través de una red, o pueden llegar a ser sistemas especializados.



\_ A continuación tenemos los requerimientos para poder crear un cluster:

* Compute Nodes: este consta de un conjunto de nodos que pueden ser:
* Computadoras (PC’s)
* Servidores
* Computadores especializados
* Network: los computadores deben ser conectados por medio de una red.
* Internet: puede que estos tengan acceso a internet para la instalación de software, configuración de librerías, etc.
* Software: se dispone de diferentes softwares, que permiten la:
* Administración: Kubernetes, OpenHPC, Apache Mesos, Apache Hadoop, etc.
* Monitoreo
* Desarrollo (Development and research): Java, C++, Matlab, OpenFOAM, HTCondor, etc.

\_ Las aplicaciones de software siempre en referencia a aplicaciones en ingeniería o de investigación.

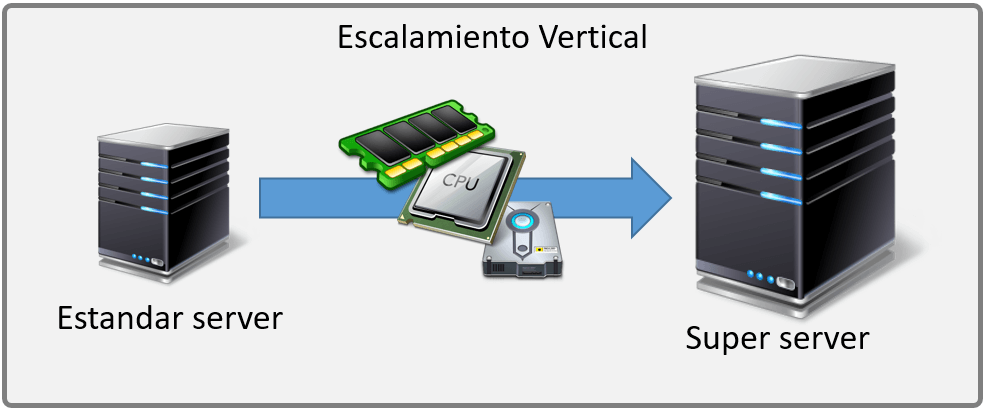
\_ Un cluster se usa comúnmente como un conjunto de máquinas en el cual, del conjunto se delega como una máquina principal o nodo master, al cual deben acceder uno o varios usuarios, los cuales requieren de un recurso computacional elevado. Todos estos usuarios entonces acceden a la máquina principal, envían sus tareas o modelos que quieran ser procesados en el cluster, y la máquina principal se encarga de distribuir entre esos nodos de cómputo utilizando un software de administración o de encolamiento de tareas.

\_ Los clusters tienen gran variedad de aplicaciones dentro del mundo científico:

* Bioinformatica
* Dinámica de fluidos (Computational Fluid Dynamics)
* Matematica
* Dinamica de moleculas (Dynamics Molecular)
* Fisica
* Modelamiento del cambio climático
* Deep learning
* Base de datos

Escalabilidad: es la capacidad del software para adaptarse a las necesidades de rendimiento a medida que el número de usuarios crece, las transacciones aumentan y la base de datos empieza a sufrir degradamiento del performance por las cargas crecientes. El escalamiento es la capacidad del software para adaptarse al creciente número de usuarios, transacciones, etc.

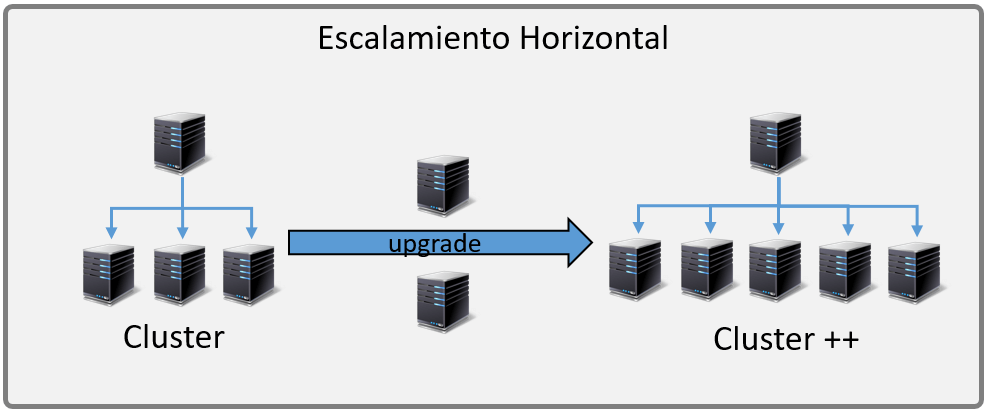
\_ En la práctica existen muchas formas de hacer que un software sea escalable ya que podemos combinar técnicas de software y hardware e incluso arquitecturas de RED, pero nos centramos en la escalabilidad horizontal y vertical, porque sin duda es una de las características más importantes para sistemas de alta demanda o uso crítico:

Escalabilidad vertical: es el más simple, y significa hacer “crecer el hardware” de uno de los nodos, es decir, aumentar el hardware por uno más potente, como disco duro, memoria, procesador, etc, pero también puede ser la migración completa del hardware por uno más potente. El esfuerzo de este crecimiento es mínimo, no tiene repercusiones en el software, ya que solo será respaldar y migrar los sistemas al nuevo hardware.

\_ Este tipo de escalamiento tiene algunos aspectos negativos, ya que el crecimiento está ligado al hardware, y este; tarde o temprano tendrá un límite, donde llegará el momento en el que tengamos el mejor procesador, el mejor disco duro, la mejor memoria y no podamos crecer más o podríamos a lo mejor comprar el siguiente modelo de servidores a un gran costo, haciendo que el rendimiento mejore un poco pero tendríamos el mismo inconveniente cuando salga uno nuevo. A continuación sus ventajas y desventajas:

* Ventajas:
* No implica un gran problema para las aplicaciones, todo el cambio es sobre el hardware.
* Es mucho más fácil de implementar que el escalamiento horizontal.
* Puede ser una solución rápida y económica (comparado con modificar el software).
* Desventajas:
* El crecimiento está limitado por el hardware.
* Una falla en el servidor implica que la aplicación se detenga.
* No soporta la alta disponibilidad.
* Hacer un upgrade del hardware al máximo puede llegar a ser muy caro, ya que las partes más nuevas suelen ser caras con respecto al rendimiento de un modelo anterior.

Escalabilidad horizontal: es sin duda el más potente, pero también el más complicado. Este modelo implica tener varios servidores (conocidos como Nodos) trabajando como un todo. Se crea una red de servidores conocida como Cluster, con la finalidad de repartirse el trabajo entre todos nodos del cluster, cuando el performance del cluster se ve afectada con el incremento de usuarios, se añaden nuevos nodos al cluster, de esta forma a medida que es requeridos, más y más nodos son agregados al cluster.



\_ Para que el escalamiento horizontal funcione deberá existir un servidor primario desde el cual se administra el cluster. Cada servidor del cluster deberá tener un software que permite integrarse al cluster, por ejemplo, para las aplicaciones Java, tenemos los servidores de aplicaciones como Weblogic, Widfly, Websphere, etc, y sobre estos se montan las aplicaciones que queremos escalar. A continuación sus ventajas y desventajas:

* Ventajas:
* El crecimiento es prácticamente infinito, podríamos agregar cuantos servidores sean necesarios.
* Es posible combinarse con el escalamiento vertical.
* Soporta la alta disponibilidad.
* Si un nodo falla, los demás siguen trabajando.
* Soporta el balanceo de cargas.
* Desventajas:
* Requiere de mucho mantenimiento.
* Es difícil de configurar.
* Requiere de grandes cambios en las aplicaciones (si no fueron diseñadas para trabajar en cluster).
* Requiere de una infraestructura más grande.

~~Hadoop: es una plataforma de software de código abierto para el almacenamiento distribuido (compartir discos duros en el almacenamiento de las computadoras que queramos) y el procesamiento distribuido de conjuntos de datos muy grandes (Big Data) en clusters de computadoras (varias computadoras juntas haciendo de una sola), construidos a partir de hardware de productos básicos, es decir, computadoras normales. Al compartir el procesamiento el almacenamiento podemos procesar mayores cantidades de datos ya que tenemos mayor poder y mayor almacenamiento, y es ahí donde entran los grandes conjuntos de datos o BIg Data, y esto lo hacemos en clusters de computadoras que como vimos antes son el conjunto de computadoras que forman Hadoop.~~

~~\_ Se usa Hadoop porque procesa Terabytes al fia, es extremadamente útil para poder hacer big data. Cuando usamos una computadora común y queremos hacer un escalado vertical, los tiempos de búsqueda se vuelven muy largos y eso es muy fastidioso y además podemos tener fallas de hardware, por lo que si se nos daña la computadora que está procesando los datos ya no tendremos manera de poder trabajarlo y los tiempos de procesamiento pueden ser muy grandes, y es por eso que se usa Hadoop, para poder compartir la carga del procesamiento entre muchas computadoras, y si una computadora se daña, no pasa nada. Hadoop hace el procesamiento por lotes y esto es lo que lo hace muy especial. Por otro lado, el escalamiento horizontal es lineal, lo que significa que podemos agregar la cantidad de computadoras que queramos a Hadoop sin problema, entonces, si tenemos un problema que requiere de un set de un millón de datos, un set de datos de 100 millones de personas, etc, podemos usar Hadoop en escalamiento horizontal y cuántas computadoras necesitemos para resolver el problema.~~

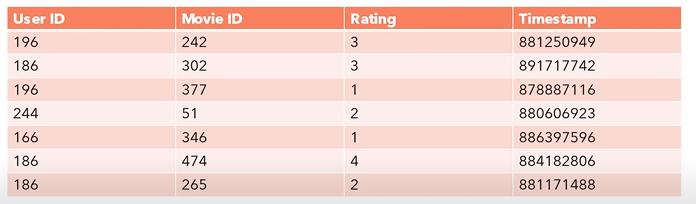
~~\_ En el mundo de Hadoop tenemos los siguientes elementos:~~

* ~~Motores de Query~~
* ~~Ecosistema de Hadoop: todos los programas que nos permiten trabajar con big data. Tenemos HDFS que es el almacenamiento distribuido de los datos, luego tenemos MapReduce que es el procesamiento distribuido de esos datos, es decir, cómo procesamos los datos para poder trabajarlos con diferentes queries, machine learning, o cualquier otro programa. Si queremos mostrar información usamos un Query que es como un software llamado Hive, si queremos hacer machine learning con la información que tenemos en nuestro set de datos podemos usar Mahout. Tenemos programas para poder controlar el ecosistema desde una interfaz gráfica, tenemos otras aplicaciones que nos permiten integrar los datos con aplicaciones como bancos o transacciones, y podemos usar NoSQL para guardar nuestros datos en una base de datos de forma distinta.~~
* ~~Almacenamiento externo de datos~~

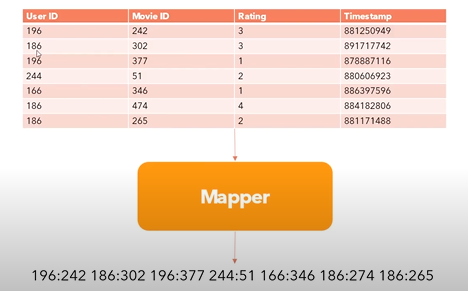
~~Hadoop MapReduce: este distribuye el procesamiento de datos del cluster o el grupo de computadoras. Divide los datos en particiones que son mapeadas (transformadas) y luego son reducidas (agregadas) por las funciones de mapeo y reducción que nosotros definamos. MapReduce es resistente a los fallos, donde un nodo master de aplicaciones monitorea los mapeos y reductores en cada partición. Entonces proveemos funciones en cualquier lenguaje de programación que podamos usar para poder transformar nuestra información y así agregarla, y de ahí el nombre MapReduce.~~

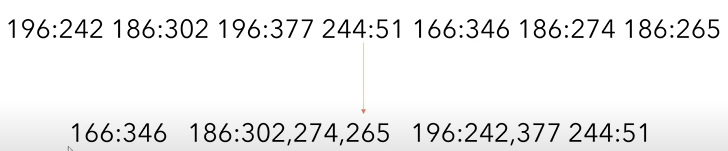
~~\_ Para saber cómo funciona primero analizamos el mapping, el mapper convierte los datos de la fuente en bruto en pares clave/valor, es decir, tenemos nuestros datos input que estamos ingresando, vamos a mapear estos datos, y al final lo que va a quedar es un par clave valor, por ejemplo clave 1 con su respectivo valor, clave 2 con su respectivo valor y así sucesivamente. Entonces así es como se mapea la información.~~



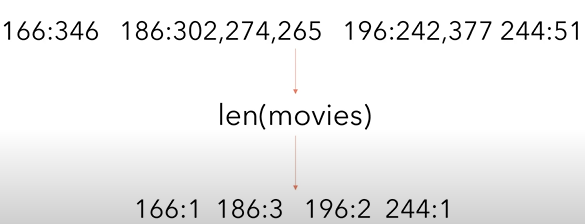
\_ Vemos un ejemplo de MovieLens(u.data) con una tabla con sus atributos y datos. De esta tabla lo que nos importa son los user ID y los movie ID, como no vamos a usar rating y timestamp, entonces queremos mapear un par clave valor para user ID y movie ID. 

\_ Entonces le pasamos los datos al mapper, obteniendo como resultado nuestro par clave valor de la manera que observamos



\_ De esta manera extraemos y organizamos lo que nos importa de nuestro set de datos, eliminando el resto. Entonces con lo anterior extraemos y organizamos, pero como tenemos por ejemplo un User ID repetido con diferentes Movie ID, vamos ayudar a reducir y agrupar todo, por ende aquí es donde hacemos shuffle and sort, por lo que tenemo todo en grupos y más organizado:

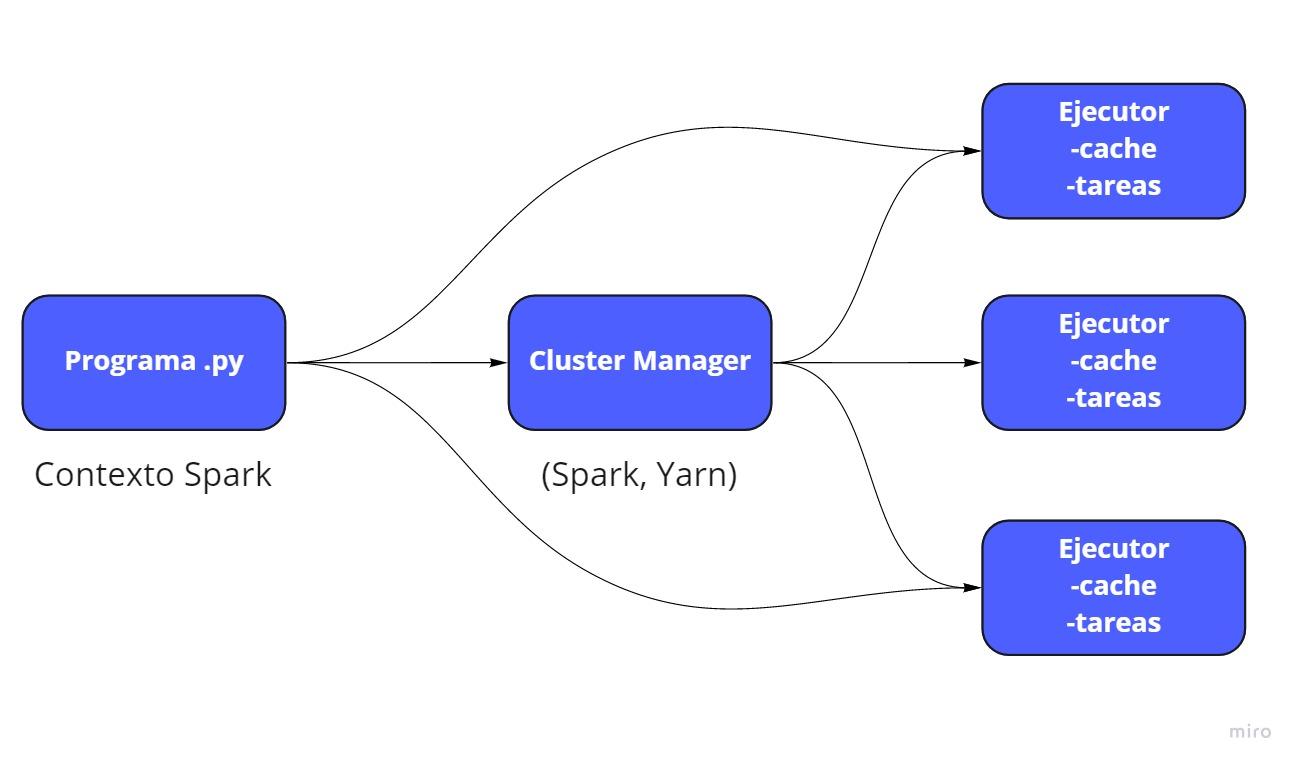
\_ El reductor es el proceso de los valores de cada llave, es decir, este es un pedazo de código que nosotros usamos para poder mostrar nuestra información como nosotros queramos, por ejemplo, como el usuario 186 vio las películas 302, 274 y 265 podemos decir que ese usuario vio una cantidad de 3 peliculas y asi con todos, esto lo hacemos en el reductor, donde le damos cualquier lógica para procesar los datos y mostrarlos de manera organizada:



## Curso 1:

Apache Spark: es un motor para analizar grandes cantidades de datos, entonces para explicar su funcionamiento decimos que tenemos nuestro programa que hemos escrito bajo un contexto de Spark. La razón por la cual usamos Spark para big data es que es escalable, donde con una computadora podemos trabajar con 100 mil datos, también podemos trabajar con un cluster de computadoras, o sea varias computadoras conectadas en serie trabajando con el mismo programa.

\_ Entonces vemos con el siguiente gráfico que tenemos nuestro programa .py que hemos escrito (archivo Python), que corre sobre un cluster manager que dispersa las cargas a otros ejecutores (computadoras) donde se les da las diferentes tareas que hacen.



\_ Spark separa el trabajo en ejecutores múltiples por computadora, idealmente un ejecutor por núcleo de CPU es un simple programa que corre desde una computadora y puede escalar el trabajo horizontalmente a varias máquinas. Otro beneficio de utilizar Spark es que hace todo mucho más rápido, en donde en su sitio web afirma que pueden correr programas 100 veces más rápido en memoria que Hadoop MapReduce, y 10 veces más rápido en disco.

DAG (Directed Acyclic Graph): para trabajar, Spark utiliza un motor DAG, en donde este consiste en no trabajar hasta que se le pida que lo haga, y crea un gráfico con todos los pasos que debe tomar para obtener los resultados.

Interfaz gráfica Spark: este framework tiene una versión gráfica y trabaja en el puerto 4040, y así podemos observar lo que está sucediendo.

¿Quién usa Spark?: empresas como Amazon, Ebay, NASA, Netflix, Groupon, IBM, TripAdviser, Yahoo, Shopify, utilizan Spark para manejar sus datos. Este es fácil de usar y se pueden utilizar lenguajes como python, Scala, R, etc. Por ende, al momento de trabajar con big data, saber de este framework es una gran habilidad.

Componentes:

* Spark Streaming: sirve para mostrar datos en tiempo real.
* Spark SQL: si tenemos por ejemplo un data warehouse, y queremos trabajar los datos que tenemos dentro de él con Spark, podemos hacerlo con Spark SQL a través de Hive.
* Spark MLLib: utilizado para machine learning.
* Spark GraphX: se utiliza para poder entender mejor los gráficos sociales.

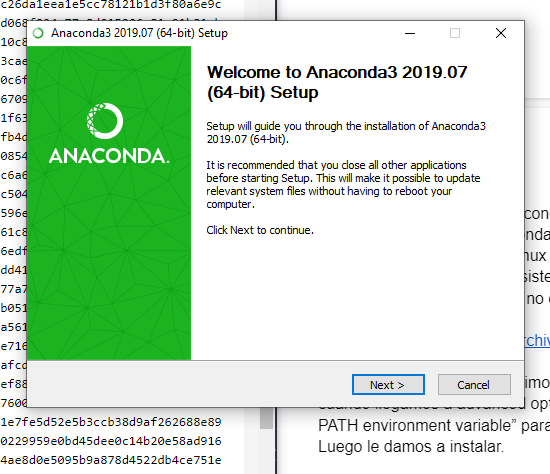
¿Por qué usar Python?: lo utilizamos ya que no se necesita compilar el código, es más fácil, tenemos menos código para escribir. Aunque debemos saber qué Spark en sí fue creado en Scala.

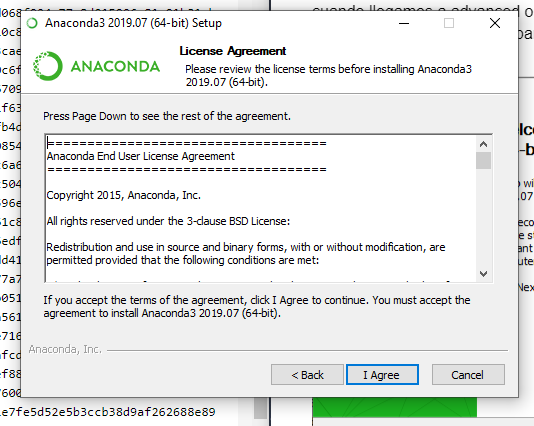
Anaconda: f

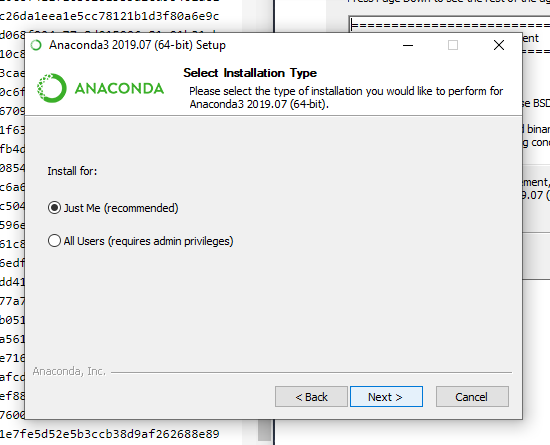
\_ Vamos a instalar Python con Anaconda en donde nos va a permitir tener diferentes ambientes de desarrollo para trabajar con diferentes proyectos. Entonces primero instalamos Anaconda y a través de este descargamos python y creamos nuestro ambiente de desarrollo:

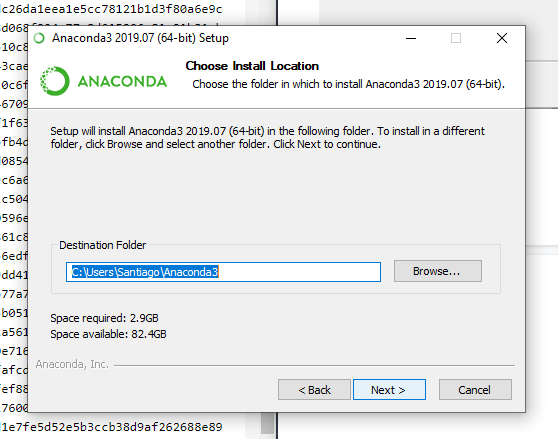
1)\_ Entramos al sitio de Anaconda que es <https://anaconda.org/> y entramos a las sección de “Download Anaconda”. Allí seleccionamos el instalador para nuestro software ya sea Windows, Linux o Mac. Es recomendable instalar la versión 3.7, y esta se encuentra para todos los sistemas operativos en la parte “ADDITIONAL INSTALLERS”, porque las versiones nuevas no cuentan con Python 3.5.

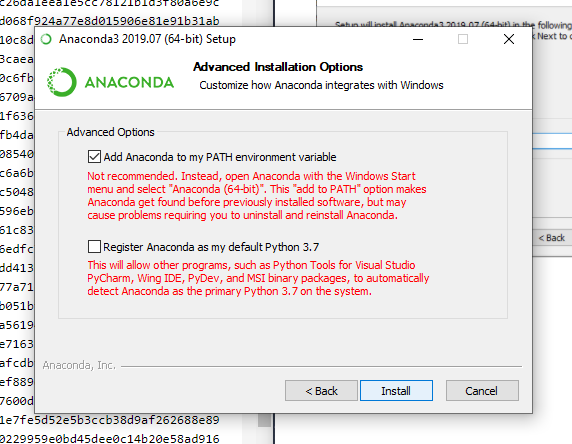
<https://repo.anaconda.com/archive/> (Anaconda3 - 2019.07)

2)\_ Una vez descargado, abrimos el ejecutable, damos siguiente a todos los pasos, y cuando llegamos a advanced options, damos click en la opción “Add Anaconda3 to my PATH environment variable” para permitirnos poder correr anaconda desde la terminal. Luego le damos a instalar.

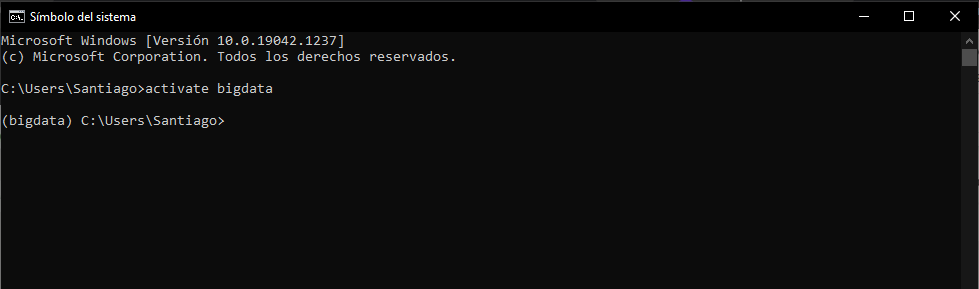
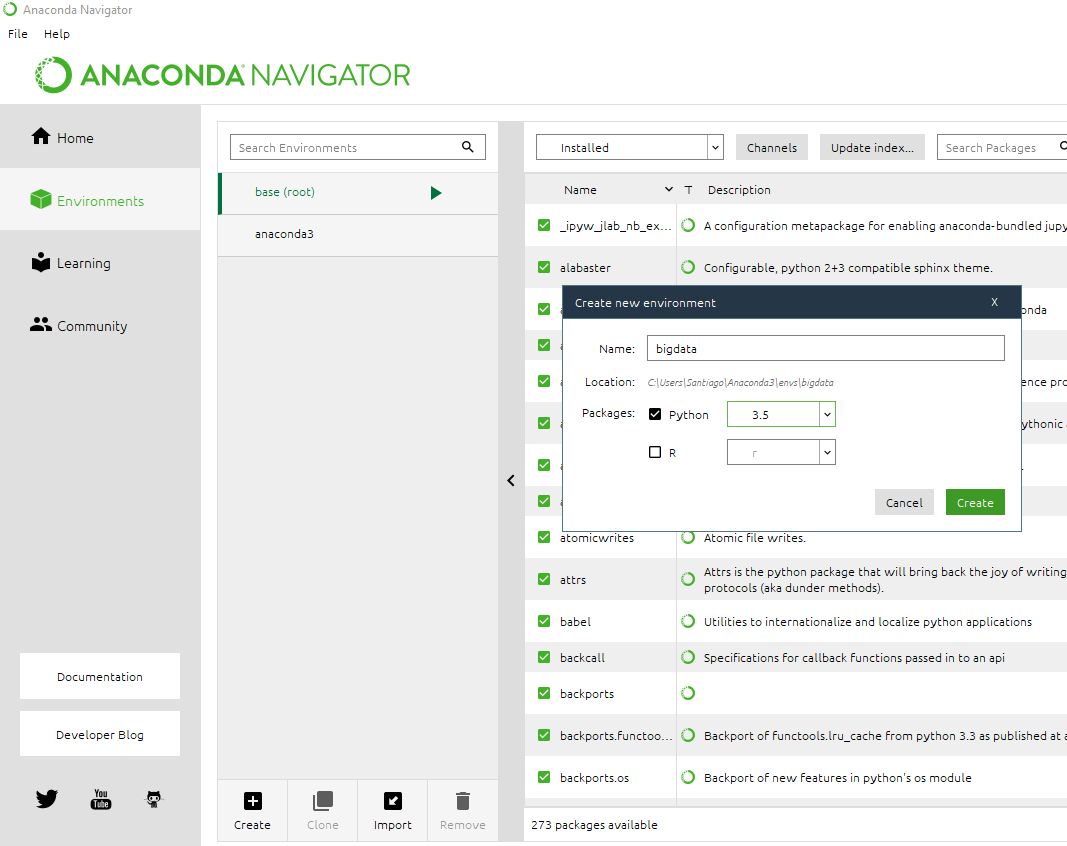








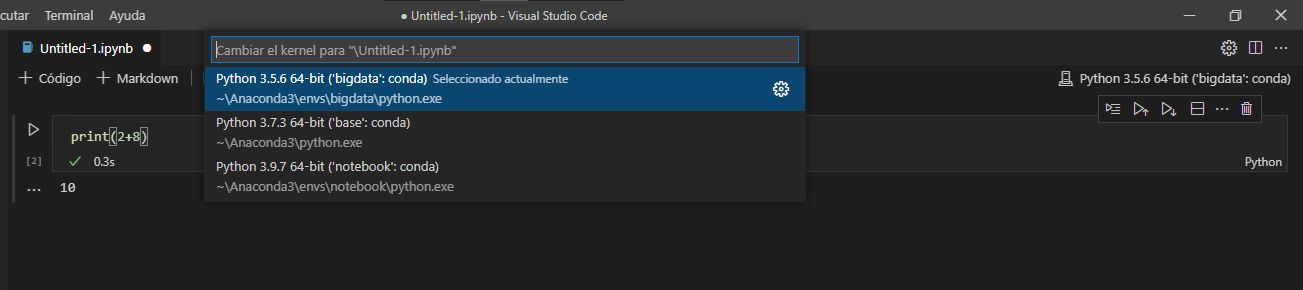
3)\_ Una vez instalado, abrimos el programa. En Home, la parte “Applications on” nos indica el ambiente de desarrollo en el que estamos trabajando.

4)\_ Vamos a Environments y creamos un nuevo entorno de desarrollo. Para esto le damos a “create” y le colocamos un nombre junto con la versión de Python que en nuestro caso es 3.5 ya que Spark trabaja con esta versión, muy importante !. Como seguramente nuestra versión de Anaconda es nueva y no contiene Python 3.5, lo que hacemos es abrir la terminal de Anaconda (Anaconda Powershell Prompt) y colocamos el siguiente comando:

5)\_ Verificamos que está todo creado, activando el entorno mediante la consola. Ahora cuando abramos Visual Studio code vamos a estar de frente en este ambiente de desarrollo, cosa que cualquier programa de pySpark va a estar instalado en este ambiente para así no alterar nada en la computadora.

\_ Como nosotros vamos a trabajar con Visual Studio code, le damos a launch para abrirlo. Pero lo más importante es que se nos cargue Visual Studio code con el ambiente de desarrollo de bigdata.

6)\_ Si el launcher no carga python, debemos instalar python en Visual Studio code, en donde en la sección de extensiones buscamos python y descargamos la versión de Microsoft junto con otras de python y jupyter. Elegimos como Kernel nuestro entorno de bigdata.



\_ Si surgen problemas con el kernel ejecutar en la terminal de Visual Studio code:

pip install pywin32==228

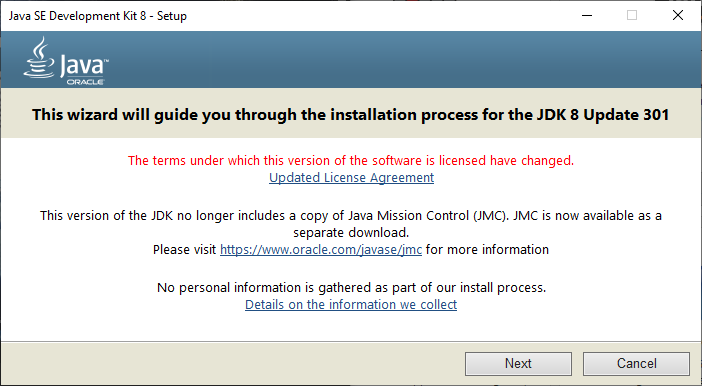
\_ Y si es necesario actualizar:

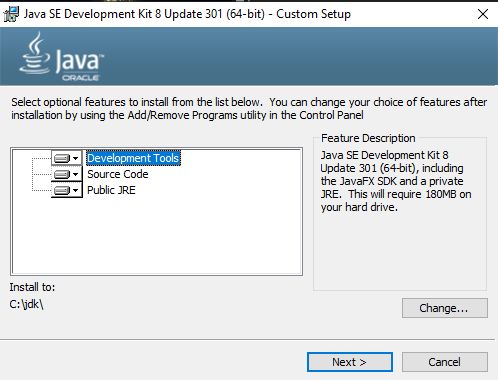
python -m pip install --upgrade pip

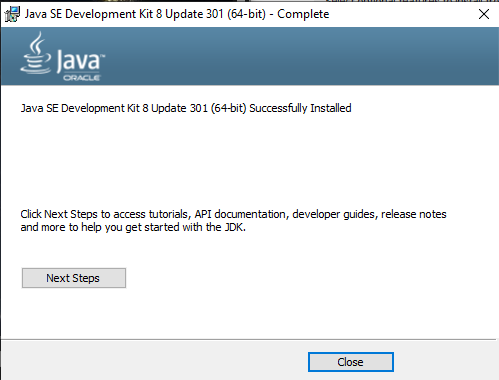
7)\_ Debemos instalar un JDK Java Development Kit, porque Spark al correr sobre Scala corre sobre Java. Por ende buscamos en el buscador jdk y en su página descargamos la el JDK Kit 8. Necesitamos crear una cuenta de usuario y acceder para poder descargarlo. El link es el siguiente:

<https://www.oracle.com/java/technologies/downloads/#java8>

\_ Mientras el archivo .exe se descarga creamos una carpeta en el disco local C llamada jdk, y es ahí donde instalaremos el archivo.



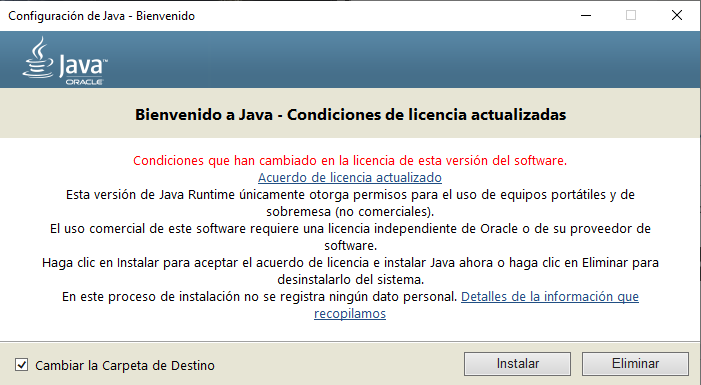


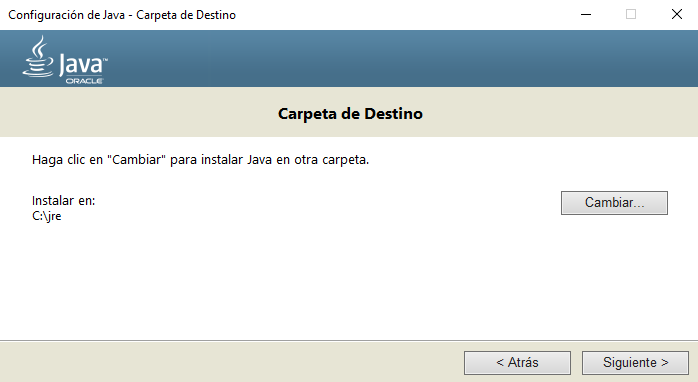


8)\_ Ahora procedemos a instalar un JRE, es decir, un Java Runtime Environment. El link es el siguiente:

<https://www.oracle.com/java/technologies/downloads/#jre8-windows>

\_ Al igual que el JDK, creamos una carpeta en el disco local C llamada jre, mientras el archivo .exe se descarga. Al ejecutar el archivo es importante seleccionar “change destination folder” o cambiar la carpeta de destino:





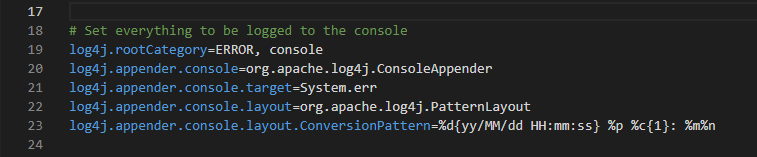
9)\_ Ahora una vez instalado Java, vamos a proceder a instalar Spark, ingresando al sitio:

<https://spark.apache.org/>

\_ En la sección de Download seleccionamos una versión estable de Spark, y tenemos que descargar el archivo como un .tgz (comprimido), donde para abrir esto tenemos que tener Winrar, al seleccionar este link nos lleva a una sección con diferentes mirrors o URLs de donde podemos descargar el archivo, pero tenemos una parte donde nos recomienda descargarlo de un determinado link.

\_ Al igual que el paso 7 y 8, creamos en el disco C una carpeta de nombre spark para tener guardado el framework allí.

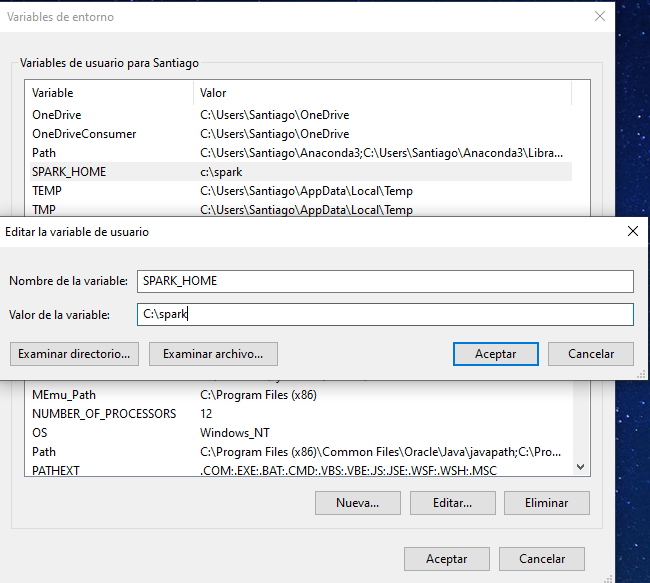
\_ Una vez descargado el archivo, abrimos el archivo comprimido, y todo el contenido lo pegamos en la carpeta Spark. Luego en la carpeta conf, editamos el archivo log4j.properties.template, en donde solo borramos la parte .template y lo abrimos al archivo con algún editor de texto. Una vez abierto procedemos a cambiar en la línea 19 donde dice INFO por ERROR, ya que esto evita que la terminal se llene de números y letras cuando corremos comandos



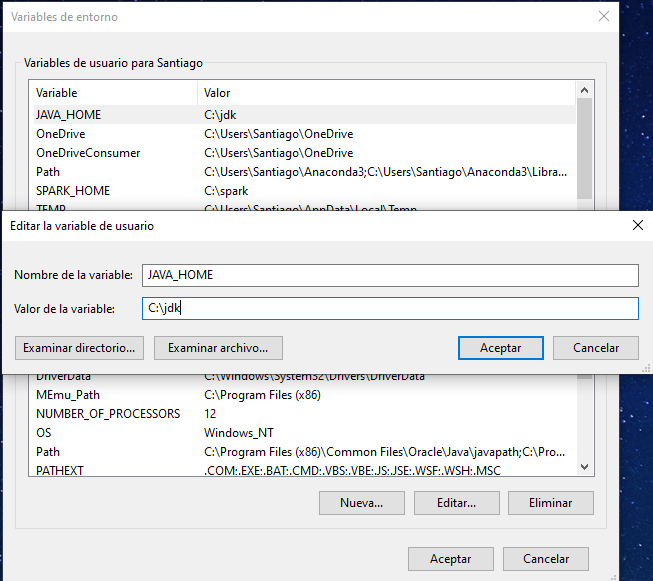
\_ El siguiente paso es engañar a la máquina haciéndole saber que Hadoop está instalado en Windows, y este paso es solo para Windows. Por ende procedemos a instalar (explicar que es), el link de descarga es el siguiente (descargar carpeta porque dentro está el archivo):

<https://drive.google.com/file/d/1XNZz8BG04Gyd_RIw9LVJhkaiQMoxc4uW/view>

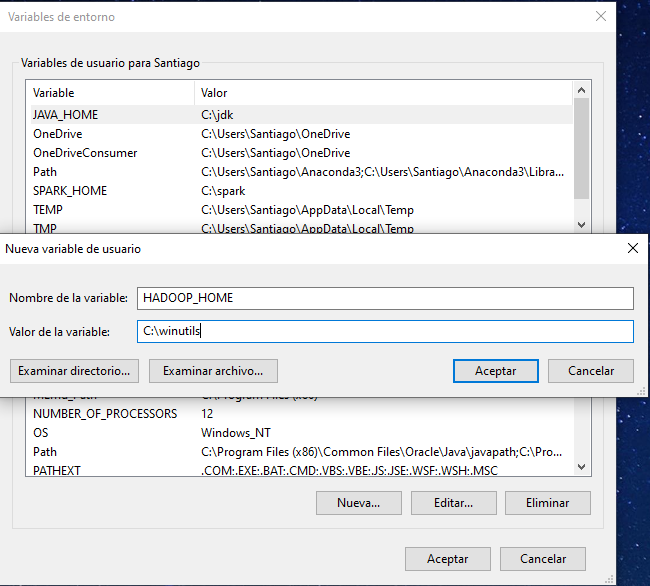
\_ Mientras se descarga creamos una carpeta llamada winutils en el disco local C, donde dentro de esta creamos otra carpeta llamada bin donde guardaremos el recurso winutils.exe.

10)\_ Una vez que hicimos el paso anterior, procedemos a editar algunas environment variables. Para hacer esto entramos a configuración -> sistema -> acerca de -> configuración avanzada del sistema -> variables de entorno, y en esta sección le damos a new, y como nombre le ponemos SPARK\_HOME, y le indicamos que está en la ubicación c:\spark 

\_ Procedemos a hacer lo mismo con JDK:



\_ Lo mismo hacemos con winutils, con la diferencia de que le colocamos como nombre HADOOP\_HOME:



\_ Editamos la variable Path, donde le damos a new y agregamos %SPARK\_HOME%\bin y %JAVA\_HOME%\bin, y le damos a aceptar a todo.

11)\_ Abrimos la consola e ingresamos el comando cd /spark para acceder a la carpeta y colocamos pyspark para verificar que todo ande correctamente. En el caso de que todo funcione bien ahora podremos trabajar con big data y machine learning utilizando Spark y Python. Apretamos ctrl - Z para salir de la consola.

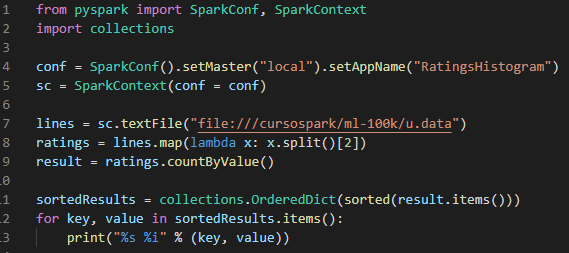
12)\_ Vamos a trabajar con set de datos, y para esto vamos a entrar a una página llamada Gropulens para conseguir datos de rating de películas, es decir, personas que han entrado a esa página y han realizado manualmente ratings. Y con estos datos vamos a crear un sistema de recomendaciones. Primero descargamos un set de datos de 100k, el archivo de descarga es un archivo zip.

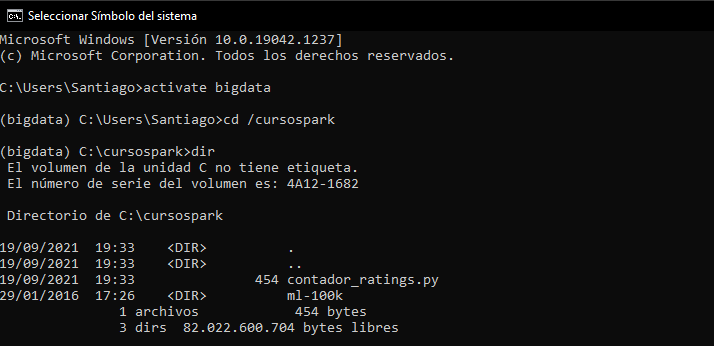
<https://grouplens.org/datasets/movielens/>

\_ Una vez descargado abrimos en un bloc de notas o en Visual Studio code el archivo u.data, ya que este contiene la información que nos interesa. Al abrirlo la primera columna representa el user ID, la segunda columna es el movie ID, la siguiente es el rating que se le dio a la película, y la cuarta columna es un ID. Entonces con los primeros tres datos vamos a trabajar el rating, el usuario y la película. Los índices de las columnas en python son del 0 al N.

\_ En el disco C vamos a crear una carpeta llamada cursospark donde guardaremos nuestro set de datos.

13)\_ Tenemos el siguiente código Python que ejecutaremos de prueba y se va a explicar mas adelante. Este mismo se guarda en la carpeta cursospark.

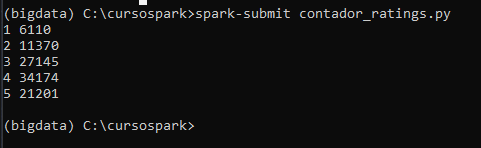


\_ Para probar este archivo abrimos la terminal y activamos nuestro entorno bigdata porque en este tenemos instalado python 3.5. Luego nos movemos a la carpeta cursospark ya que en esa tenemos el archivo .py y podemos hacer dir para verificar que este. 

\_ Si en el archivo Python nos surge un problema con el from pyspark, lo que tenemos que hacer es, en la terminal de Visual Studio code, ejecutar:

pip install pyspark

\_ Ahora para correr Spark, colocamos spark-submit y el nombre del archivo. Podemos ver a continuación que tenemos 6110 con 1 estrella, y así sucesivamente, como curiosidad vemos que la gente lo que más hace es poner un rating de 4 estrellas:



RDD (Resilient Distributed Dataset): así es como funciona de verdad spark, este es un set de datos resiliente (capacidad de los datos para estar disponibles para usuarios o aplicaciones) y está distribuido. Tenemos que aprender a cargar los RDDs con el set de datos que deseamos y a estos RDD debemos correrles funciones con los objetos RDD que estemos creando. Los RDD representan un gran set de datos, podemos usar métodos para alterarlos o podemos filtrarlos.

* Resilient: aguanta la falla de una computadora sin afectar el programa. Recordamos que al trabajar con un cluster de computadoras, si hay una falla, no importa porque tenemos nuestro RDD y podemos trabajar en todas las otras computadoras.
* Distributed: está distribuido en varias computadoras. Sabemos que a Spark lo trabajamos en un cluster de computadoras.
* Dataset: es el set de datos que estamos analizando. Cuando corramos el código en python todo lo que estamos haciendo es cargar nuestro set de datos y convirtiéndolo en un RDD.

\_ Debemos empezar creando todo esto en un SparkContext (sc) y esto se corre a través de un SparkConfiguration (conf), y el código para hacer esto es sc.textFile(“direccion\_archivo”), en la dirección ponemos la dirección donde tenemos el dataset. Es de esta forma como se crea un RDD. Podemos también poner una dirección de un dataset en la nube.

\_ Los RDD no solamente pueden ser creados con documentos .text sino que podemos usar otros documentos como hivecontext, JDBC, cassandra, HBase, Json, CSV, sequence, object files. Entonces, todas estas extensiones de documentos pueden abarcar lo que conforman nuestro set de datos y lo que queremos organizar.

\_ Podemos correr diferentes funciones para transformar las RDD, las más comunes son:

* Map: permite tomar un set de datos y transformarlo a otro dada una función que haga esto justamente, por ejemplo, si queremos sacar el cuadrado de nuestro set de datos podemos crear un map que apunte a una función que multiplique a los números por sí mismos.
* Flatmap: tiene la misma función que map, pero puede producir varios outputs dado un solo input., es decir, que mediante un solo input o dato de entrada, se pueden producir varios outputs.
* Filter: nos permite filtrar los datos que no queremos que sean procesados del set de datos.
* Distinct: ayuda a obtener datos específicos en un set de datos.
* Sample: nos permite tomar una muestra de datos del RDD para poder hacer un experimento.
* Union, Intersection, Substract, Cartesian: nos permiten hacer intersecciones entre RDDs, Cartesian por ejemplo nos da todas las posibles combinaciones del RDD que estamos trabajando.

\_ A continuación tenemos un ejemplo con map:

rdd = sc.parallelize(1,2,3,4)

redd.map(lambda x: x\*x)

\_ Como vemos, parallelize es nomás una forma de crear un rdd, y nuestro set de datos consiste de los números 1, 2, 3 y 4, entonces con la primera línea estamos creando un RDD. Luego a ese RDD le aplicamos la función map con el cuadrado del contenido, por lo que el resultado sería 1, 4, 9 y 16.

Lambda: varios métodos de RDD aceptan una función como parámetro, por lo que para hacerlo más fácil utilizamos lambda. Este ejemplo con lambda es:

redd.map(lambda x: x\*x), donde le decimos que al x de nuestra función la multiplique por sí misma. Ahora, esto es lo mismo que decir:

def squareIt(x)

return x\*x

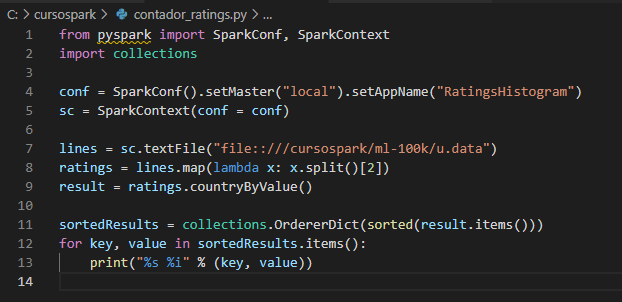
rdd = map(squareIt), donde creamos una función mediante dif (define) que se llame squareIt para x, y retornamos x\*x, y por último llamamos a esta función mediante el map.

\_ Con lambda pasamos una función como parámetro. Lambda es una manera rápida de escribir una función. Es un truco de python.

Acciones en un RDD: esto es muy importante de saber ya que nada ocurre en el RDD hasta qué llamamos a una acción, y ahí es donde empieza a trabajar Spark en el cluster. ahora , tenemos diferentes cosas que podemos correr:

* Collect: este recolecta todos los valores del RDD.
* Count: cuenta los valores.
* countByValue: cuenta cuantas veces un valor ocurre en un RDD.
* Take y Top: sirven para tomar valores de los resultados finales de un RDD.
* Reduce: permite combinar diferentes valores para un valor clave, es decir, permite hacer agregaciones.

\_ Continuando con el práctico, vamos a analizar el código



\_ En la primera línea tenemos pyspark que es el paquete de python que contiene todo lo de Spark, donde importamos SparkConf (Spark Configuration)y SparkContext. También importamos collections porque para este ejercicio en particular utilizamos collections en la parte de código común (línea 11 a 13) para poder mostrar los resultados de cierta manera.

\_ Analizando la línea 4 vemos que conf es igual a SparkConf() que es la función del paquete pyspark, le estamos aplicando .setMaster(“local”) donde con esto le indicamos al programa Spark que vamos a correr esto localmente y que lo vamos a correr solamente en un núcleo de nuestra CPU por eso le decimos local. Con setAppName le decimos el nombre de la aplicación.

\_ En la línea 5 activamos el SparkContext, y como sabemos esto nos sirve para trabajar con Spark y tener nuestras RDD. entonces inicializamos la variable sc.

\_ En la línea 7 le aplicamos a la variable sc la función textFile y es acá donde cargamos nuestro set de datos. Y a todo esto lo pasamos a una variable u objeto RDD que se va a llamar lines.

\_ En la línea 8, al objeto RDD lines, le aplicamos una de las funciones RDD, que en este caso es map (permite tomar un set de datos y transformarlo a otro dada una función), y lo que vamos a hacer es una función split, es decir, dividir nuestra información por líneas, y el número dos significa que como el índice de nuestro set de datos del archivo u.data arranca en 0, el indice 0 corresponde al user ID, el índice 1 corresponde al movie ID, luego en el índice 2 tenemos el rating y esto es lo que queremos, por lo que seleccionamos el índice 2 para extraer los datos. Todo esto lo almacenamos en el objeto ratings.

\_ Recordamos que el programa que estamos escribiendo justamente es un histograma para poder contar los ratings

\_ En la línea 9 lo que hacemos es aplicarle un transformación a la variable ratings con una acción que es countByValue donde este contaba cuántas veces un valor ocurre en un RDD. Y así es como sabemos por ejemplo cuántas veces está el rating número 3, etc.

\_ Finalmente de la línea 11 a 13 mostramos los resultados utilizando la librería de python collections.

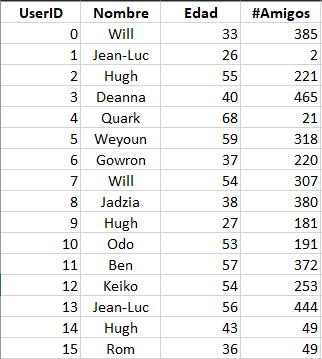
Pares clave-valor: en programación python tenemos lo que son los diccionarios donde estos tienen una clave que representa un valor. A modo de ejemplo vamos a calcular el número promedio de amigos por edad, utilizando los datos de una red social, en este caso de estudio nuestra clave va a ser la edad y el valor va a ser el número de amigos que esa edad tiene en promedio. En lugar de tener una lista de edades y otra lista de número de amigos, lo que hacemos es que guardamos en pares (edad, número de amigos). El uso útil que le podemos dar a esto es a través de la función reduceByKey por ejemplo, donde esta combina todos los valores encontrados para una clave y lo hacemos definiendo una función. Su aplicación sería:

rdd.reduceByKey(lambda x,y: x+y)

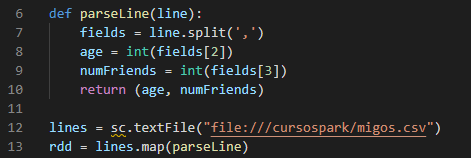
\_ Esa función que estamos viendo combina los valores que queremos, por ende reduceByKey cumple con lo que necesitamos, pero a su vez tenemos otras funciones:

* groupByKey(): agrupa los valores con la misma clave.
* sortByKey(): clasifica los RDD por clave.
* keys() y values(): crean un RDD con solo las claves o solo los valores.

\_ Ahora para mapear la información, si no vamos a modificar las claves nos tenemos que asegurar de llamar la función mapValues() o flatMapValues(), donde solamente si la transformación no afecta a las claves. Hacer esto es importante ya que ayuda a Spark a mantener la partición original del RDD para hacer todo mucho más eficiente.

\_ A continuación vemos la tabla de nuestro set de datos:

\_ Por ejemplo, queremos ver cuál es el número promedio de amigos para un usuario de 33 años. Ahora el código es el siguiente:



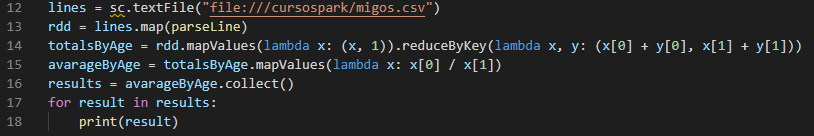
\_ En la línea 6 estamos creando el procesamiento de datos, es decir, cómo queremos ver nuestros datos, dándole todos los parámetros de lo que queremos extraer y cómo queremos crearlo.

\_ En la línea 7 indicamos que separamos los campos por “,” al ser un archivo CSV.

\_ En las líneas 8-9 indicamos que las columnas contienen la edad y el número de amigos, y como queremos aplicarles matemática (promedio) tenemos que indicar que los valores extraídos son en este caso enteros (int).

\_ En la línea 10 es donde creamos el par clave-valor, donde age es la clave y numFriends es el valor.

\_ Las líneas 12 y 13 ya las conocemos, donde en la 12 traemos nuestro set de datos y en la 13 aplicamos el parse Line al set de datos.



\_ En la línea 14 mapValues transforma todos los valores en el par clave-valor, en la parte de rdd.mapValues vemos que tiene (x, 1) y es así porque nuestro primer par es (33, 385), eso es el “lambda x”, y a esto le aplicamos una tupla que contiene el #Amigos y el valor de 1, y lo hacemos así porque para obtener el promedio necesitamos contar todo el número total de amigos vistos para una edad determinada y la cantidad de veces que ocurrió, por eso agregamos el 1. El par se vería de la siguiente manera:

(33, 385) -> (33, (385, 1))

(33, 2) -> (33, (2, 1))

(55, 22) -> (33, (221, 1))

\_ Ahora en reduceByKey estamos sumando todas las veces que se encuentra el mismo valor, y usamos esta función para combinar cosas juntas por la misma clave. Entonces lo que vemos a continuación sería la suma de 2 con 385 y el “‘, 2“ es porque es la segunda vez que encontramos el usuario 33:

(33, (387, 2))

\_ En la línea 15 estamos haciendo el promedio por edad, donde a esta variable le decimos que es igual totalByAge.mapValues y esta vez sacamos el promedio mediante una función.

\_ En la línea 16, mediante la función collect(), estamos recolectando lo que hicimos en el rdd, y así es como recuperamos todo lo que hemos hecho en las líneas 14 y 15 para que luego los pongamos en las variables results.

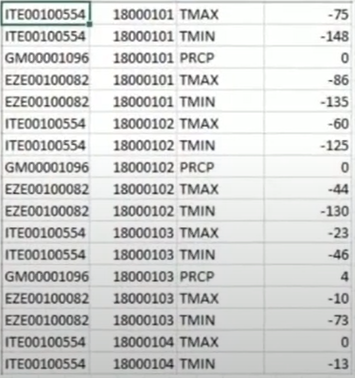
\_ Finalmente en las líneas 17 y 18, estamos mostrando los resultados.

Filtrado de RDDs: el filtrado nos sirve para escoger solo lo que queremos ver, y seguimos utilizando el concepto de clave valor. El filtro es otra opción que podemos usar en un mapa que hemos creado de un RDD, lo transforma eliminando la información que deseamos.

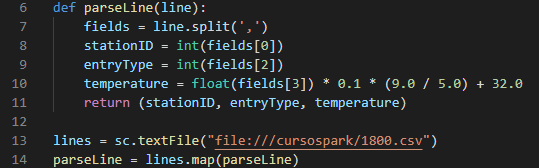
\_ En el ejemplo que vamos a realizar, los datos del clima incluyen temperaturas mínimas, máximas, fechas y otros datos que no queremos como la estación, el día, etc. ya que lo que nos interesa es resolver el problema de encontrar las temperaturas mínimas. Entonces, la función que vamos a utilizar se llama filter(), y esta permite remover los datos de un RDD, y en este ejercicio vamos a filtrar todo lo que no sea TMIN (temperatura mínima). La función que vamos a utilizar para hacer el filtrado es:

minTemps = parasedLines.filter(lambda x: “TMIN” in x[1])

\_ Como sabemos parasedLines contiene el pre procesamiento de datos de nuestro set, y en la función lambda le decimos que solo queremos la temperatura mínima. A continuación vemos un poco de nuestro set de datos donde las columnas son estacionDeClimaID, fecha, tipoDeObservacion y temperatura. De todo esto lo que nos interesa es la temperatura cuando solo dice TMIN.



\_ Entonces analizando el código tenemos:



\_ En la línea 7 separa los datos por comas.

\_ En las líneas 8 y 9 extraemos los campos que queremos de nuestro set de datos.

\_ En la línea 10, usamos lógica de python, en la que hacemos un cambio de temperatura a fahrenheit como para agregarle un poco de complejidad si lo pidiera el cliente.

\_ En la línea 11 retornamos una lista con los valores que parseamos.

\_ Hasta ahora mapeamos los datos inputs. Ahora vamos a crear la RDD que contiene los datos procesados



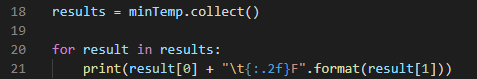
\_ En la línea 15, le aplicamos a nuestra RDD parseLine una función filter() donde le decimos que solo queremos TMIN de la columna 1.



\_ En la línea 16, todo nuestro RDD consiste de TMIN, lo aplicaremos al nuevo map que creamos conteniendo las claves-valor que queremos, y estas serán (stationID, temperatura).



\_ En la línea 17 podemos usar reduceByKey para encontrar la temperatura mínima para cada estación, ya que esta función agrega cada temperatura mínima de todas las estaciones de temperatura. En la función lambda, en x e y combinamos dos temperaturas de la misma estación y luego usamos la función min() para encontrar la mínima de las dos.



\_ En la línea 18 siempre usamos la función collect cuando queremos mostrar, utilizar o recolectar los resultados de la RDD. Siguiendo esto, en el ciclo for vamos a mostrar los resultados en fahrenheit con decimales en un formato de python.

Ocurrencias: podemos contar cuántas veces ocurre algo. Nuestro ejemplo consiste en ver el número de palabras en un documento. Empezaremos viendo una diferencia entre la función map y la función flatmap:

* map(): transforma cada elemento de un RDD a un nuevo elemento, y lo importante acá es que teniendo un input, nos da un output. Ahora si tenemos la frase “hola como estas mi nombre es santiago”, para poder convertir este texto en mayúsculas, o sea “HOLA COMO ESTAS MI NOMBRE ES SANTIAGO”, el código sería:

lines = sc.textFile(“hola.txt”)

fullcaps = lines.map(lambda x: x.upper())

* flatmap(): puede crear varios elementos de uno, es decir, a partir de un input puedo tener varios outputs. Entonces si tenemos el mismo texto “hola como estas mi nombre es santiago”, mediante una función podemos tener cada palabra separada por línea:

“hola

como

estas

mi

nombre

es

santiago”

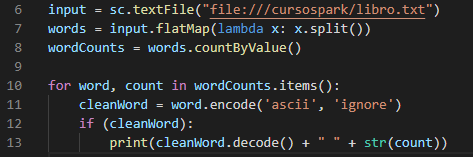
\_ El codigo seria:

lines = sc.textFile(“hola.txt”)

words= lines.flatMap(lambda x: x.split())

\_ Split significa cortar.

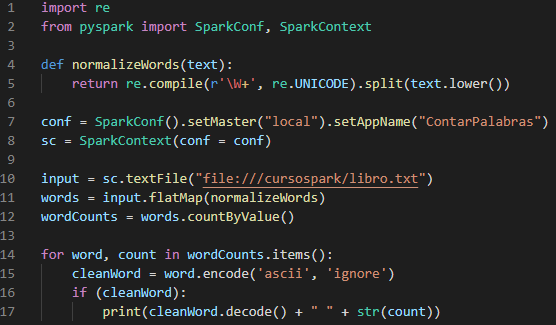
\_ Ahora en el ejemplo que vamos a trabajar, vamos a usar como set de datos un libro de Borges llamado la biblioteca de Babel. Analizando el código tenemos



\_ En la línea 7 aplicamos flatMap() para cortar cada palabra y ponerla en su propia línea, y a esa información la ponemos en su propia RDD llamada words.

\_ En la línea 8 contamos cuántas veces ocurre una palabra y los resultados los guardamos en una variable llamada wordCounts.

\_ En el ciclo for vamos a ir por cada palabra en la nueva lista de palabras que creamos, aclaramos que el ascii se encarga de los errores de codificación de texto, y simplemente mostramos las palabras. Pero los errores con esto es que tenemos comas, signos de puntuación, y no es perfecto, ya que cuenta todas las líneas que se están cortando, por lo que ahora vamos a mejorarlo. Para arreglarlo usamos RegEx (expresiones regulares) y es una función particular de python.



\_ Ahora en la línea 11 le pasamos normalizeWords en lugar del split que hacíamos.

\_ En la línea 4, en la función normalizeWords hemos importado re que es la librería de RegEx.

\_ En la línea 5 la expresión r'\W+' indica que queremos cortar las líneas en palabras, luego hacemos un split para cada palabra identificada y las convertimos en lowercase.

\_ Ahora vamos a determinar cuales son las palabras que más ocurren, es decir, vamos a clasificar por número de ocurrencias. El primer paso consiste en crear un mapa de palabras para convertir cada palabra en esa misma palabra + 1, luego llamamos a reduceByKey() para contar cuántas veces ocurre una palabra. A continuación tenemos la función que realiza esto:

wordCounts = words.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)

\_ En el + 1 es donde vamos a contar el número de ocurrencias y luego aplicamos reduceByKey porque queremos contar cuántas veces ocurre una palabra.

\_ Luego tenemos que invertir nuestros pares clave-valor de (conteo, palabra) a (palabra, conteo), y a esto lo hacemos con la función:

wordCountsSorted = wordCounts.map(lambda (x, y): (y, x)).sorteByKey()

Variables broadcast: estas pueden ser accedidas por los ejecutores (esclavos u otras computadoras en el cluster) cuando sea necesario. Son como unas variables globales. Se aplica de la siguiente manera:

sc.broadcast(): para enviar lo que queremos como broadcast y luego utilizamos .value() para obtener la variable que está en broadcast y utilizarla. A continuación analizamos el código:



\_ En la línea 4 indicamos un diccion vacío llamado movieNames.

\_ En la línea 5 abrimos un archivo con los nombres de las películas

\_ En la línea 7 separamos los datos por “|” porque así están separados los datos en el archivo u.item del set de datos ml-100k.

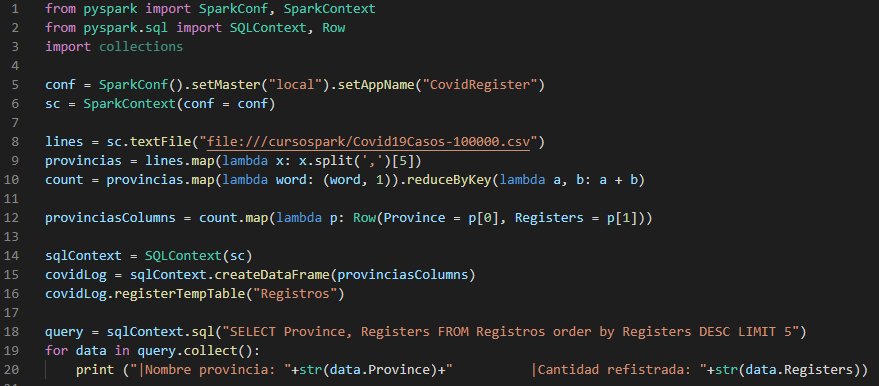
\_ En la línea 8 extraemos el movieID que está como int y luego le asignamos con el igual, el movieID como movieName que se encuentra en el campo 1. Y así es cómo hacemos para asignar que un número corresponde a cierto nombre.

\_ En la línea 14 creamos un objeto llamado nameDict que contiene el broadcast que estamos haciendo del mapeo de movieID con movieName.

\_ En la línea 23 tenemos un mapeo que toma cada par clave-valor. El mapeo reemplazará eso con el nameDict que creamos usando broadcast. Y podemos hacer esto en el map porque hicimos broadcast.

\_ Y de esta forma le decimos al programa que en vez de mostrarnos números, nos muestre el nombre correspondiente a ese número.

Spark SQL: vamos a tratar de ejecutar una aplicación de Spark creada en python que procesa una determinada información, la carga en un RDD que nos permite procesar y distribuir toda la información en memoria y posteriormente convertiremos ese RDD en un dataframe para poder hacer un nuevo procesamiento pero con Spark SQL. Tenemos un archivo CSV de registros de covid de 2020, donde la idea es contar la cantidad de gente registrada por cada provincia. Analizando el código tenemos:



\_ En las líneas 1 y 2, primero importamos las librerías que necesitamos de python.

\_ En la línea 9 hacemos una variable provincias donde procesamos la información del CSV con la operación map un split por comas “,” y tomamos la posición 5 de la columna del archivo ya que en estas están las provincias.

\_ En la línea 10 creamos una variable count para contar el número de veces que se repiten las provincias que aparecen en el archivo y a su vez son la cantidad de personas que fueron registradas de una determinada provincia. Entonces a través de una función map aplicada al RDD provincias, cuya operación es generar w que es la provincia y un 1 haciendo la función de pares, es decir, por ejemplo (Córdoba, 1) y cuando encuentre otro será (Cordoba, 2) y así, y para posteriormente realizar la operación reduceByKey que va a agrupar todas aquellas provincias del RDD. Y como resultado será el count teniendo como contenido las provincias con sus respectivos número de veces que aparece.

\_ En la línea 12 generamos un dataframe para poder procesar ese contenido del RDD con la provincia y la cantidad de veces que se registró una persona de ese lugar, con la potencia de SQL. Entonces sobre el RDD generamos ese dataframe. Indicamos las columnas que va tener el dataframe, que en este caso como lo que primero que obtenemos es el nombre de la provincia y luego el número de veces, usando la función row (fila) le vamos a indicar que en la posición 0 tenemos las provincias y en la posición 1 tenemos los registros.

\_ En la línea 14 generamos el SQLContext para poder indicarle las sentencias SQL a la hora de procesar esa información.

\_ En la línea 15 creamos el dataframe a partir del RDD provinciasColumns que tiene indicadas las columnas de Province y Register.

\_ En línea 16 registramos nuestra tabla y le indicamos como queremos que se llame la tabla.

\_ En línea 18 a través de ese sqlContext podemos indicarle la sentencia SQL que necesitemos y nos va a devolver el resultado. En este caso hacemos un SELECT de Province y Register de la tabla Registros por eso el FROM y está ordenado por los Registers de forma descendente y los 5 máximos.

# **Material**

**Definicion Apache Spark:**

<https://www.youtube.com/watch?v=WR9HnAdYOfs&t=260s> **(visto)**

**Curso Big Data con phyton 1 Windows/Anaconda/VS:** <https://www.youtube.com/watch?v=8jfijXZjKvY&list=PLMF23FOyQZQtIUBt5H_CZguSTvYX6O9Re> (30 videos de 5 min aprox)

**Curso Big Data con phyton 2 Ubuntu:**

<https://www.youtube.com/watch?v=rIuHmjn1IEs>

<https://www.youtube.com/watch?v=BJkThqgDEZw>

<https://www.youtube.com/watch?v=iMOgTbaDJXc&list=PLwH0tlWs8nkQ-56HPCFeKsCJIStOOn_3j>

**Tesis sobre procesamiento de datos con spark - ejemplo**

[Tesis Zequeira Corregida final.pdf](https://www.ridaa.unicen.edu.ar/xmlui/bitstream/handle/123456789/2004/Tesis%20Zequeira%20Corregida%20final.pdf?sequence=1&isAllowed=y)

Qué es apache spark? Pág 33

**Curso Spark Streaming:** <https://www.youtube.com/watch?v=3KfW-otZujs&list=PL9sbKmQTkW076JzfKvnUpGoKVocjkl1S5>

**Spark SQL**

<https://www.youtube.com/watch?v=Mxw6QZk1CMY>

<https://www.youtube.com/watch?v=agqn_-KN4hU>

Por qué su uso?

Arquitectura

Usuarios - roles

Concepto pendientes:

procesamiento por lotes

aprendizaje automático

bases de datos de grafos

motor de ejecución de grafos acíclicos dirigidos (DAG)

Hadoop, basado en el modelo MapReduce

**Tools para big data**

[ANÁLISIS, USO Y DESARROLLO EXPER. DE HERRAMIENTAS Y TECNO. OPEN SOURCE EN BIG DATA](https://upcommons.upc.edu/bitstream/handle/2117/114322/memoria.pdf?sequence=1&isAllowed=y)

**Procesamiento de datos**

[Procesamiento de datos con Spark y Spark SQL - YouTube](https://www.youtube.com/watch?v=8JaTkGocKJI)

**Diseño e implementación del clúster spark**

[0757076\_00000\_0000-1.pdf](https://accedacris.ulpgc.es/bitstream/10553/55628/6/0757076_00000_0000.pdf)

***Links de referencia por orden de aparición en la ppt***

Slide 3 - Introducción

https://medium.com/@christian.zevillanos/big-data-definici%C3%B3n-ecosistema-y-aplicaci%C3%B3n-4b49890a5dc2

Slide 4 - Ecosistema Big Data

<https://www.slideshare.net/isragaytan/apache-spark-meetup>

Slide 6 - Qué es Apache Spark?

https://www.esic.edu/rethink/tecnologia/apache-spark-introduccion-que-es-y-como-funciona

Slide 7-10

https://www.ridaa.unicen.edu.ar/xmlui/bitstream/handle/123456789/2004/Tesis%20Zequeira%20Corregida%20final.pdf?sequence=1&isAllowed=y

Ejemplo de aplicación

https://medium.com/@christian.zevillanos/big-data-definici%C3%B3n-ecosistema-y-aplicaci%C3%B3n-4b49890a5dc2

De las diferencias anteriores se puede observar que el concepto data science se engloba dentro del concepto de big data. En este sentido, la ciencia de datos juega un papel importante en muchas áreas de aplicación.

En resumidas cuentas, data science se desenvuelve dentro del ámbito del big data para obtener información útil a través del análisis predictivo, donde los resultados se utilizan para tomar decisiones inteligentes. De esta forma, sin big data no existiría el concepto de data science. Y sin el segundo, el primero no tendría (u obtendría) tanto valor.

En consecuencia, es fácil entender que el perfil de científico de datos sea uno de los más demandados actualmente en el mercado, tal y como concluye el **informe EPYCE 2017**: posiciones y competencias más demandadas, que realiza anualmente la EAE Business School.

Anaconda es una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de [más de 720 paquetes de código abierto](https://docs.anaconda.com/anaconda/packages/pkg-docs.html).

Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, Las librerías de Ciencia de datos y Conda. Todas estás se instalan de manera automática y en un procedimiento muy sencillo.

