



UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA  
Escuela de Ingeniería Informática



---

# *LEARNING ANALYTICS CON HADOOP/SPARK SOBRE EL DATASET OULAD*

---

Grado en Ingeniería Informática

David Ramírez Gil

TRABAJO FIN DE GRADO - 06/2019

Tutor: Javier Jesús Sánchez Medina



## RESUMEN

El objetivo de este TFG es la planificación, configuración y puesta en marcha de un clúster Big Data basado en Apache Hadoop y Apache Spark para aplicar *Learning Analytics* al dataset ofrecido por la Open University de Reino Unido. Es decir, interpretar y transformar la información recogida sobre el alumnado con el objetivo de adaptar y mejorar la formación impartida. Para ello se ha construido una aplicación web con NodeJS y Bootstrap que conecta con el programa en Scala que procesa y ejecuta las consultas gracias a unos Bash scripts. Finalmente, el usuario podrá estudiar en la aplicación web los datos requeridos y consultar gráficas para comparar resultados.

## ABSTRACT

The objective of this TFG is the planification, configuration and start up of a Big Data cluster based on Apache Hadoop and Apache Spark to apply *Learning Analytics* to the dataset offered by the Open University of United Kingdom. In other words, interpret and transform the collected information about the students with the objective of adapt and improve the education provided. With that purpose in mind, it has been build a web application based on NodeJS and Bootstrap that connect with the Scala program that process and execute the queries through to some Bash scripts. Finally, the user will be able to investigate the data required in the web application and check graphs to compare results.

# Contenido

1.	INTRODUCCIÓN. ESTADO ACTUAL Y OBJETIVOS.....	1
1.	JUSTIFICACIÓN DE LAS COMPETENCIAS ESPECÍFICAS CUBIERTAS.....	2
2.	APORTACIONES.....	3
3.	DESARROLLO.....	4
3.1	INTRODUCCIÓN. PLANIFICACIÓN Y ENTORNO DE TRABAJO .....	4
3.2	DATASET OULAD .....	5
3.3	CLÚSTER BIG DATA HADOOP/SPARK .....	7
3.3.1	INTRODUCCIÓN .....	7
3.3.2	INSTALACIÓN Y CONFIGURACIÓN NODO BASE.....	8
3.3.3	INSTALACIÓN Y CONFIGURACIÓN DE HADOOP .....	9
3.3.4	CONFIGURACIÓN SSH .....	10
3.3.5	CONFIGURACIÓN HDFS .....	11
3.3.6	CONFIGURACIÓN YARN .....	13
3.3.7	MONTAR CLÚSTER DISTRIBUIDO.....	14
3.3.8	INSTALACIÓN Y CONFIGURACIÓN SPARK.....	18
3.4	DESARROLLO DE LA APLICACIÓN SCALA.....	19
3.4.1	INTRODUCCIÓN .....	19
3.4.2	CONFIGURACIÓN ENTORNO DE DESARROLLO.....	19
3.4.3	DESGLOSE DE LA APLICACIÓN .....	22
3.4.4	REPOSITORIO CON EL CÓDIGO DE LA APLICACIÓN SCALA.....	25
3.5	DESARROLLO DE LA APLICACIÓN WEB .....	26
3.5.1	INTRODUCCIÓN .....	26
3.5.2	CONFIGURACIÓN ENTORNO DE DESARROLLO.....	26
3.5.3	DESGLOSE DE LA APLICACIÓN .....	28
3.5.4	ANÁLISIS BASH SCRIPTS.....	44
3.5.5	REPOSITORIO CON EL CÓDIGO DE LA APLICACIÓN WEB .....	46
4.	CONCLUSIONES.....	47
5.	TRABAJOS FUTUROS .....	48
6.	NORMATIVA Y LEGISLACIÓN .....	49
7.	FUENTES DE INFORMACIÓN .....	49

8.	ANEXOS.....	52
8.2	ANEXO 1 – GRÁFICA 1.....	52
8.3	ANEXO 2 – GRÁFICA 2.....	54
8.4	ANEXO 3 – GRÁFICA 3.....	55
8.5	ANEXO 4 – GRÁFICA 4.....	56

## Tabla de ilustraciones

1 Estructura del dataset OULAD .....	5
2 Descarga de Apache Spark .....	18
3 Creación de proyecto Scala con sbt.....	20
4 Configuración versión JDK, sbt y Scala .....	21
5 Configuración fichero build.sbt .....	21
6 Primera parte del código Scala .....	22
7 Segunda parte del código Scala .....	23
8 Tercera parte (fragmento 1) del código Scala .....	24
9 Tercera parte (fragmento 2) del código Scala .....	24
10 Cuarta parte del código Scala .....	25
11 Creación de proyecto Node.js Express App.....	27
12 Selección formulario entidades .....	28
13 Código HTML + Bootstrap formularios .....	29
14 Código Javascript selección formulario .....	29
15 Filtros con checkbox modificados .....	30
16 Código Javascript modificación comportamiento checkbox.....	30
17 Código Javascript POST formulario.....	31
18 Tabla con resultados de la consulta .....	32
19 Código Javascript carga fichero JSON .....	32
20 Código Javascript para carga cabecera.....	33
21 Código Javascript carga tabla completa .....	33
22 Código Javascript POST gráficas .....	34
23 Gráfica 1 - Nivel estudios alumnado.....	35
24 Gráfica 1 - Resultados exámenes y tipos de calificaciones .....	36
25 Código Javascript valores gráfica 1 .....	37
26 Código Javascript generador de gráficas Chart.js.....	37
27 Gráfica 2 - Histórico calificaciones en los distintos módulos .....	38
28 Código Javascript valores gráfica 2 .....	39
29 Código Javascript generador de gráfica 2 (Chart.js + addon).....	40
30 Gráfica 3 - Comparativa calificaciones entre géneros.....	41
31 Gráfica 4 - Comparativa calificaciones entre alumnos con/sin discapacidad .....	41
32 Código Javascript valores gráfica 3/4 .....	42
33 Código Javascript valores gráfica 3/4 (parte 2) .....	43
34 Código Javascript generador de gráficas 3/4 (Chart.js + addon).....	43
35 Código script spark-submit consultas.....	45
36 Código script spark-submit gráficas .....	45
37 Código script descarga y formato fichero JSON .....	46

# 1. INTRODUCCIÓN. ESTADO ACTUAL Y OBJETIVOS

A día de hoy los gobiernos e instituciones educativas de todo el mundo sufren cada vez más una mayor presión para presentar al mundo laboral un número de alumnos altamente cualificados que va en continuo aumento. Esto está provocando un aumento notorio en los gastos y las organizaciones están buscando formas de aumentar la rentabilidad y la eficiencia de sus inversiones. Una de esas medidas se basa en aprovechar mejor el mundo digital en el que vivimos, un mundo en el que vamos dejando un rastro de datos a cada paso que damos. Sin embargo, no basta con recopilar toda información que se genera, hay que interpretarla y transformarla en información útil para toda el área educativa.

Aquí es donde entra en juego el proceso de *Learning Analytics*, donde se pretende aprovechar al máximo los datos recopilados con objetivos tales como mejorar la retención de alumnado, proporcionar una mejor atención y comentarios al alumnado y mejorar la calidad y la efectividad general de la enseñanza provocando así que el alumno pueda desarrollar todo su potencial sin desistir y en un ambiente personalizado a sus necesidades.

Al comienzo del desarrollo de este proyecto, comenzamos la búsqueda de un tema para sin tener una idea clara en la que basarnos, pero si estaba claro que debía ser algo orientado a la ciencia de datos ya que durante los últimos años creemos que no se le ha dado la importancia que esta ciencia requiere los estudios cursados. Este sentimiento crítico y el planteamiento ofrecido por el tutor acerca del Big Data aplicado a *Learning Analytics* hizo que me decantara por este proyecto con el fin de poder aportar algo a mejorar la calidad de la enseñanza.

De la reunión mencionada anteriormente surgieron los objetivos planteados para este trabajo fin de grado. En principio se espera un estudio teórico y posterior puesta en marcha de un clúster Big Data gracias a las tecnologías de Apache Hadoop y Apache Spark, una aplicación web que permita al usuario filtrar la información a recibir y otra aplicación Scala que conecte las peticiones del usuario con el clúster. Además, se ha documentado el proceso de lanzamiento del clúster que alojará un *dataset* cedido por la *Open University* de Reino Unido que comentaremos más adelante.

Por tanto, a lo largo de la memoria se pretenden plantear principalmente los siguientes aspectos del proyecto:

- Estudio del *dataset* a utilizar
- Proceso de estudio, configuración y despliegue del clúster Big Data.
- Desarrollo de aplicación Scala para interacción con clúster.
- Desarrollo de aplicación Web y conexión con aplicación Scala.
- Estudio de posibles resultados obtenidos por el usuario.

## 1. JUSTIFICACIÓN DE LAS COMPETENCIAS ESPECÍFICAS CUBIERTAS

*TI02: capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados.*

La primera parte de este proyecto consistía en el estudio, configuración y despliegue de un clúster Big Data. Además, dado que los recursos de los que disponía el CICEI eran limitados y mi situación personal no me permitía trabajar allí a menudo tuve que desplegar el clúster de manera virtual con una potente máquina doméstica teniendo muy en cuenta la gestión de recursos entre otros aspectos.

*TI03: capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas.*

Una parte importante del proyecto es la aplicación web destinada al usuario, la cual debe permitir de forma sencilla e intuitiva filtrar entre gigas de información y poder visualizar los datos de forma ordenada o incluso en forma de gráficos, permitiendo así una mejor interpretación.

Todo ello ofreciendo siempre los datos de cada una de las tablas del dataset completamente actualizados y unidos en una misma tabla en cada consulta solicitada por el usuario.



*TI06: capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.*

El proyecto completo integra una aplicación web con el clúster permitiendo la subida (resultados de las consultas) y descarga de datos (resultados finales exportados a la web en el formato apropiado).

## 2. APORTACIONES

En primer lugar, se pretende que la primera parte del proyecto, es decir, el estudio, configuración y despliegue de un clúster Big Data con Apache Hadoop y Apache Spark sirva como referencia a cualquier persona que pretenda iniciarse en la ciencia de datos o tenga interés en un campo en continuo crecimiento como es el Big Data.

En cuanto tema escogido sobre el que aplicar la potencia del clúster, tal y como mencione anteriormente en la introducción los procesos de *Learning Analytics* son una herramienta muy joven y en desarrollo que permitirá la exploración de nuevos caminos para la motivación del alumnado, la mejora del proceso de aprendizaje y un acercamiento a las necesidades específicas del alumnado. Creemos firmemente que esta puede ser parte de la solución a los problemas a los que se enfrenta la comunidad educativa y que he sufrido en mayor o menor medida a lo largo de mi vida académica. Dado lo críticos que los estudiantes hemos sido con el sistema educativo parece que dedicar el proyecto fin de grado a desarrollar una herramienta que pretende mejorar la calidad de la educación es la mejor aportación a aquellos que desean mejorar el sistema.

Por último, en el ámbito económico y tecnológico se ha demostrado que no son necesarios grandes recursos para llevar a cabo un sistema sencillo pero efectivo y que las tecnologías utilizadas a pesar de funcionar correctamente aún son muy recientes y tiene un amplio margen de mejora y crecimiento, especialmente los procesos de *Learning Analytics*, cuya implementación y uso debería ser imperativo en las organizaciones e instituciones dedicadas a la educación en la era digital en la que vivimos, donde la problemática de la obtención de la información necesaria está más que solventada de manera natural.

## 3. DESARROLLO

### 3.1 INTRODUCCIÓN. PLANIFICACIÓN Y ENTORNO DE TRABAJO

La metodología planteada al comienzo del proyecto consistía en los siguientes puntos:

- Estudio teórico y práctico acerca del desarrollo de un clúster Big Data y de las tecnologías Apache Hadoop y Apache Spark<sup>1</sup>.
- Estudio y familiarización con el lenguaje a usar para la aplicación Spark, en este caso el elegido fue Scala.
- Planificación, configuración y puesta en marcha del clúster.
- Estudio del dataset ofrecido por la *Open University*.
- Desarrollo de aplicación Scala y aplicación web para manejo del usuario de manera progresiva y simultánea.
- Implementación de gráficos de acuerdo a las peticiones de un docente<sup>2</sup>.
- Pruebas para el despliegue en clúster físico cedido por el CICEI.

Dicha planificación fue seguida y llevada a cabo sin incidencias, salvo la parte final que, debido a problemas con las versiones, el uso compartido del clúster y dificultades personales que impedían visitar el centro con la frecuencia necesaria impidieron el uso de dicho clúster físico.

En cuanto al entorno de trabajo y las tecnologías utilizadas se destaca:

- Apache Hadoop y Apache Spark.
- Linux (CentOS) como sistema operativo para el despliegue del clúster.
- *Dataset OULAD*<sup>3</sup> proporcionado por la *Open University* de Reino Unido.
- Scala como lenguaje de programación para la aplicación Spark.
- Node.js + Bootstrap + Javascript + Bash scripts para la aplicación web de manejo de usuario.
- ChartJS para la creación de gráficos.
- PC Sobremesa:
  - Windows 10 Home 64 bits
  - Intel Core i7-7700 QuadCore @3.60GHz – 4.20GHz.
  - 16GB RAM DDR4

---

<sup>1</sup> Enlace curso Udemy [aquí](#).

<sup>2</sup> Tutor y docente: Javier Jesús Sánchez Medina

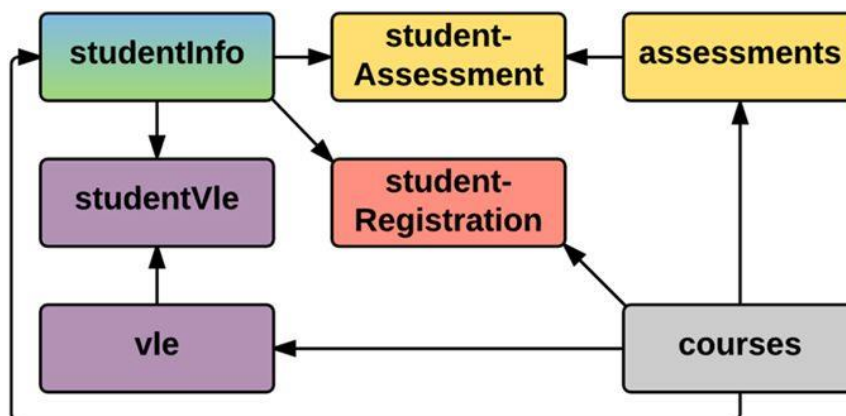
<sup>3</sup> Enlace al dataset [aquí](#).

## 3.2 DATASET OULAD

A continuación, se expondrá la información relativa al *dataset* utilizado para el desarrollo de este trabajo fin de grado. El *dataset* escogido, *OULAD* (*Open University Learning Analytics dataset*), es completamente gratuito y se permite su uso con fines académicos o de investigación.

El *dataset* es el resultado de la información recogida por *Open University*, una de las mayores universidades a distancia del mundo con alrededor de 170000 estudiantes registrados en sus distintos programas. El *OULAD* contiene datos de 2013 y 2014 sobre 22 cursos, 32.593 estudiantes, sus resultados académicos y sus interacciones con el entorno virtual de aprendizaje (VLE). Todos los estudiantes han sido informados de la recopilación de dicha información y se asegura que todos los datos sensibles que puedan permitir identificar a un usuario han sido anonimizados (género, nivel educativo, edad, región, discapacidades, etc.)

En cuanto a la estructura del *dataset*, es importante conocer las tablas que lo forman, así como los campos que las componen ya que estos serán las herramientas de estudio y filtrado que utilizará el usuario de la aplicación.



1 Estructura del dataset OULAD

A continuación, se detallarán los campos que componen las tablas anteriores:

### Tabla *studentInfo*:

- *code\_module*: ID del módulo al que pertenece el estudiante.
- *code\_presentation*: ID presentación del estudiante registrado en un módulo.
- *id\_student*: ID estudiante.
- *gender*: género estudiante.
- *region*: región donde habita el estudiante.

- *highest\_education*: nivel educativo.
- *imd\_band*: índice de calidad de vida de lugar de residencia (Index of Multiple Deprivation).
- *age\_band*: rango edad estudiante.
- *num\_of\_prev\_attemprs*: número de intentos para aprobar el módulo.
- *studied\_credits*: número total de créditos de todos los módulos que esté cursando.
- *disability*: indica discapacidad del estudiante.
- *final\_result*: resultado final del módulo/presentación.

**Tabla courses:**

- *code\_module*: ID del módulo.
- *code\_presentation*: ID de la presentación.
- *length*: duración del módulo/presentación en días.

**Tabla studentRegistration:**

- *code\_module*: ID del módulo.
- *code\_presentation*: ID de la presentación.
- *id\_student*: ID estudiante.
- *date\_registration*: día de registro en el módulo
- *date\_unregistration*: día de baja en el módulo. Si han finalizado el curso el campo está vacío.

**Tabla assessments:**

- *code\_module*: ID del módulo al que pertenece la evaluación.
- *code\_presentation*: ID de la presentación a la que pertenece la evaluación.
- *id\_assessments*: ID de la evaluación.
- *assessment\_type*: tipo de evaluación. Hay tres tipos:
  - Tutor Marked Assessment (TMA).
  - Computer Marked Assessment (CMA).
  - Final Examen (Exam).
- *date*: información acerca del día de cierre de las evaluaciones.
- *weight*: peso de la evaluación. Los exámenes valen siempre el 100%.

**Tabla studentAssessment:**

- *id\_assessment*: ID de la evaluación.
- *id\_student*: ID del estudiante.
- *date\_submitted*: día del envío de la evaluación.
- *is\_banked*: semáforo que indica si la evaluación ha sido transferida de una anterior presentación.
- *score*: puntuación del alumno en la evaluación.

#### Tabla studentVle:

- *code\_module*: ID del módulo.
- *code\_presentation*: ID de la presentación.
- *id\_student*: ID del estudiante.
- *id\_site*: ID del VLE.
- *date*: día de interacción del estudiante con el material.
- *sum\_click*: número de veces que el alumno ha interactuado con el material.

#### Tabla vle:

- *id\_site*: ID del material.
- *code\_module*: ID del módulo.
- *code\_presentation*: ID de la presentación.
- *activity\_type*: rol asociado al material del módulo.
- *week\_from*: semana estimada para empezar a usar el material.
- *week\_to*: semana estimada para terminar de usar el material.

## 3.3 CLÚSTER BIG DATA HADOOP/SPARK

### 3.3.1 INTRODUCCIÓN

En este apartado se explicará paso a paso la instalación y configuración de todas las máquinas, tecnologías y servicios necesarios para poner en marcha nuestro clúster Hadoop/Spark.

Para ello, comenzaremos con una introducción donde se explicará brevemente la creación la máquina virtual a usar, la instalación de un sistema operativo (CentOS en nuestro caso) y la configuración básica de ambas partes para el correcto funcionamiento del clúster. A continuación, se realizará la instalación y configuración de Apache Hadoop y Yarn en un único nodo, para luego montar el clúster virtual distribuido a partir de este primero nodo. Para finalizar instalaremos y configuraremos Apache Spark, dejando nuestro clúster listo para ejecutar nuestras aplicaciones.

Antes de comenzar debemos descargar el software de virtualización, en nuestro caso VirtualBox y una imagen del sistema operativo a utilizar, para la realización de este proyecto se ha escogido CentOS 6.7.

Destacar que, durante los meses de desarrollo de este proyecto, la última versión de VirtualBox liberada fue la 6, pero no funciona correctamente con múltiples distribuciones Linux, entre ellas CentOS. Algunos de los problemas son:

- Problemas con los drivers de la tarjeta gráfica (no se ve el ratón).
- No encuentra correctamente las librerías del sistema.
- Problemas a la hora de ajustar el tamaño de la pantalla y la resolución.

Por tanto, se recomienda descarga la versión 5.2 de VirtualBox<sup>1</sup> y la versión 6.7 de CentOS<sup>2</sup> (en caso de ser el sistema operativo escogido). Ambos enlaces de descarga se proporcionarán al pie de página de sus respectivas webs oficiales.

### 3.3.2 INSTALACIÓN Y CONFIGURACIÓN NODO BASE

Una vez descargado e instalado el software de virtualización, crearemos una nueva máquina virtual con las siguientes características (recomendadas):

- Nombre: nodo1-67.
- Tipo: Linux.
- Versión: Red Hat (64-bit).
- Tamaño de memoria: 4096MB.
- Disco duro: creamos un nuevo disco duro virtual reservado dinámicamente de al menos 30GB.
- Procesador: aumentamos el número de núcleos entre 2 y 4 para un mejor rendimiento de Apache Hadoop.

A continuación, añadimos la imagen ISO de nuestro sistema operativo al controlador en el apartado “Almacenamiento” de la configuración de nuestra máquina e iniciamos la máquina para comenzar la instalación de CentOS.

Los aspectos recomendados a destacar del proceso de instalación son:

- Nombre del host: nodo1.
- Creamos contraseña para el usuario root.
- Permitimos utilización de todo el espacio del disco duro virtual (recordamos que es reservado dinámicamente).
- Para una mayor comodidad y para un posible uso de las distintas interfaces gráficas ofrecidas por Apache se recomienda seleccionar una instalación “Desktop”.
- Una vez finalizada la instalación creamos un nuevo usuario.
- Se recomienda la instalación de las “*Guest Additions*”.

### 3.3.3 INSTALACIÓN Y CONFIGURACIÓN DE HADOOP

Antes de descargar e instalar Hadoop es necesario tener instalado el JDK de Java, puesto que Apache Hadoop está construido en lenguaje Java y es necesario para su funcionamiento.

El JDK de Java no se encuentra instalado por defecto normalmente, aun así, podemos comprobar la versión de Java con el comando.

```
java -version
```

En caso de no estar instaladas, accedemos a la web de Oracle<sup>3</sup> y descargamos la última versión para Linux x64 y lo instalamos mediante RPM o cualquier otro mecanismo del sistema.

```
rpm -ivh jdkXXXXXX.rpm
```

Para asegurarnos de que se usa el JDK que acabamos de descargar podemos utilizar el siguiente comando y en caso de que sea necesario seleccionar la correcta.

```
alternatives --config java
```

Para finalizar con Java, es recomendable configurar las variables de entorno (seguramente el instalador lo haya hecho por nosotros) para evitarnos problemas futuros con Apache Hadoop y Apache Spark.

Debemos utilizar un fichero que cargue las variables cuando el usuario que hemos creamos anteriormente (en nuestro caso "user1") acceda al sistema, por tanto editamos el fichero `"/home/user1/.bashrc"` e incorporamos el acceso a Java.

```
export JAVA_HOME=/usr/java/jdkXXXXX  
export PATH=$PATH:$JAVA_HOME/bin
```

---

<sup>1</sup> Enlace descarga VirtualBox 5.2 [aquí](#).

<sup>2</sup> Enlace descarga CentOS 6.7 [aquí](#).

<sup>3</sup> Enlace descarga JDK [aquí](#).

Puesto que ya nos encontramos en disposición de instalar Hadoop, vamos a la web oficial y descargamos el software. Se recomienda descargar e instalar la versión 2.9 debido a que la versión 3 es muy reciente y ligeramente inestable.

Una vez descargado desempaquetamos el software en el directorio `"/opt/"` y cambiamos el nombre del directorio a `"Hadoop"` para mayor comodidad.

```
tar xvf hadoopXXX-bin.tar  
mv hadoop-XXXX hadoop
```

Modificamos los permisos de la carpeta para que pertenezca al usuario con el que vamos a trabajar, en nuestro caso `"user1"`.

```
cd /opt  
chown -R user1:user1 hadoop
```

A continuación, debemos configurar las variables de entorno tal y como hicimos anteriormente, pero esta vez con Hadoop. Editamos el fichero `"/home/user1/.bashrc"`.

```
export HADOOP_HOME=/opt/hadoop  
export PATH=$PATH:$HADOOP_HOME/bin  
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

Por último, ejecutamos el comando siguiente para comprobar que se accede correctamente y que hemos instalado la versión adecuada (en nuestro caso la 2.9).

```
hadoop version
```

### 3.3.4 CONFIGURACIÓN SSH

Para poder trabajar con Hadoop debemos configurar correctamente la conectividad SSH, para ello debemos seguir los siguientes pasos:

Entramos como usuario no root, en nuestro caso `"user1"` y generamos las claves con el siguiente comando.

```
ssh-keygen
```



Este comando habrá creado el directorio `"/home/user1/.ssh"` en caso de que no existiese y habrá generado dos ficheros con las claves pública y privada. En dicho directorio debemos crear un fichero llamado `"authorized_keys"` que contendrá las claves públicas de todos los nodos del clúster. Puesto que ahora mismo solo tenemos un nodo, creamos ese fichero a partir de la clave pública del nodo en el que nos encontramos.

```
cd /home/user1/.ssh  
cp id_rsa.pub authorized_keys
```

Comprobamos el acceso SSH al propio nodo1 y en caso de acceso correcto cerramos la conexión. No debe pedirnos contraseña de acceso.

```
ssh nodo1  
exit
```

### 3.3.5 CONFIGURACIÓN HDFS

Antes de comenzar con la configuración de HDFS debemos conocer cuáles son los ficheros de configuración principales con los que vamos a trabajar a continuación. Todos ellos se encuentran en el directorio `"/opt/hadoop/etc/hadoop/"`:

- `core-site.xml`: configuración general del clúster
- `hdfs-site.xml`: configuración del sistema de ficheros HDFS

Para comenzar, debemos acceder como el usuario `root` y crear un directorio de datos. En dicho directorio daremos permisos al usuario `"user1"`.

```
mkdir /datos  
chown user1:user1 /datos
```

Accedemos al fichero `"core-site.xml"` y lo editamos de la siguiente forma indicándole que el sistema de ficheros a usar es por defecto (es decir, HDFS) y le indicamos donde se encuentra, en nuestro caso en el propio nodo1, en el puerto 9000

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://nodo1:9000</value>
  </property>
</configuration>
```

Ahora editamos el otro fichero de configuración, el llamado “hdfs-site.xml” y lo editamos de la siguiente forma indicándole que no replique los datos (puesto que de momento solo tenemos un nodo) y los directorios para los datos y metadatos.

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/datos/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/datos/datanode</value>
  </property>
</configuration>
```

Una vez especificados ya podemos crear los directorios de datos y metadatos para el sistema de ficheros HDFS. A pesar de que los crea automáticamente es mejor crearlos nosotros mismos para evitarnos problemas con los permisos.

```
mkdir /datos/namenode
mkdir /datos/datanode
```

Formateamos el sistema de ficheros y comprobamos que dentro del directorio “/datos/namenode” tenemos otro subdirectorio llamado “current”.

```
dfs namenode -format
ls -l /datos/namenode
```

Iniciamos los procesos HDFS y comprobamos con el comando “*jps*” que se están ejecutando.

```
start-dfs.sh
jps
22962 DataNode
23187 SecondaryNameNode
23395 Jps
22837 NameNode
```

Se debe haber creado el directorio “*/data/datanode*” con un subdirectorío denominado “*current*”. Si se desea es posible acceder a la interfaz web de administración para ver el estado del clúster. Ejecute un navegador y vaya a la dirección *nodo1:50070*.

### 3.3.6 CONFIGURACIÓN YARN

Antes de comenzar con la configuración de Yarn debemos conocer cuáles son los ficheros de configuración principales con los que vamos a trabajar a continuación. Todos ellos se encuentran en el directorio “*/opt/hadoop/etc/hadoop/*”:

- *mapred-site.xml*: configuración procesos MapReduce.
- *yarn-site.xml*: configuración de los procesos Yarn.

En primer lugar, debemos asegurarnos de parar HDFS en caso de que estuviese en marcha. A continuación, copiamos el fichero “*mapred-site.xml.template*” a “*mapred-site.xml*” y lo editamos añadiendo la siguiente propiedad que indica que vamos a utilizar Yarn para la gestión de los procesos.

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

Ahora editamos el fichero “*yarn-site.xml*” especificando las siguientes propiedades. En ellas se indica la máquina maestra (ResourceManager), los servicios auxiliares gestionados por MapReduce y la localización de estos en las librerías de Apache Hadoop.

```
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>nodo1</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Iniciamos los servicios de HDFS y Yarn y comprobamos con el comando “*jps*” que se están ejecutando correctamente.

```
start-dfs.sh
start-yarn.sh
jps
1137 Jps
22962 DataNode
23187 SecondaryNameNode
419 ResourceManager
22837 NameNode
541 NodeManager
```

Si se desea es posible acceder a la interfaz web de administración de Hadoop para que está funcionando. Ejecute un navegador y vaya a la dirección *nodo1:8088*. En el apartado “Nodes” del menú lateral podrá ver si el nodo1 se encuentra activo.

### 3.3.7 MONTAR CLÚSTER DISTRIBUIDO

Ahora que ya hemos configurado el nodo principal con todo lo necesario para hacer funcionar el clúster (a excepción de Spark) podemos utilizar dicho nodo para clonarlo y así ahorrarnos tener que montar el resto de nodos de forma manual. Sin embargo, tendremos que realizar algunas pequeñas modificaciones que veremos a continuación en este capítulo.

En primer lugar, debemos asegurarnos de para el clúster y el nodo1. Nos dirigimos al software de virtualización, en nuestro caso VirtualBox, y procedemos a configurar la red interna que utilizara Hadoop para conectar los distintos nodos. Vamos al apartado “Red” de la configuración y verificamos que el adaptador 1 está conectado a una red tipo “NAT”. En la pestaña del adaptador 2 habilitamos el adaptador de red y lo conectamos a una red de tipo “Red Interna”.

Una vez tengamos la red interna configurada procedemos a la clonación de la máquina virtual tantas veces como se requiera (en nuestro caso solo usaremos un clúster de 3 nodos, por tanto, realizamos dos clonaciones). Hay que asegurarse de reinicializar la dirección MAC de las tarjetas de red en el proceso de clonación.

Ahora que ya tenemos todas las máquinas clonadas, procedemos a iniciarlas para configurarlas individualmente antes de comenzar con la configuración de Hadoop. Lo primero que debemos hacer es cambiar el nombre del host, ya que todas tendrán como nombre “nodo1”. Para ello editamos el fichero “*/etc/sysconfig/network*” de los nodos 2 y 3 estableciendo en la etiqueta “HOSTNAME” el valor correspondiente a cada máquina.

```
HOSTNAME=nodoX
```

A continuación, debemos establecer una dirección IP estática a cada máquina y reflejarla en el fichero “*/etc/hosts*” para que puedan conocer las direcciones de los demás nodos.

```
192.168.0.10  nodo1
192.168.0.20  nodo2
192.168.0.30  nodo3
```

Para conectar las maquinas a la red interna con las direcciones especificadas anteriormente podemos utilizar la interfaz gráfica, editando las conexiones de la máquina, editando la interfaz “eth1” y especificando en el apartado de “Parámetros IPv4” la dirección IP anterior de la máquina correspondiente. Este proceso también puede realizarse vía comandos o editando el fichero “*etc/sysconfig/network-scripts/ifcfg-eth1*” y estableciendo la IP correspondiente.

Para comprobar que la red interna funciona correctamente basta con realizar un ping al resto de máquinas y comprobar que se recibe respuesta.

```
ping nodo1
ping nodo2
ping nodo3
```

Como mencionamos anteriormente para poder trabajar con Hadoop debemos configurar correctamente la conectividad SSH, para ello debemos actualizar el fichero "authorized\_keys" de todos los nodos con las claves públicas de todos ellos. Dado que las máquinas han sido clonadas y todas tienen los mismos pares de claves que generamos anteriormente lo mejor será eliminar el contenido del directorio "/home/user1/.ssh" y volver a generar las claves en cada uno de los nodos.

```
rm * /home/user1/.ssh/  
ssh-keygen
```

Por último, debemos generar el fichero "authorized\_keys" y mediante el comando "scp" transferirlo de nodo en nodo incluyendo en él la clave pública del último nodo que lo ha recibido. De esta forma cuando lleguemos al último nodo el fichero contendrá las claves de todas y cada una de las máquinas, por lo que volveremos a usar el comando "scp" para sobrescribir este fichero en todas las máquinas del clúster. Una vez terminado este proceso deberemos ser capaz de conectarnos a cualquier máquina del clúster vía SSH sin necesidad de introducir la clave.

Ahora que ya tenemos la red interna y SSH configurados correctamente en todos los nodos, podemos proceder a configurar Apache Hadoop para que funcione como un auténtico clúster distribuido.

En primer lugar, debemos borrar el contenido del directorio "/datos" de todos los nodos ya que anteriormente la estructura maestro/esclavo se encontraba compartiendo una misma máquina. Sin embargo, ahora tenemos un maestro y dos esclavos.

```
rm -rf /datos/*
```

Modificamos el fichero "/opt/hadoop/etc/hadoop/hdfs-site.xml" en todos y cada uno de los nodos y cambiamos únicamente el valor de la propiedad que permite que los datos se repliquen 2 veces, puesto que tenemos dos nodos esclavos.

```
<property>  
  <name>dfs.replication</name>  
  <value>1</value>  
</property>
```

Ahora debemos indicar qué máquinas van a actuar como esclavos en el clúster, para ello editamos el fichero `"/opt/hadoop/etc/hadoop/slaves"` en todos y cada uno de los nodos.

```
nodo2  
nodo3
```

Antes de arrancar el clúster debemos volver a crear el HDFS y permitir en el Firewall de las máquinas las conexiones de la red interna. Esto último puede hacerse desde la interfaz gráfica o mediante la utilidad `iptables` desde la consola de comandos.

```
hdfs namenode -format  
iptables -A input -i eth1 -j ACCEPT
```

Para finalizar podemos iniciar todos los servicios y mediante el comando `"jps"` comprobar que se están ejecutando correctamente en los nodos maestro y esclavos.

```
start-dfs.sh  
start-yarn.sh
```

Nodo maestro:

```
jps  
28833 NameNode  
29538 Jps  
29268 ResourceManager  
29062 SecondaryNameNode
```

Nodo esclavo:

```
jps  
6373 Jps  
6086 DataNode  
6232 NodeManager
```

## 3.3.8 INSTALACIÓN Y CONFIGURACIÓN SPARK

Llegados a este punto ya deberíamos tener el clúster Hadoop configurado correctamente y listo para funcionar. Ahora procederemos a la descarga, instalación y configuración de Spark de forma que quede integrado sin problemas con Hadoop, el sistema de ficheros HDFS y el servicio de procesos Yarn. Esta instalación la llevaremos a cabo únicamente en el nodo1, es decir, el nodo maestro. Debido a que todos los ficheros necesarios son copiados a HDFS cuando se lanza una aplicación a través de Spark.

Para comenzar accedemos a la [web oficial](#) de Apache Spark y descargamos el paquete de tipo “Pre-built for Apache Hadoop 2.7 and later” ya que, aunque tenemos la posibilidad de descargar Spark en modo *standalone*, necesitaremos una versión preparada para integrarse en el clúster que hemos configurado anteriormente. En cuanto a la versión recomendamos la 2.3 hasta conocer mejor la estabilidad y compatibilidad de la nueva versión 2.4.

### Download Apache Spark™

1. Choose a Spark release:
2. Choose a package type:
3. Download Spark: [spark-2.3.3-bin-hadoop2.7.tgz](#)
4. Verify this release using the 2.3.3 [signatures](#), [checksums](#) and [project release KEYS](#).

Note that, Spark is pre-built with Scala 2.11 except version 2.4.2, which is pre-built with Scala 2.12.

### Link with Spark

Spark artifacts are [hosted in Maven Central](#). You can add a Maven dependency with the following coordinates:

```
groupId: org.apache.spark
artifactId: spark-core_2.11
version: 2.4.3
```

Latest News

- Plan for dropping Python 2 support (Jun 03, 2019)
- Spark 2.4.3 released (May 08, 2019)
- Spark 2.4.2 released (Apr 23, 2019)
- Spark 2.4.1 released (Mar 31, 2019)

[Archive](#)

 **APACHECON**  
LAS VEGAS: Sept. 9-12, 2019  
BERLIN: Oct. 22-24, 2019

[Download Spark](#)

### 2 Descarga de Apache Spark

Una vez tengamos el fichero descargado debemos descomprimirlo en nuestro directorio Hadoop y cambiarle el nombre para una mayor comodidad.

```
tar -xvf spark-2.3.3-bin-without-hadoop.tgz -C /opt/hadoop/
mv /opt/hadoop/spark-2.3.3-bin-without-hadoop /opt/hadoop/spark
```

Para finalizar todo este proceso, debemos configurar las variables de entorno tal y como hicimos anteriormente, pero esta vez con Spark. Editamos el fichero “/home/user1/.bashrc”. En caso de querer probar Spark solo debemos acceder al directorio “opt/Hadoop/spark/bin” y hacer uso de, por ejemplo, la consola Scala.

```
export PATH=$PATH /opt/hadoop/spark/bin:/opt/hadoop/spark/sbin
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
export SPARK_HOME=/opt/hadoop/spark
```



## 3.4 DESARROLLO DE LA APLICACIÓN SCALA

### 3.4.1 INTRODUCCIÓN

La función de esta aplicación es sin duda la más importante de todo el proyecto, ya que será la encargada de leer, cargar en memoria y realizar la unión de todos y cada uno de los ficheros que componen la base de datos *OULAD* que especificamos anteriormente. También tendrá la misión de recibir las variables generadas por el usuario al realizar la consulta en la aplicación web y darles el formato apropiado para que Spark SQL pueda realizar la consulta. Posteriormente se almacenará en el clúster el fichero resultado de la query que luego descargará la aplicación web para el uso de dichos datos.

### 3.4.2 CONFIGURACIÓN ENTORNO DE DESARROLLO

Para el desarrollo de la aplicación se ha utilizado el entorno de desarrollo IntelliJ IDEA<sup>1</sup> sobre un sistema operativo Linux (CentOS) aunque las posibilidades son muchas y todas ellas válidas para el desarrollo de una aplicación Scala.

En primer lugar, descargamos el entorno de desarrollo de la web oficial y lo instalamos. Una vez tengamos el entorno de desarrollo debemos instalar uno de los requisitos fundamentales para la ejecución de nuestro proyecto, el JDK de Java.

El JDK de Java no se encuentra instalado por defecto normalmente, aun así, podemos comprobar la versión de Java con el comando.

```
java -version
```

En caso de no estar instaladas, accedemos a la web de Oracle<sup>2</sup> y descargamos la última versión para Linux x64 y lo instalamos mediante RPM o cualquier otro mecanismo del sistema.

```
rpm -ivh jdkXXXXXX.rpm
```

---

<sup>1</sup> Enlace descarga IntelliJ IDEA [aquí](#).

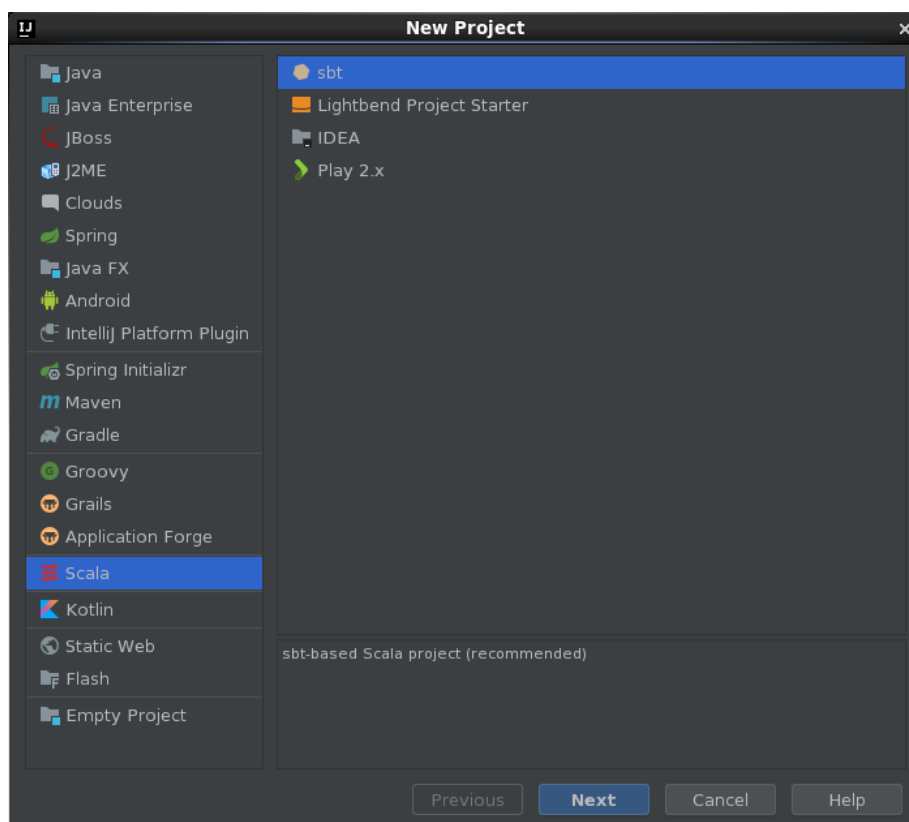
<sup>2</sup> Enlace descarga JDK [aquí](#).

Para finalizar con Java, es recomendable configurar las variables de entorno (seguramente el instalador lo haya hecho por nosotros).

Debemos utilizar un fichero que cargue las variables cuando el usuario que hemos creamos anteriormente (en nuestro caso "user1") acceda al sistema, por tanto editamos el fichero `"/home/user1/.bashrc"` e incorporamos el acceso a Java.

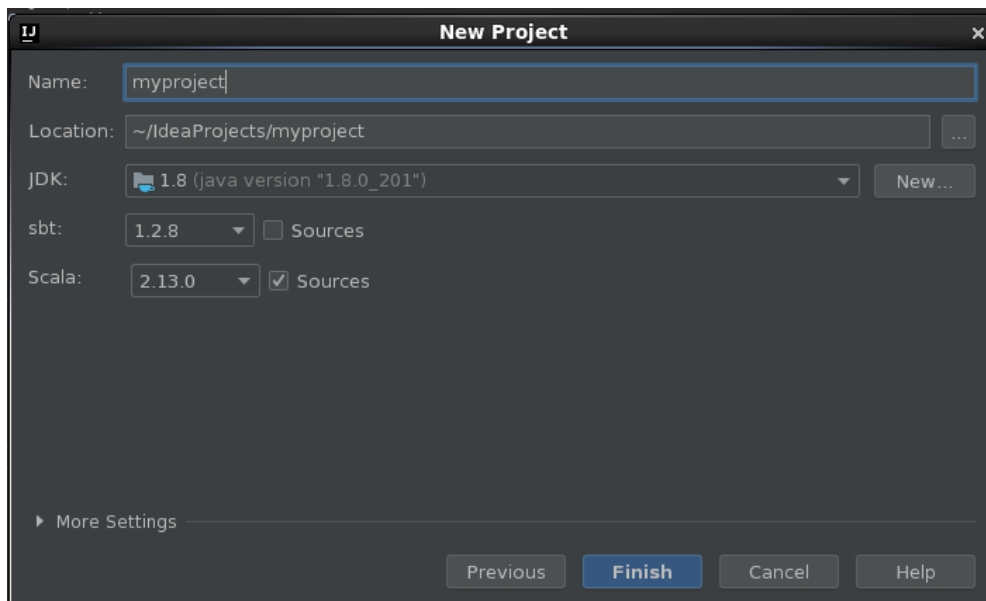
```
export JAVA_HOME=/usr/java/jdkXXXXX
export PATH=$PATH:$JAVA_HOME/bin
```

Llegados a este punto ejecutamos el entorno de desarrollo y creamos un nuevo proyecto Scala con compilador sbt.



### 3 Creación de proyecto Scala con sbt

Seguidamente nos aseguramos de seleccionar el JDK de Java que hemos instalado anteriormente. También podemos personalizar la versión de sbt y de Scala si lo consideramos oportuno.



#### 4 Configuración versión JDK, sbt y Scala

Para finalizar con la configuración de nuestro proyecto deberemos editar el fichero build.sbt que encontraremos en la raíz del proyecto. Ahí tendremos que importar las claves pre configuradas del compilador, así como las dependencias necesarias para nuestro proyecto. En nuestro caso necesitamos "spark-core" y "spark-sql". Además, podemos especificar el nombre de nuestro proyecto, la versión del mismo y la versión del lenguaje Scala en caso de querer utilizar una versión específica distinta a la configurada en la creación del proyecto.

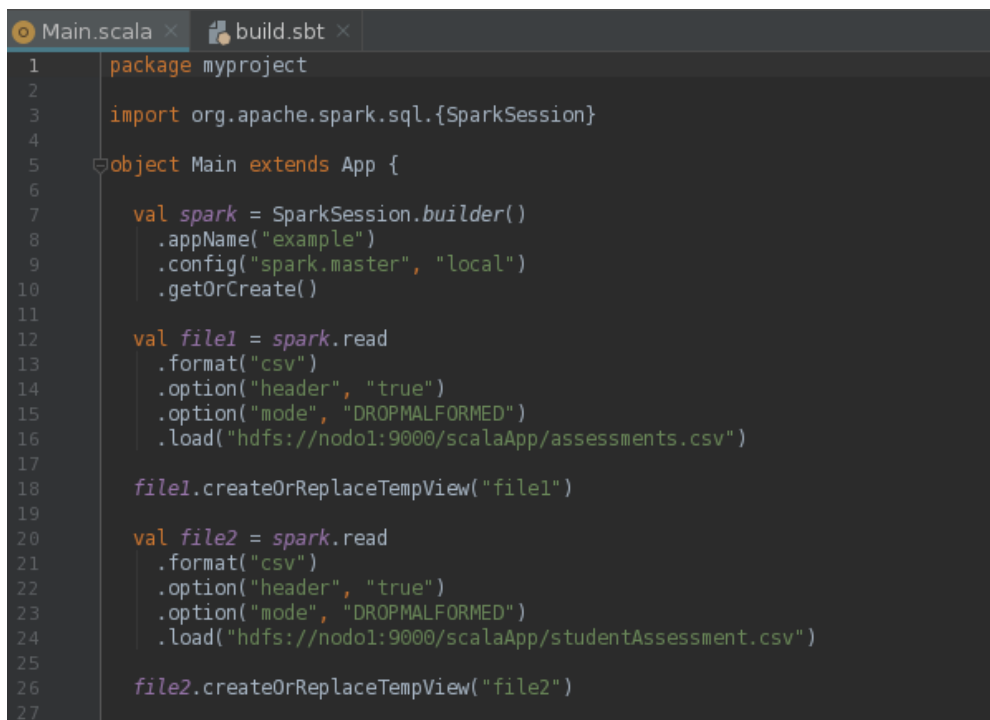
```
1  import sbt.Keys._
2
3  name := "scalaApp"
4
5  version := "0.1"
6
7  scalaVersion := "2.11.8"
8
9
10 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
11 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.2.0"
```

#### 5 Configuración fichero build.sbt

### 3.4.3 DESGLOSE DE LA APLICACIÓN

Podemos dividir la estructura de la aplicación en 4 partes que comentaremos a continuación. Dichos comentarios serán acompañados por capturas de código para una mejor comprensión, sin embargo, no se expondrá el código completo. Para consultar el código completo se podrá acceder a los repositorios indicados en la [siguiente sección](#) de este capítulo.

La primera parte de la aplicación la forman la creación de una SparkSession que nos permitirá interactuar con el clúster, por ejemplo, con las lecturas y escrituras en HDFS. Dichas lecturas son el siguiente paso a seguir en la aplicación. Leeremos uno a uno los siete ficheros con extensión .csv que forman el [dataset](#) a utilizar en el proyecto y generaremos una vista temporal que nos facilitará una posible consulta a dicho fichero.



```
1 package myproject
2
3 import org.apache.spark.sql.{SparkSession}
4
5 object Main extends App {
6
7     val spark = SparkSession.builder()
8         .appName("example")
9         .config("spark.master", "local")
10        .getOrCreate()
11
12    val file1 = spark.read
13        .format("csv")
14        .option("header", "true")
15        .option("mode", "DROPMALFORMED")
16        .load("hdfs://nodol:9000/scalaApp/assessments.csv")
17
18    file1.createOrReplaceTempView("file1")
19
20    val file2 = spark.read
21        .format("csv")
22        .option("header", "true")
23        .option("mode", "DROPMALFORMED")
24        .load("hdfs://nodol:9000/scalaApp/studentAssessment.csv")
25
26    file2.createOrReplaceTempView("file2")
27
```

6 Primera parte del código Scala

La segunda parte de la aplicación consiste en llevar a cabo los “inner join” de las tablas que hemos cargado en memoria. Para ello especificamos los campos de la tabla que comparten valores y el tipo de unión. Además, se ha llevado a cabo la eliminación de ciertos campos que debido a la anonimización del *dataset* o a la nula información que estos aportaban no iban a ser utilizados en las consultas. Por acabar se crean las vistas

temporales de las nuevas tablas generadas con el fin de utilizarlas tanto en las consultas regulares como en las gráficas generadas que veremos posteriormente.

```
//INNER JOIN DE LAS TABLAS QUE COMPONEN EL DATASET
val join1 = file1.join(file2, Seq("id_assessment"), joinType = "inner").drop( colNames = "is_banked", "date_submitted", "date")
val join2 = join1.join(file3, Seq("id_student", "code_module", "code_presentation"), joinType = "inner").drop( colNames = "imd_band
val join3 = join2.join(file4, Seq("id_student", "code_module", "code_presentation"), joinType = "inner").drop( colName = "date_unre
val join4 = join3.join(file5, Seq("id_student", "code_module", "code_presentation"), joinType = "inner").drop( colName = "date")
val join5 = join4.join(file6, Seq("id_site", "code_module", "code_presentation"), joinType = "inner").drop( colName = "week_from")
val join_final = join5.join(file7, Seq("code_module", "code_presentation"), joinType = "inner")
join_final.createOrReplaceTempView( viewName = "table")
join1.createOrReplaceTempView( viewName = "tableChart2")
join2.createOrReplaceTempView( viewName = "tableChartMain")
```

### 7 Segunda parte del código Scala

La tercera parte del código consiste en la inicialización de variables y la lectura de parámetros, los cuales son almacenados en las variables inicializadas anteriormente para luego ser formateados de forma que sea compatible con Spark SQL y las queries que posteriormente analizaremos.

Debemos destacar el primer parámetro, que analizaremos posteriormente con más detenimiento, el cual nos indica la entidad objetivo de la query realizada por el usuario o si por el contrario se trata de una petición para generar una gráfica. Esto nos permite evitar asignaciones de variables y queries erróneas, así como optimizar ligeramente la aplicación.

La principal modificación necesaria para el funcionamiento de la aplicación es la transformación de los parámetros que el usuario ha decidido ignorar con el fin de no aplicar dicho filtro y obtener todos los resultados en ese campo. Para ello cambiamos el “undefined” que nos llega de la aplicación web y el script que ejecuta la aplicación Scala por un valor “null”. En el caso de los símbolos o de los valores con múltiples palabras separadas por espacios en blanco se realizan operaciones similares.

```
117 // VARIABLE FILTRO QUERYS/CHARTS
118
119 var filter = args(0)
120
121 if (filter == "assessment") {
122
123     // VARIABLES INTERFAZ QUERY ASSESSMENT
124
125     assessment_type = args(1)
126     if (assessment_type == s"" "undefined" "") {assessment_type = "null"}
127
128
129     weight = args(2)
130     weight_symbol = weight match {
131     case "less" => "<="
132     case "greater" => ">="
133     case "undefined" => "null"
134 }
```

8 Tercera parte (fragmento 1) del código Scala

```
174 education = args(7)
175 if (education == s"" "undefined" "") {
176     education = "null"
177 } else if (education == s"" "HE" "") {
178     education = s"" "HE Qualification" ""
179 } else if(education == s"" "A" "") {
180     education = s"" "A Level or Equivalent" ""
181 } else if (education == s"" "LowerA" "") {
182     education = s"" "Lower Than A Level" ""
183 } else if (education == s"" "Post" "") {
184     education = s"" "Post Graduate Qualification" ""
185 }
```

9 Tercera parte (fragmento 2) del código Scala

La cuarta y última parte del código de la aplicación Scala consiste en la formación de las queries a partir de los parámetros recogidos y formateados anteriormente.

En la imagen siguiente se pueden apreciar dos tipos de queries. La primera de ellas aplica los filtros que hemos obtenido por parámetros gracias a la interacción del usuario vía aplicación web y otras cuatro queries que corresponden a las cuatro posibles gráficas que podemos generar en la aplicación web. Estas últimas no aplican filtros ya que necesitan todos los datos obtenidos en dos de las vistas generadas anteriormente.

- tableChartMain: vista con todos los datos acerca de las calificaciones de los estudiantes y la información personal de dichos estudiantes.
- tableChart2: vista con todos los datos acerca de las calificaciones de los estudiantes.

Por último, nos encontramos con el envío de la query por parte de la SparkSession, el visionado de las primeras veinte columnas por consola, la escritura del resultado de la consulta en un fichero formato “.json” en el HDFS del clúster y el cierre de la sesión.

```
312 // QUERY COURSE
313
314 query = s"SELECT * FROM table WHERE (gender= $gender OR $gender is null)" +
315     s"AND (region = $region OR $region is null)" +
316     s"AND (highest_education = $education OR $education is null)" +
317     s"AND (age_band = $age OR $age is null)" +
318     s"AND (disability = $disability OR $disability is null)" +
319     s"AND (final_result = $result OR $result is null)" +
320     s"AND (date_registration $registration_symbol $registration_num OR $registration_num is null)" +
321     s"AND (sum_click $clicks_symbol $clicks_num OR $clicks_num is null)" +
322     s"AND (activity_type = $activity OR $activity is null)" +
323     s"AND (week_to = $week OR $week is null)" +
324     s"AND (module_presentation_length $length_symbol $length_num OR $length_num is null)"
325
326 } else if (filter == "chart1") {
327
328     query = s"SELECT * FROM tableChartMain" //QUERY CHART1
329
330 } else if (filter == "chart2") {
331
332     query = s"SELECT * FROM tableChart2" //QUERY CHART2
333
334 } else if (filter == "chart3") {
335
336     query = s"SELECT * FROM tableChartMain" //QUERY CHART3
337
338 } else if (filter == "chart4") {
339
340     query = s"SELECT * FROM tableChartMain" //QUERY CHART4
341
342 }
343
344 val sqlDF = spark.sql(query) //ENVIO QUERY
345
346 sqlDF.show()
347
348 sqlDF.write.json(path = "hdfs://nod01:9000/output/output.json") //ESCRITURA SALIDA QUERY EN HDFS
349
350 spark.stop()
351
352 }
```

#### 10 Cuarta parte del código Scala

### 3.4.4 REPOSITORIO CON EL CÓDIGO DE LA APLICACIÓN SCALA

Se ha habilitado un repositorio público en GitHub donde es posible consultar el código completo del paquete principal que ha sido comentado en esta sección.

Para consultar el código utilice este [enlace](#).

## 3.5 DESARROLLO DE LA APLICACIÓN WEB

### 3.5.1 INTRODUCCIÓN

Esta aplicación será la encargada de permitir al usuario interactuar con la información, permitiéndole realizar consultas aplicando los filtros que estime, visualizar el resultado de su consulta y generar gráficos acerca de determinada información que le permitirá interpretar los datos con mayor facilidad. Esta comunicación con el clúster será llevada a cabo por unos scripts que mediante el paso de determinados parámetros ejecutará la aplicación Scala que hemos visto anteriormente.

Para el desarrollo de esta aplicación se han utilizado principalmente los siguientes componentes:

- Node.js + Express.js
- HTML + Bootstrap + CSS
- Javascript + jQuery

### 3.5.2 CONFIGURACIÓN ENTORNO DE DESARROLLO

Para el desarrollo de la aplicación se ha utilizado el entorno de desarrollo WebStorm<sup>1</sup> sobre un sistema operativo Linux (CentOS) aunque las opciones para el desarrollo de una aplicación web son muchas y todas ellas válidas.

En primer lugar, descargamos el entorno de desarrollo de la web oficial y lo instalamos. Una vez tengamos el entorno de desarrollo debemos instalar los dos requisitos principales para generar la base de nuestra aplicación. Estos son Node.js y el gestor de paquetes npm.

Para descargar una versión reciente de Node.js primero tendremos que actualizar el repositorio del sistema. A continuación, los comandos necesarios.

```
yum install -y gcc-c++ make  
curl -sL https://rpm.nodesource.com/setup_10.x | sudo -E bash -  
sudo yum install nodejs  
sudo yum install npm
```

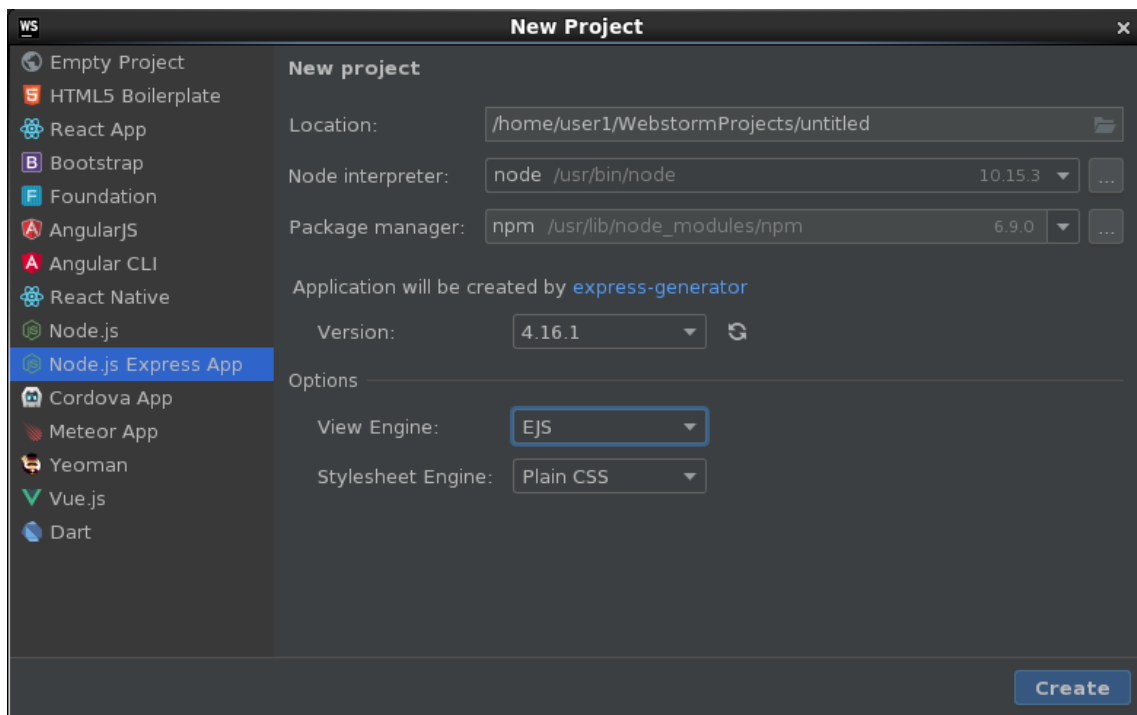
---

<sup>1</sup> Enlace descarga WebStorm [aquí](#).



Ahora que ya tenemos listos los requisitos previos a la creación de nuestro proyecto, podemos ejecutar nuestro entorno de desarrollo WebStorm para crearlo. En la ventana de creación del proyecto seleccionamos una aplicación “Node.js Express App” y nos aseguramos de escoger como “Node interpreter” y “Package manager” las versiones de Node.js y npm que hemos instalado anteriormente.

Por defecto deberíamos tener express instalado con el entorno. Además, se ha escogido EJS como plantilla para nuestro HTML. Si pulsamos el botón de crear debería generar nuestro proyecto y ya podríamos comenzar a trabajar en él.



### 11 Creación de proyecto Node.js Express App

A lo largo del desarrollo se han utilizado numerosos módulos auxiliares para solventar determinadas situaciones, estas instalaciones se han realizado siempre con el gestor de paquetes npm. A continuación, entraremos más en detalle en determinadas partes del código y se mencionarán solo los módulos de mayor importancia.

Para una mayor información acerca de los módulos usados o de la configuración establecida al servidor se podrá consultar el código completo en los repositorios habilitados que serán indicados en la siguiente sección de este capítulo.

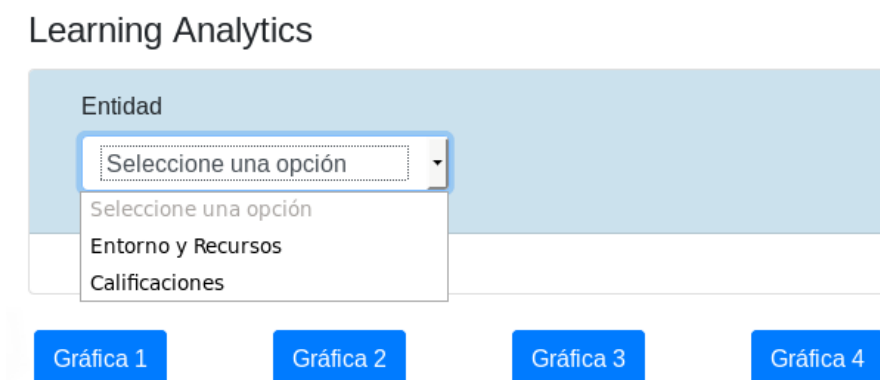
### 3.5.3 DESGLOSE DE LA APLICACIÓN

Para comentar la aplicación de una forma más cómoda y organizada vamos a dividir esta sección en tres partes, las cuales corresponden a las tres vistas principales de la aplicación web.

- Página principal. Formulario consultas y acceso a gráficas.
- Tabla resultado de la consulta.
- Gráficas generadas.

En todas las partes comenzaremos con un breve comentario de la interfaz y el estilo si fuese preciso y continuaremos comentando el backend necesario para comprender el funcionamiento de la aplicación.

En cuanto a la página principal de la aplicación, su estructura se basa en un gran formulario contenido en un contenedor proporcionado por Bootstrap que muestra unos campos u otros en función de la entidad que seleccionemos en la ventana inicial.



12 Selección formulario entidades

Se ha dividido el formulario en dos entidades para facilitar el uso de los filtros al usuario. Dichas entidades son:

- Entorno y Recursos: en este formulario encontramos los filtros referentes al entorno de aprendizaje online, los recursos contenidos en él y los alumnos
- Calificaciones: en este formulario encontramos los filtros referentes a los alumnos y sus calificaciones.

A continuación, veremos una imagen que muestra el código que genera este básico formulario inicial, el resto de las 300 líneas de código que forman la interfaz de la página principal sigue el mismo patrón, variando únicamente en el tipo de elemento en función del tipo de filtro.

```
<!-- Select Base -->
<div class="card mx-auto">
<div class="card-header" style="...">
<div class="form-group">
<label class="col-md-4 control-label" for="selectbasic">Entidad</label>
<div class="col-md-4">
<select id="selectbasic" name="selectbasic" class="form-control" style="..." onchange="changeSelect(this.value)">
<option value="none" selected disabled>Selecione una opción</option>
<option value="course">Entorno y Recursos</option>
<option value="assessment">Calificaciones</option>
</select>
</div>
</div>
</div>
</div>
```

### 13 Código HTML + Bootstrap formularios

Cabe destacar el uso de Javascript para mostrar en pantalla un tipo de formulario determinado en función de la entidad seleccionada en el formulario inicial que vimos anteriormente.

```
function changeSelect(value) {
  if (value === "assessment") {
    document.getElementById( elementId: 'assessment_filter').style.display = 'block';
    document.getElementById( elementId: 'student_filter').style.display = 'block';
    document.getElementById( elementId: 'course_filter').style.display = 'none';
  } else if (value === "course") {
    document.getElementById( elementId: 'course_filter').style.display = 'block';
    document.getElementById( elementId: 'student_filter').style.display = 'block';
    document.getElementById( elementId: 'assessment_filter').style.display = 'none';
  }

  if (value === "assessment" || value === "course") {
    document.getElementById( elementId: 'btn_div').style.display = 'block';
  } else if (value !== "assessment" || value !== "course") {
    document.getElementById( elementId: 'btn_div').style.display = 'none';
  }
}
```

### 14 Código Javascript selección formulario

Además, ha sido necesaria la utilización de Javascript para modificar el comportamiento de los “checkbox”. Ya que era necesario dar la posibilidad al usuario de seleccionar y deseleccionar ciertos elementos y al mismo tiempo no permitir que pueda seleccionar más de uno.

En las siguientes imágenes se pueden apreciar los elementos mencionados y el código Javascript con el que se ha solventado el problema.

## Learning Analytics

Entidad  
Calificaciones

Tipo de Calificaciones  
Seleccione una opción

Peso de Calificación  
 ≤20  ≥20

Puntuación  
 ≤ 0-100  ≥

15 Filtros con checkbox modificados

```
// Control de checkbox formularios
function checkboxCheck1(element) {
    if(element.checked) {
        element.classList.add("marked1");
    }else{
        element.classList.remove( tokens: "marked1");
    }
    if(document.getElementsByClassName( className: "marked1").length>1) {
        element.checked=false;
        element.classList.remove( tokens: "marked1");
    }
}
```

16 Código Javascript modificación comportamiento checkbox

Una vez el usuario ha terminado de aplicar los filtros que considere necesarios a su consulta, pulsará el botón de enviar. Este botón generará una petición POST, la cual será la encargada de recoger los filtros que ha aplicado el usuario en la interfaz y almacenarlos en distintas variables.

En caso de que el usuario haya decidido no utilizar algunos de los filtros a su disposición, dichas variables pueden tener dos posibles valores en función del tipo de filtro que se haya dejado en blanco.

Si el usuario ha dejado en blanco algún campo de texto, como por ejemplo el filtro de la puntuación (se requieren valores entre 0-100) el valor almacenado en la variable será una String vacía, lo cual nos dará problemas más adelante en el paso de parámetros al script que ejecutará la aplicación Scala. Por tanto, modificaremos ese valor con un “undefined” antes de enviar el parámetro.

En la imagen siguiente podemos ver el código Javascript que se encarga de realizar el proceso comentado anteriormente y en función del filtro principal que indica la entidad, ejecutará el script enviando unas variables u otras.

```
router.post('/table', function(req, res, next) {

  var filter = req.body.selectbasic;

  var assessment_type = req.body.assessment_type;
  var weight = req.body.weight;
  var score = req.body.score;
  var score_num = req.body.score_num;
  if (score_num === "") score_num = "undefined";
  var gender = req.body.gender;
  var region = req.body.region;
  var education = req.body.education;
  var age = req.body.age;
  var disability = req.body.disability;
  var result = req.body.result;

  var registration = req.body.registration_check;
  var registration_num = req.body.registration_num;
  if (registration_num === "") registration_num = "undefined";
  var clicks = req.body.clicks;
  var clicks_num = req.body.clicks_num;
  if (clicks_num === "") clicks_num = "undefined";
  var activity = req.body.activity;
  var week = req.body.week;
  if (week === "") week = "undefined";
  var length = req.body.length;
  var length_num = req.body.length_num;
  if (length_num === "") length_num = "undefined";

  if (filter === "assessment") {
    //console.log("ASSESSMENT TEST " + shell.exec('./public/shell/test.sh ' + filter + ' ' + assessment_type + ' ' + weight + ' ' + score + ' ' + score_num
    // + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability + ' ' + result));
    console.log(" [SPARK SUBMIT]" + shell.exec( $HUNG: './public/shell/submit.sh ' + filter + ' ' + assessment_type + ' ' + weight + ' ' + score + ' ' + score_num
    + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability + ' ' + result));
    console.log(" [SPARK GET]" + shell.exec( $HUNG: './public/shell/getData.sh'));
  } else if (filter === "course") {
    //console.log("COURSE TEST " + shell.exec('./public/shell/test.sh ' + filter + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability
    // + ' ' + result + ' ' + registration + ' ' + registration_num + ' ' + clicks + ' ' + clicks_num + ' ' + activity + ' ' + week + ' ' + length + ' ' + length_num));
    console.log(" [SPARK SUBMIT]" + shell.exec( $HUNG: './public/shell/submit.sh ' + filter + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability
    + ' ' + result + ' ' + registration + ' ' + registration_num + ' ' + clicks + ' ' + clicks_num + ' ' + activity + ' ' + week + ' ' + length + ' ' + length_num));
    console.log(" [SPARK GET]" + shell.exec( $HUNG: './public/shell/getData.sh'));
  }

  res.render('../views/table.ejs', { title: 'Learning Analytics App' });
}
```

### 17 Código Javascript POST formulario

La ejecución de dichos scripts, su contenido y el módulo utilizado para hacer posible dicha ejecución desde la aplicación web serán comentados con mayor detenimiento al [final](#) de este capítulo.

Lo siguiente que el usuario verá tras enviar la consulta será una ligera animación simulando un reloj indicando que se está procesando su consulta, esto se ha añadido debido a que en función de los filtros utilizados y los valores de estos la obtención de respuesta puede demorarse unos minutos.

Hay que recordar que en cada consulta se realizará la unión de todas las tablas (por si hubiese algún cambio en ellas), la subida de un fichero “.json” que puede llegar a pesar

varios gigas al HDFS del clúster y por último la descarga de dicho fichero para su uso en la aplicación web.

Una vez se ha completado este proceso el usuario podrá ver el resultado de su consulta en una tabla generada y rellenada a partir del fichero con extensión “.json” como la mostrada a continuación.

#### Learning Analytics

id_student	code_module	code_presentation	id_assessment	assessment_type	weight	score	gender	region	highest_education	age_band	disability	final_result
11391	AAA	2013J	1752	TMA	10	78	M	East Anglian Region	HE Qualification	55<=	N	Pass
28400	AAA	2013J	1752	TMA	10	70	F	Scotland	HE Qualification	35-55	N	Pass
31604	AAA	2013J	1752	TMA	10	72	F	South East Region	A Level or Equivalent	35-55	N	Pass
32885	AAA	2013J	1752	TMA	10	69	F	West Midlands Region	Lower Than A Level	0-35	N	Pass
38053	AAA	2013J	1752	TMA	10	79	M	Wales	A Level or Equivalent	35-55	N	Pass
45462	AAA	2013J	1752	TMA	10	70	M	Scotland	HE Qualification	0-35	N	Pass
45462	AAA	2013J	1752	TMA	10	72	F	North Western Region	A Level or Equivalent	0-35	N	Pass
52130	AAA	2013J	1752	TMA	10	72	F	East Anglian Region	A Level or Equivalent	0-35	N	Pass
53025	AAA	2013J	1752	TMA	10	71	M	North Region	Post Graduate Qualification	55<=	N	Pass
57506	AAA	2013J	1752	TMA	10	68	M	South Region	Lower Than A Level	35-55	N	Pass
58873	AAA	2013J	1752	TMA	10	73	F	East Anglian Region	A Level or Equivalent	0-35	N	Pass
59185	AAA	2013J	1752	TMA	10	67	M	East Anglian Region	Lower Than A Level	35-55	N	Pass
62155	AAA	2013J	1752	TMA	10	73	F	North Western Region	HE Qualification	0-35	N	Pass
63400	AAA	2013J	1752	TMA	10	83	M	Scotland	Lower Than A Level	35-55	N	Pass
65002	AAA	2013J	1752	TMA	10	66	F	East Anglian Region	A Level or Equivalent	0-35	N	Withdrawn
70464	AAA	2013J	1752	TMA	10	59	F	West Midlands Region	A Level or Equivalent	35-55	N	Pass
71361	AAA	2013J	1752	TMA	10	82	M	Ireland	HE Qualification	35-55	N	Pass
74372	AAA	2013J	1752	TMA	10	60	M	East Anglian Region	A Level or Equivalent	35-55	N	Fail
75091	AAA	2013J	1752	TMA	10	67	M	South West Region	A Level or Equivalent	35-55	N	Pass
77367	AAA	2013J	1752	TMA	10	73	M	East Midlands Region	A Level or Equivalent	0-35	N	Pass
91265	AAA	2013J	1752	TMA	10	75	M	North Western Region	HE Qualification	0-35	N	Pass
94961	AAA	2013J	1752	TMA	10	74	M	South Region	Lower Than A Level	35-55	N	Withdrawn
98094	AAA	2013J	1752	TMA	10	62	M	Wales	Lower Than A Level	35-55	N	Pass
100893	AAA	2013J	1752	TMA	10	63	M	Yorkshire Region	A Level or Equivalent	0-35	N	Pass
101781	AAA	2013J	1752	TMA	10	84	M	South Region	Lower Than A Level	35-55	N	Pass

18 Tabla con resultados de la consulta

Para generar la tabla con los resultados a partir del fichero “.json” descargado del clúster se ha hecho uso de la biblioteca jQuery<sup>1</sup>, principalmente del método getJSON() que permite cargar un fichero “.json” mediante la ejecución de una petición HTTP GET al directorio donde hemos descargado el fichero. Posteriormente se almacenará el contenido del fichero en un array que hemos limitado a 500 elementos por problemas de memoria en el navegador para consultas que generan ficheros muy extensos.

Recordar que el proceso de descarga y formateado del fichero “.json” se comentará al final de este capítulo cuando se detalle el contenido de los scripts.

```

$(document).ready(function () {
    $.getJSON("../exit/data.json", function (data) {

        var jsonItems = []; // Array para JSON
        $.each(data, function (index, value) {
            jsonItems.push(value);
            return index<499;
        });
    });
}

```

19 Código Javascript carga fichero JSON

<sup>1</sup> Enlace descarga jQuery [aquí](#).

Una vez hemos cargado el contenido del fichero en un array podemos proceder a generar y rellenar la tabla resultado. En primer lugar, extraemos la cabecera de la tabla del array para almacenarlo en otra variable. A continuación, generamos la tabla, insertamos la primera fila y cargamos en dicha fila la cabecera extraída anteriormente celda a celda.

```
// Extrae header de la tabla
var col = [];
for (var i = 0; i < jsonItems.length; i++) {
    for (var j in jsonItems[i]) {
        if (col.indexOf(j) === -1) {
            col.push(j);
        }
    }
}

var table = document.createElement( tagName: "table");

// Crea el header de la tabla con los datos extraídos anteriormente.
var tr = table.insertRow(-1);

for (var i = 0; i < col.length; i++) {
    var th = document.createElement( tagName: "th");
    th.innerHTML = col[i];
    tr.appendChild(th);
}
```

#### 20 Código Javascript para carga cabecera

Para finalizar realizaremos un proceso similar al anterior para generar el resto de la tabla. Añadiendo fila por fila y rellenando las celdas una a una a partir del array inicial donde cargamos el fichero “.json”. Una vez se ha terminado de rellenar la tabla la insertamos a un nodo padre vacío que hemos especificado en el HTML de la vista.

```
// Rellena la tabla por filas
for (var i = 0; i < jsonItems.length; i++) {

    tr = table.insertRow(-1);

    for (var j = 0; j < col.length; j++) {
        var tabCell = tr.insertCell( index: -1);
        tabCell.innerHTML = jsonItems[i][col[j]];
    }
}

var divContainer = document.getElementById( elementId: "jsonTable");
divContainer.innerHTML = "";
divContainer.appendChild(table);
```

#### 21 Código Javascript carga tabla completa

A continuación, vamos a analizar los distintos gráficos que el usuario podrá generar desde la página principal que vimos anteriormente. Los datos de estudio que forman estos gráficos han sido discutidos previamente con un docente con el fin de aportar información útil para el usuario objetivo de este proyecto.

Estas gráficas han sido generadas gracias a la librería Chart.js<sup>1</sup> a pesar de que el planteamiento inicial incluía la librería D3.js. Este cambio se debe a que la gran potencia y complejidad de la librería D3.js parecía desaprovecharse para el tipo de gráficas que necesitábamos. Además, no se adaptaba bien al uso de fichero JSON. Este cambio fue consultado y aceptado por el tutor de este proyecto.

Cada vez que generamos una gráfica se realiza una nueva consulta al clúster de forma que los datos se encuentre siempre actualizados. Para ello se ejecuta el script que analizaremos más adelante indicándole como primer parámetro de que gráfica se trata. Las consultas que generan las gráficas no requieren de filtros puesto que necesitaremos descargar toda la información como vimos [anteriormente](#) cuando estudiamos la aplicación Scala.

```
// POST GRÁFICA 1
router.post('/chart1', function(req, res, next) {
  //console.log("CHART1 TEST" + shell.exec('./public/shell/test.sh ' + "chart1"));
  console.log(" [SPARK SUBMIT CHART1]" + shell.exec( string: './public/shell/submit.sh '+ "chart1"));
  console.log(" [SPARK GET] " + shell.exec( string: './public/shell/getData.sh'));

  res.render('../views/chart1.ejs', { title: 'Learning Analytics App' });
});

// POST GRÁFICA 2
router.post('/chart2', function(req, res, next) {
  //console.log("CHART2 TEST" + shell.exec('./public/shell/test.sh ' + "chart2"));
  console.log(" [SPARK SUBMIT CHART1]" + shell.exec( string: './public/shell/submit.sh '+ "chart2"));
  console.log(" [SPARK GET] " + shell.exec( string: './public/shell/getData.sh'));

  res.render('../views/chart2.ejs', { title: 'Learning Analytics App' });
});

// POST GRÁFICA 3
router.post('/chart3', function(req, res, next) {
  //console.log("CHART3 TEST" + shell.exec('./public/shell/test.sh ' + "chart3"));
  console.log(" [SPARK SUBMIT CHART1]" + shell.exec( string: './public/shell/submit.sh '+ "chart3"));
  console.log(" [SPARK GET] " + shell.exec( string: './public/shell/getData.sh'));

  res.render('../views/chart3.ejs', { title: 'Learning Analytics App' });
});

// POST GRÁFICA 3
router.post('/chart4', function(req, res, next) {
  //console.log("CHART4 TEST" + shell.exec('./public/shell/test.sh ' + "chart4"));
  console.log(" [SPARK SUBMIT CHART1]" + shell.exec( string: './public/shell/submit.sh '+ "chart4"));
  console.log(" [SPARK GET] " + shell.exec( string: './public/shell/getData.sh'));

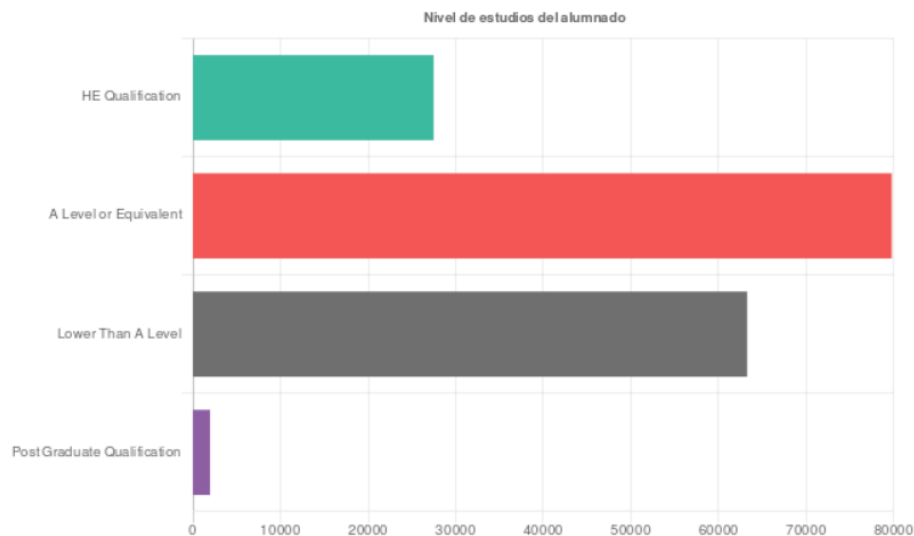
  res.render('../views/chart4.ejs', { title: 'Learning Analytics App' });
});
```

## 22 Código Javascript POST gráficas

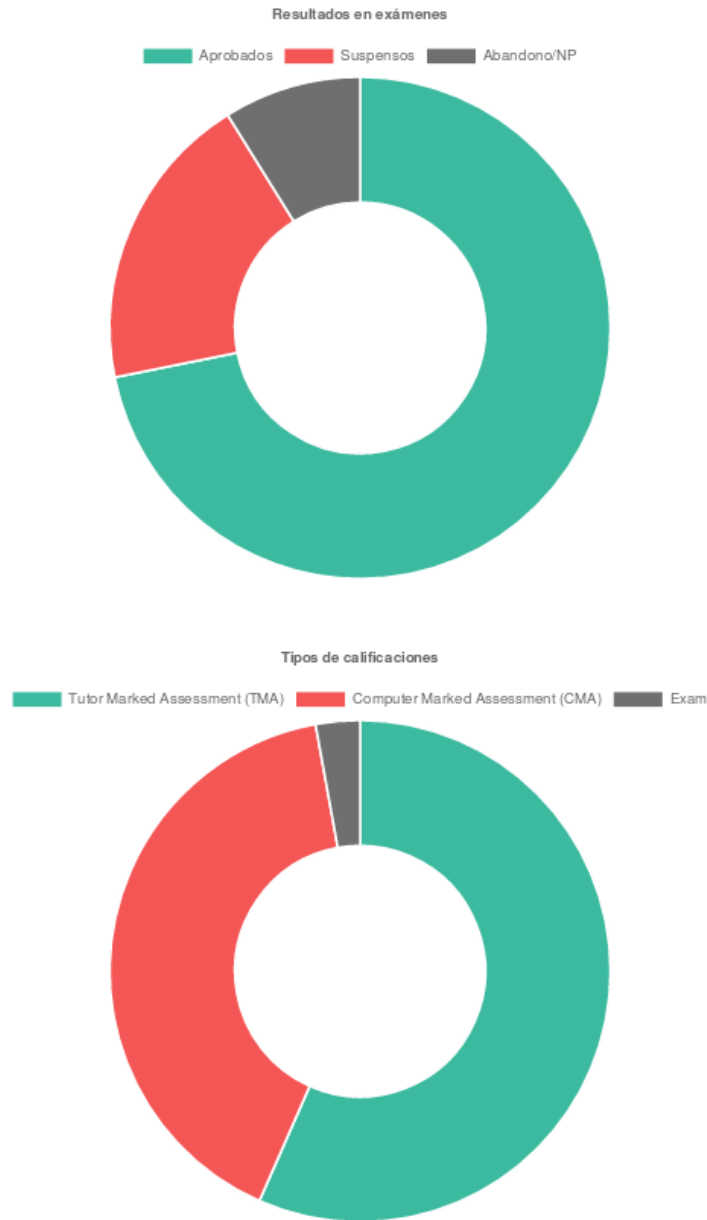


La primera gráfica la forman en realidad tres gráficas distintas. Todas ellas se basan en la misma consulta, pero muestran aspectos distintos de la misma. Estas tres gráficas representan la siguiente información.

- Resultados generales en los exámenes
  - Aprobados
  - Suspensos
  - No presentados/Abandonos
- Tipos de calificaciones utilizadas por los distintos cursos
  - Tutor Marked Assessment (TMA)
  - Computer Marked Assessment (CMA)
  - Exam
- Nivel de estudios del alumnado matriculado
  - HE Qualification
  - A Level or Equivalent
  - Lower Than A Level
  - Post Graduate Qualification



23 Gráfica 1 - Nivel estudios alumnado. [Ver Anexo 1](#)



24 Gráfica 1 - Resultados exámenes y tipos de calificaciones. [Ver Anexo 1](#)

Para generar las gráficas a partir del fichero “.json” descargado del clúster se ha seguido el mismo procedimiento inicial que con la tabla de resultado con el uso de la biblioteca jQuery<sup>2</sup>, principalmente del método getJSON() que permite cargar un fichero “.json” mediante la ejecución de una petición HTTP GET al directorio donde hemos descargado el fichero. Posteriormente se almacenará el contenido del fichero en un array.

<sup>2</sup> Enlace web oficial Chart.js [aquí](#).

<sup>2</sup> Enlace descarga jQuery [aquí](#).

A dicho array se le aplicará la función reduce con el fin de agrupar en otra variable de tipo array los resultados para cada una de los posibles valores del campo. Es decir, para el caso de los resultados de los exámenes tendremos un array que contendrá otros tres arrays, estos tres subarrays almacenarán todas las entradas “Pass”, “Fail” y “Withdrawn” que se encontraron en el fichero JSON.

Este procedimiento se llevará a cabo para las tres gráficas.

```

$(document).ready(function () {
    $.getJSON("../exit/data.json", function (d) {

        var jsonfile = []; // Array para JSON
        $.each(d, function (index, value) {
            jsonfile.push(value);
        });

        var count1 = jsonfile.reduce((acc, cur) => cur.final_result === "Pass" ? ++acc : acc, 0);
        var count2 = jsonfile.reduce((acc, cur) => cur.final_result === "Fail" ? ++acc : acc, 0);
        var count3 = jsonfile.reduce((acc, cur) => cur.final_result === "Withdrawn" ? ++acc : acc, 0);
        var data1 = [count1, count2, count3];

        var count4 = jsonfile.reduce((acc, cur) => cur.assessment_type === "TMA" ? ++acc : acc, 0);
        var count5 = jsonfile.reduce((acc, cur) => cur.assessment_type === "CMA" ? ++acc : acc, 0);
        var count6 = jsonfile.reduce((acc, cur) => cur.assessment_type === "Exam" ? ++acc : acc, 0);
        var data2 = [count4, count5, count6];

        var count7 = jsonfile.reduce((acc, cur) => cur.highest_education === "HE Qualification" ? ++acc : acc, 0);
        var count8 = jsonfile.reduce((acc, cur) => cur.highest_education === "A Level or Equivalent" ? ++acc : acc, 0);
        var count9 = jsonfile.reduce((acc, cur) => cur.highest_education === "Lower Than A Level" ? ++acc : acc, 0);
        var count10 = jsonfile.reduce((acc, cur) => cur.highest_education === "Post Graduate Qualification" ? ++acc : acc, 0);
        var data3 = [count7, count8, count9, count10];
    });
}

```

#### 25 Código Javascript valores gráfica 1

Una vez tenemos los valores a mostrar almacenados, es hora de generar la gráfica. El proceso de creación de la gráfica es muy similar para las tres que componen este primer apartado y por tanto solo veremos una de ellas. Además de los valores y la etiquetas el único cambio significativo es el tipo de gráfica (doughnut para las dos primeras y horizontalBar para la tercera). Esta simplicidad a la hora de generar las gráficas es una de las características más destacadas de la librería Chart.js.

```

var ctx = document.getElementById( "elementId: "canvas1").getContext('2d');
var config = {
    type: 'doughnut',
    data: {
        labels: ["Aprobados", "Suspensos", "Abandono/NP"],
        datasets: [{
            label: 'Graph Line',
            data: data1,
            backgroundColor: ["#3cba9f", "#f45555", "#706f6f"]
        }]
    },
    options: {
        title: {
            display: true,
            text: 'Resultados en exámenes'
        },
        responsive: true,
        maintainAspectRatio: false,
        animation: {
            animateScale: true,
        }
    }
};
var chart1 = new Chart( ctx, config);

```

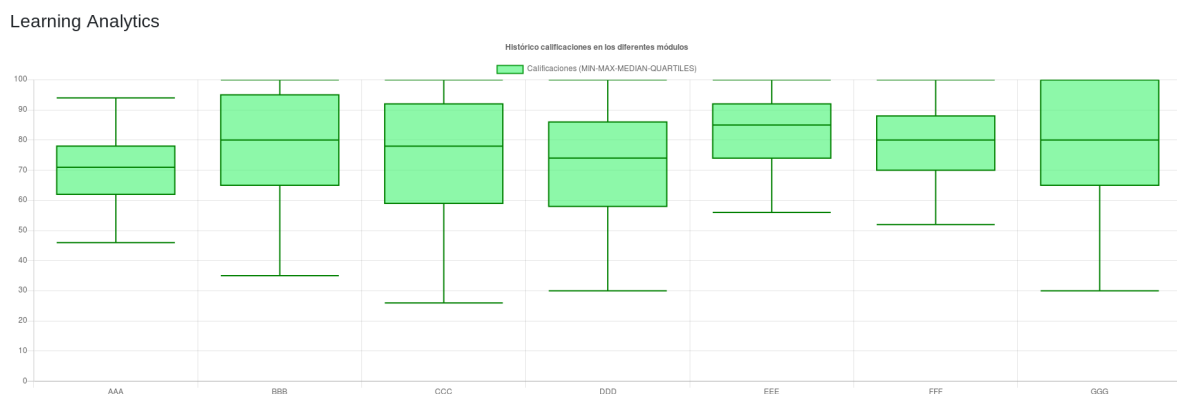
#### 26 Código Javascript generador de gráficas Chart.js

Las siguientes tres gráficas son ligeramente más complejas y en este caso la librería base de Chart.js no permitía construir las gráficas que buscábamos. Sin embargo, existe una librería adicional denominada “Chart.js Box Plot addon”<sup>1</sup> que nos permitirá construir diagramas de caja y de violín.

La instalación es realmente sencilla y podemos llevarla a cabo desde el gestor de paquetes npm.

```
npm install --save chart.js chartjs-chart-box-and-violin-plot
```

Una vez instalada la librería adicional, podemos construir la segunda de las gráficas. Este gráfico genera un diagrama de cajas y bigotes que recoge un seguimiento histórico de las puntuaciones de los alumnos en los distintos cursos que se imparten utilizando los datos de todas las presentaciones disponibles en el *dataset*. Pudiendo evaluar así datos como la media, los mínimos y máximos alcanzados y los cuartiles.



27 Gráfica 2 - Histórico calificaciones en los distintos módulos. [Ver Anexo 2](#)

Para generar la gráfica a partir del fichero “.json” descargado del clúster se ha seguido el mismo procedimiento inicial que con las gráficas anteriores, haciendo uso de la biblioteca jQuery, principalmente del método `getJSON()` que permite cargar un fichero “.json” mediante la ejecución de una petición HTTP GET al directorio donde hemos descargado el fichero. Posteriormente se almacenará el contenido del fichero en un array.

El tratamiento de dicho array para extraer los datos necesarios es más complejo que el visto anteriormente ya que no basta con saber el número de apariciones de un determinado valor, en este caso necesitamos todos los resultados académicos separados por módulos con el fin de poder tratar los datos matemáticamente para representarlos como vimos anteriormente.

Para extraer las calificaciones sin alterar ningún valor y diferenciar por los distintos módulos se ha recorrido el array con el método “forEach” preguntando por el nombre de los distintos cursos. En función de a qué curso pertenezca el dato de la calificación del alumno se almacenará en un array por separado. A estos arrays se les aplicará el método “map” con el objetivo de crear un nuevo array de tipo “Number” ya que anteriormente el método “push” utilizado al recorrer el array principal almacenaba los valores como una “String”.

```
$(document).ready(function () {
    $.getJSON("../exit/data.json", function (d) {

        var jsonfile = []; // Array para JSON
        $.each(d, function (index, value) {
            jsonfile.push(value);
        });

        var d1 = [], d2 = [], d3 = [], d4 = [], d5 = [], d6 = [], d7 = [];

        jsonfile.forEach( callbackfn: function(json) {
            if (json.code_module === "AAA") {
                d1.push(json.score);
            } else if (json.code_module === "BBB"){
                d2.push(json.score);
            } else if (json.code_module === "CCC"){
                d3.push(json.score);
            } else if (json.code_module === "DDD"){
                d4.push(json.score);
            } else if (json.code_module === "EEE"){
                d5.push(json.score);
            } else if (json.code_module === "FFF"){
                d6.push(json.score);
            } else if (json.code_module === "GGG"){
                d7.push(json.score);
            }
        });

        // String to Number
        var data1 = d1.map(Number);
        var data2 = d2.map(Number);
        var data3 = d3.map(Number);
        var data4 = d4.map(Number);
        var data5 = d5.map(Number);
        var data6 = d6.map(Number);
        var data7 = d7.map(Number);
    });
});
```

## 28 Código Javascript valores gráfica 2

La construcción de la gráfica es similar a las gráficas anteriores, sin embargo, esta vez y gracias a la nueva librería instalada podemos especificar que queremos generar una gráfica de tipo “Boxplot”, incluyendo como siempre las etiquetas y los arrays con los datos que hemos extraído anteriormente, pero esta vez en una variable externa (“boxplotdata”) a la variable con la configuración de la gráfica a generar (“config”).

Dejamos el código a continuación para mayor claridad. Se recomienda comparar con el constructor de la gráfica anterior para apreciar la diferencia en las variables mencionadas.

```

const boxplotData = {
  labels: ['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF', 'GGG'],
  datasets: [{
    label: 'Calificaciones (MIN-MAX-MEDIAN-QUARTILES)',
    backgroundColor: 'rgba(66,244,113,0.6)',
    borderColor: 'green',
    borderWidth: 2,
    outlierColor: '#000000',
    outlierRadius: 0,
    padding: 10,
    itemRadius: 0,
    data: [data1, data2, data3, data4, data5, data6, data7]
  }]
};

const ctx = document.getElementById("canvas2").getContext("2d");
var config = {
  type: 'boxplot',
  data: boxplotData,
  options: {
    responsive: true,
    maintainAspectRatio: false,
    legend: {
      position: 'top',
    },
    title: {
      display: true,
      text: 'Histórico calificaciones en los diferentes módulos'
    }
  }
};
var chart = new Chart(ctx, config)

```

29 Código Javascript generador de gráfica 2 (Chart.js + addon)

Para finalizar con el apartado de las gráficas analizaremos las dos últimas que, al igual que la anterior, existía la problemática de que Chart.js no permitía construir las gráficas que buscábamos. Sin embargo, existe una librería adicional denominada “Chart.js Box Plot addon”<sup>1</sup> que nos permitirá construir los diagramas de violín.

La instalación es realmente sencilla y podemos llevarla a cabo desde el gestor de paquetes npm como vimos anteriormente.

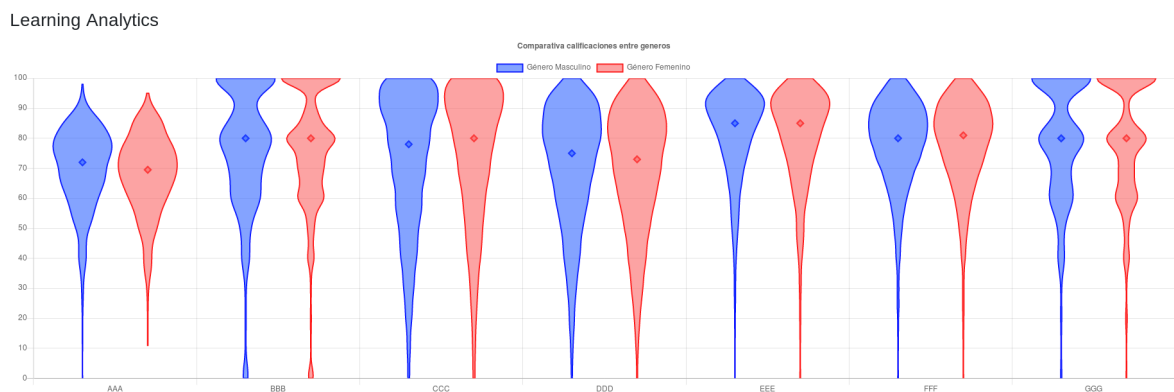
```
npm install --save chart.js chartjs-chart-box-and-violin-plot
```

Suponiendo que ya tenemos lista la librería adicional podemos comenzar a construir los dos diagramas de violín. Estos diagramas son una combinación de los diagramas de caja y los diagramas de densidad que se giran y se colocan a cada lado para mostrar la distribución de los datos de estudio. Además, disponemos de un punto central que indica la media.

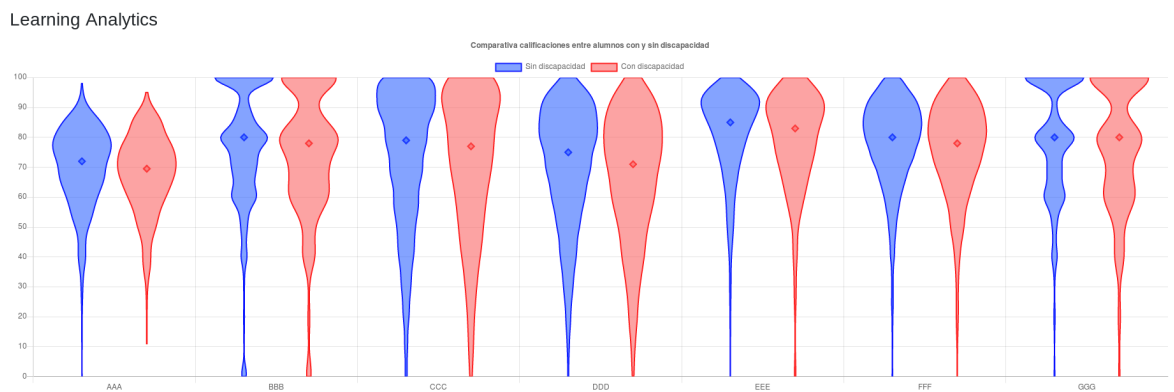
Estos dos últimos diagramas son muy similares entre sí, por lo que, aunque hablaremos de ambos únicamente mostraremos el código de una de ellas ya que solo difieren en las etiquetas y los conjuntos de datos, aunque estos se obtienen de la misma forma. Si desean consultar el código completo al final de este capítulo tendrán disponible el enlace al repositorio.

El objetivo de estos diagramas fue, al igual que las gráficas anteriores, discutido con un docente. Con ellos se pretende estudiar todas las calificaciones de los alumnos separados por cursos con la peculiaridad de que dividiremos el muestreo por sexo y por discapacidad. Con ello se pretende estudiar si existe algún tipo de discriminación o dificultad apreciable en las calificaciones de los alumnos en función de su sexo o de si sufren algún tipo de discapacidad.

Cabe destacar que en las pruebas que se llevaron a cabo en este proyecto no se apreciaron diferencias destacables entre sexos, aunque es cierto que la media de las notas de los alumnos con algún tipo de discapacidad siempre es ligeramente menor que la de los alumnos sin discapacidad en todos los cursos de los que disponemos de información.



30 Gráfica 3 - Comparativa calificaciones entre géneros. [Ver Anexo 3](#)



31 Gráfica 4 - Comparativa calificaciones entre alumnos con/sin discapacidad. [Ver Anexo 4](#)

Para generar la gráfica a partir del fichero “.json” descargado del clúster se ha seguido el mismo procedimiento inicial que con las gráficas anteriores, haciendo uso de la biblioteca jQuery, principalmente del método getJSON(). Posteriormente se almacenará el contenido del fichero en un array.

El tratamiento de dicho array para extraer los datos es el mismo que el visto anteriormente, pero duplicando el procedimiento ya que necesitamos todos los resultados académicos separados por módulos y a su vez por sexo/discapacidad con el fin de poder representar los datos tal y como vimos en los diagramas de violín anteriores.

Para extraer las calificaciones sin alterar ningún valor y diferenciar por los distintos módulos se ha recorrido el array con el método “forEach” preguntando por el nombre de los distintos cursos. En función de a qué curso pertenezca el dato de la calificación del alumno se almacenará en un array por separado siempre asegurándonos de que pertenece a un mismo sexo o nivel de discapacidad. Esta operación se repetirá al completo para llenar el array con los datos de género/discapacidad opuestos.

A todos estos arrays se les aplicará el método “map” con el objetivo de crear nuevos arrays de tipo “Number” ya que anteriormente el método “push” utilizado al recorrer el array principal almacenaba los valores como una “String”.

```
jsonfile.forEach( callbackfn: function(json) {
  if (json.code_module === "AAA" && json.gender === "M") {
    d1.push(json.score);
  } else if (json.code_module === "BBB" && json.gender === "M"){
    d2.push(json.score);
  } else if (json.code_module === "CCC" && json.gender === "M"){
    d3.push(json.score);
  } else if (json.code_module === "DDD" && json.gender === "M"){
    d4.push(json.score);
  } else if (json.code_module === "EEE" && json.gender === "M"){
    d5.push(json.score);
  } else if (json.code_module === "FFF" && json.gender === "M"){
    d6.push(json.score);
  } else if (json.code_module === "GGG" && json.gender === "M"){
    d7.push(json.score);
  }
});

jsonfile.forEach( callbackfn: function(json) {
  if (json.code_module === "AAA" && json.gender === "F") {
    d8.push(json.score);
  } else if (json.code_module === "BBB" && json.gender === "F"){
    d9.push(json.score);
  } else if (json.code_module === "CCC" && json.gender === "F"){
    d10.push(json.score);
  } else if (json.code_module === "DDD" && json.gender === "F"){
    d11.push(json.score);
  } else if (json.code_module === "EEE" && json.gender === "F"){
    d12.push(json.score);
  } else if (json.code_module === "FFF" && json.gender === "F"){
    d13.push(json.score);
  } else if (json.code_module === "GGG" && json.gender === "F"){
    d14.push(json.score);
  }
});
```



```

// String to Number

var data1 = d1.map(Number);
var data2 = d2.map(Number);
var data3 = d3.map(Number);
var data4 = d4.map(Number);
var data5 = d5.map(Number);
var data6 = d6.map(Number);
var data7 = d7.map(Number);
var data8 = d8.map(Number);
var data9 = d9.map(Number);
var data10 = d10.map(Number);
var data11 = d11.map(Number);
var data12 = d12.map(Number);
var data13 = d13.map(Number);
var data14 = d14.map(Number);

```

### 33 Código Javascript valores gráfica 3/4 (parte 2)

La construcción de la gráfica es similar a la gráfica anterior, sin embargo, esta vez y gracias a la nueva librería instalada podemos especificar que queremos generar una gráfica de tipo “Violin”, incluyendo como siempre las etiquetas y los arrays con los datos que hemos extraído anteriormente.

```

const boxplotData = {
  labels: ['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF', 'GGG'],
  datasets: [
    {
      label: 'Género Masculino',
      backgroundColor: 'rgba(51,102,255,0.6)',
      borderColor: 'blue',
      borderWidth: 2,
      outlierColor: '#000000',
      outlierRadius: 0,
      padding: 10,
      data: [data1, data2, data3, data4, data5, data6, data7]
    },
    {
      label: 'Género Femenino',
      backgroundColor: 'rgba(250,102,102,0.6)',
      borderColor: 'red',
      borderWidth: 2,
      outlierColor: '#000000',
      outlierRadius: 0,
      padding: 10,
      data: [data8, data9, data10, data11, data12, data13, data14]
    }
  ]
};

const ctx = document.getElementById( elementId: "canvas3").getContext("2d");
var config = {
  type: 'violin',
  data: boxplotData,
  options: {
    responsive: true,
    maintainAspectRatio: false,
    legend: {
      position: 'top',
    },
    title: {
      display: true,
      text: 'Comparativa calificaciones entre generos'
    }
  }
};
var chart = new Chart(ctx, config)

```

### 34 Código Javascript generador de gráficas 3/4 (Chart.js + addon)

## 3.5.4 ANÁLISIS BASH SCRIPTS

Tal y como hemos comentado anteriormente, la aplicación web sirve como interfaz para que el usuario pueda personalizar su consulta al clúster. Los parámetros que forman la consulta deben ser transmitidos al programa Scala que comentamos [anteriormente](#). Para ello primero deben ser cargados y almacenados en variables por la aplicación web mediante el uso de Javascript para luego ser enviados como parámetros al ejecutar los scripts que veremos a continuación.

A continuación, vemos como haciendo uso del paquete shelljs<sup>1</sup> podemos ejecutar comandos o en nuestro caso, ejecutar un script desde la aplicación web. A dicho script debemos enviar todos los parámetros cargados del formulario siendo el más importante el primero de todos, “filter”, el cual nos indica la entidad o gráfica sobre la que hacer la consulta.

```
if (filter === "assessment") {
  //console.log("ASSESSMENT TEST " + shell.exec('./public/shell/test.sh ' + filter + ' ' + assessment_type + ' ' + weight + ' ' + score + ' ' + score_num
  // + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability + ' ' + result));
  console.log(" [SPARK SUBMIT]" + shell.exec( string: './public/shell/submit.sh ' + filter + ' ' + assessment_type + ' ' + weight + ' ' + score + ' ' + score_num
  + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability + ' ' + result));
  console.log(" [SPARK GET]" + shell.exec( string: './public/shell/getData.sh'));
} else if (filter === "course") {
  //console.log("COURSE TEST " + shell.exec('./public/shell/test.sh ' + filter + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability
  // + ' ' + result + ' ' + registration + ' ' + registration_num + ' ' + clicks + ' ' + clicks_num + ' ' + activity + ' ' + week + ' ' + length + ' ' + length_num));
  console.log(" [SPARK SUBMIT]" + shell.exec( string: './public/shell/submit.sh ' + filter + ' ' + gender + ' ' + region + ' ' + education + ' ' + age + ' ' + disability
  + ' ' + result + ' ' + registration + ' ' + registration_num + ' ' + clicks + ' ' + clicks_num + ' ' + activity + ' ' + week + ' ' + length + ' ' + length_num));
  console.log(" [SPARK GET]" + shell.exec( string: './public/shell/getData.sh'));
}
```

El script se encargará de, en función del contenido del parámetro mencionado antes (“filter”), asignar unas variables determinadas y enviarlas de nuevo mediante la ejecución del comando “spark-submit”, el cual ejecutará la aplicación Scala con los parámetros que le hemos indicado.

Destacar que el envío de algunos parámetros se realiza siguiendo el siguiente patrón:

- “”\$parameter””

El motivo es que los parámetros compuestos por texto deben incluir comillas simples para que la consulta en la aplicación Scala reconozca el valor como una “String”. En los parámetros compuestos por valores numéricos no es necesario formatearlos de esta forma y por tanto son enviados directamente.

- \$parameter

---

<sup>2</sup> Enlace paquete ShellJS [aquí](#).

El script comenzará siempre eliminando los datos de la anterior consulta (si la hubiera) del clúster para asegurarnos así que el fichero que descargaremos posteriormente es el más reciente.

```
#!/bin/bash

echo "Ejecutando código scala"
filter=$1

hdfs dfs -rm -R /output/output.json

if [ $1 == "assessment" ]
then

    assessment_type=$2
    weight=$3
    score=$4
    score_num=$5
    gender=$6
    region=$7
    education=$8
    age=$9
    disability=${10}
    result=${11}

    spark-submit --class myproject.Main ./public/jar/scalaapp_2.11-0.1.jar $filter ""$assessment_type"" $weight $score $score_

elif [ $1 == "course" ]
then

    gender=$2
    region=$3
    education=$4
    age=$5
    disability=$6
    result=$7
    registration=$8
    registration_num=$9
    clicks=${10}
    clicks_num=${11}
    activity=${12}
    week=${13}
    length=${14}
    length_num=${15}

    spark-submit --class myproject.Main ./public/jar/scalaapp_2.11-0.1.jar $filter ""$gender"" ""$region"" ""$education""
```

35 Código script spark-submit consultas

En caso de que la consulta sea para la generación de una gráfica el único parámetro necesario tal y como explicamos anteriormente es el primero (“\$filter”) que indicará al programa Scala de qué tipo de gráfica para que realice la query pertinente. Por tanto, el resto de parámetros serán enviados como “undefined”.

```
elif [ $1 == "chart1" ]
then

    spark-submit --class myproject.Main ./public/jar/scalaapp_2.11-0.1.jar $filter ""Exam"" undefined undefined undefined ""undefined"" ""undefined"" ""Pos

elif [ $1 == "chart2" ]
then

    spark-submit --class myproject.Main ./public/jar/scalaapp_2.11-0.1.jar $filter ""undefined"" undefined undefined undefined ""undefined"" ""undefined""

elif [ $1 == "chart3" ]
then

    spark-submit --class myproject.Main ./public/jar/scalaapp_2.11-0.1.jar $filter ""undefined"" undefined undefined undefined ""undefined"" ""undefined""

elif [ $1 == "chart4" ]
then

    spark-submit --class myproject.Main ./public/jar/scalaapp_2.11-0.1.jar $filter ""undefined"" undefined undefined undefined ""undefined"" ""undefined""

fi
```

36 Código script spark-submit gráficas

Una vez el proceso de consulta se ha completado, es momento de descargar el fichero con extensión “.json” con el fin de que la aplicación web pueda hacer uso de los datos resultado de la consulta. Para ello dispondremos de otro script que se ejecutará seguidamente del descrito anteriormente.

Este script descargará el fichero “.json” resultado de la consulta en el directorio especificado. Dado que la descarga del fichero se produce en bloques, tendremos que unificar todos los fichero resultado en un único fichero “.json” y eliminar la carpeta que contenía dichos bloques.

Seguidamente es momento de dar formato a dicho fichero “.json”, para ello haremos uso del comando Linux “sed” que nos permitirá editar el fichero al instante gracias a las siguientes expresiones regulares:

- ‘**1s/{/{/**’: Esta expresión buscará y sustituirá la primera llave “{” de la primera línea por un corchete seguido de la llave anterior “[{”.
- ‘**s/}/},/g**’: Esta expresión buscará y sustituirá la última llave “}” de cada línea por una llave seguida de una coma “},”.
- ‘**\$s/},/}]/**’: Esta expresión buscará y sustituirá la última llave “}” de la última línea por una llave seguida de un corchete “}]]”.

```
#!/bin/bash

echo "Obteniendo datos"
hdfs dfs -get /output/output.json ./public/exit/
cat ./public/exit/output.json/part-* > ./public/exit/data.json
sed -i '1s/{/{/' ./public/exit/data.json
sed -i 's/}/},/g' ./public/exit/data.json
sed -i '$s/},/}]/' ./public/exit/data.json
rm -R ./public/exit/output.json
echo "Consulta finalizada"
```

37 Código script descarga y formato fichero JSON

### 3.5.5 REPOSITORIO CON EL CÓDIGO DE LA APLICACIÓN WEB

Se ha habilitado un repositorio público en GitHub donde es posible consultar el código completo de la aplicación que ha sido comentado en esta sección.

Para consultar el código utilice este [enlace](#).

## 4. CONCLUSIONES

Cuando comenzamos la búsqueda de temas para desarrollar el trabajo final de título sabíamos que debíamos aprovechar la oportunidad para aprender algo con lo que no se haya podido trabajar a lo largo de estos últimos años en la universidad. Enseguida pensamos en Big Data, un concepto que ha ido surgiendo a lo largo de numerosas asignaturas, pero sobre el que nunca hemos entrado en detalle, algo que a día de hoy aun alcanzamos a comprender ya que todos los docentes que han hablado de este campo lo presentan como uno de los más importantes de cara al futuro en la ciencia de datos.

Al comenzar un estudio superficial sobre el tema y pudimos comprobar que existe una tendencia por parte de las compañías para integrar en ellas el Big Data motivada por la enorme cantidad de datos que son capaces de recopilar ya sea de fuentes externas, como sus clientes, o de la propia compañía. El gran volumen, la variedad y el rápido crecimiento de dichos datos requieren el uso de tecnologías como la que nos había llamado la atención.

Sin embargo, en la reunión con el tutor nos sorprendió al hablar de *Learning Analytics*, un campo de investigación y conocimiento relativamente joven que lleva a la práctica el Big Data para un entorno educativo con el objetivo de proporcionar un aprendizaje personalizado, adaptativo y transparente. Como estudiante cuyo paso por la universidad no ha sido del todo satisfactorio y que lleva años discutiendo con compañeros posibles mejoras en el sistema educativo universitario, se aceptó la propuesta tan pronto como terminamos de discutirla con el tutor.

En cuanto al desarrollo del trabajo, una buena parte se ha basado en la formación, esto era de esperar ya que como mencionamos anteriormente este es un tema que durante nuestra formación apenas se ha estudiado. En cuanto nos vimos preparados comenzó el desarrollo y pudimos observar que estamos ante unas tecnologías muy recientes y con amplio margen de mejora. Además, el número de usuarios que se encuentran utilizando dichas tecnologías es bastante reducido, por lo que encontrar información o ayuda a lo largo del proceso de desarrollo de este proyecto no ha sido fácil.

Aunque no hemos podido probar personalmente los recursos del CICEI por motivos de desplazamiento, compañeros de carrera han compartido sus experiencias y debemos mostrar nuestra decepción una vez más con la poca importancia que parece recibir la ciencia de datos en esta institución y que se ve reflejada en los limitados recursos con los que tienen que ingeniárselas los compañeros del centro de investigación.

A pesar de esto estamos satisfechos con el trabajo realizado, el cual demuestra que no es necesario poseer grandes recursos para aplicar Big Data a un entorno como el educativo. Aunque a pesar de la accesibilidad y el potencial de lo expuesto en este proyecto todo queda en manos del usuario, en este caso el docente, ya que sin su trabajo de interpretación y el esfuerzo por cambiar o mejorar aquello que considere nos encontramos antes un trabajo casi inútil.

Por tanto, creemos que además de la divulgación de campos como *Learning Analytics* se debe concienciar e instruir a las organizaciones e instituciones dedicadas a la educación para dar un uso correcto de las tecnologías y aplicar los cambios necesarios para mejorar, al fin y al cabo, es lo que da valor al desarrollo de proyectos como este.

## 5. TRABAJOS FUTUROS

Todos los objetivos planteados al comienzo del desarrollo de este proyecto final de carrera se han cumplido satisfactoriamente y sientan una base de forma que este proyecto puede ser utilizado para el estudio de diversos *datasets* con solo cambiar unas líneas de código referentes a los parámetros de las consultas.

Sin embargo, se consideran las siguientes mejoras de cara a perfeccionar las aplicaciones:

- La unión de todas las tablas que forman el *dataset* cada vez que se realiza una nueva consulta ralentiza la ejecución de manera considerable. Aunque esto es inevitable con un *dataset* con una estructura como la del *OULAD*, sería recomendable utilizar un *dataset* mejor preparado para trabajar como uno solo.
- La aplicación web a pesar de ser completamente funcional y tiene un amplio margen de mejora en la interfaz.
- La representación de los resultados de las consultas podría ser más cómoda para el usuario.
- Se pueden añadir más gráficas para el estudio de los datos, aunque para ello sería fundamental discutirlo con el posible usuario de la aplicación, a ser posible un docente.
- Posibilidad de almacenar los resultados de las consultas más recientes.

## 6. NORMATIVA Y LEGISLACIÓN

El *dataset* utilizado para el desarrollo de este proyecto final de grado es público y gratuito bajo una licencia [CC-BY 4.0](#) la cual permite copiar, redistribuir, transformar y crear a partir del material bajo las condiciones de reconocimiento.

Pueden encontrarse más detalles acerca del dataset en el siguiente [enlace](#).

## 7. FUENTES DE INFORMACIÓN

- |     |                                       |   |
|-----|---------------------------------------|---|
| [1] | <i>Open University Dataset</i>        | <a href="https://www.nature.com/articles/sdata2017171">https://www.nature.com/articles/sdata2017171</a>   |
| [2] | Información Big Data                  | <a href="https://www.sas.com/es_es/insights/big-data/what-is-big-data.html">https://www.sas.com/es_es/insights/big-data/what-is-big-data.html</a>   |
| [3] | Información detallada Big Data        | <a href="https://www.oracle.com/big-data/guide/what-is-big-data.html">https://www.oracle.com/big-data/guide/what-is-big-data.html</a>   |
| [4] | Información <i>Learning Analytics</i> | <a href="https://www.oei.es/historico/divulgacioncientifica/?Learning-analytics-instrumento">https://www.oei.es/historico/divulgacioncientifica/?Learning-analytics-instrumento</a>       |
| [5] | Información <i>Learning Analytics</i> | <a href="https://digitallearning.northwestern.edu/article/2016/04/12/learning-analytics-basics">https://digitallearning.northwestern.edu/article/2016/04/12/learning-analytics-basics</a> |
| [6] | Información <i>Learning Analytics</i> | <a href="https://es.wikipedia.org/wiki/Anal%C3%ADtica_de_aprendizaje">https://es.wikipedia.org/wiki/Anal%C3%ADtica_de_aprendizaje</a>   |
| [7] | Curso Hadoop Big Data                 | <a href="https://www.udemy.com/monta-un-cluster-hadoop-big-data-desde-cero/">https://www.udemy.com/monta-un-cluster-hadoop-big-data-desde-cero/</a>                                       |
| [8] | Documentación Hadoop                  | <a href="https://hadoop.apache.org/docs/stable/">https://hadoop.apache.org/docs/stable/</a>   |
| [9] | Información Hadoop                    | <a href="https://www.sas.com/es_es/insights/big-data/hadoop.html">https://www.sas.com/es_es/insights/big-data/hadoop.html</a>   |

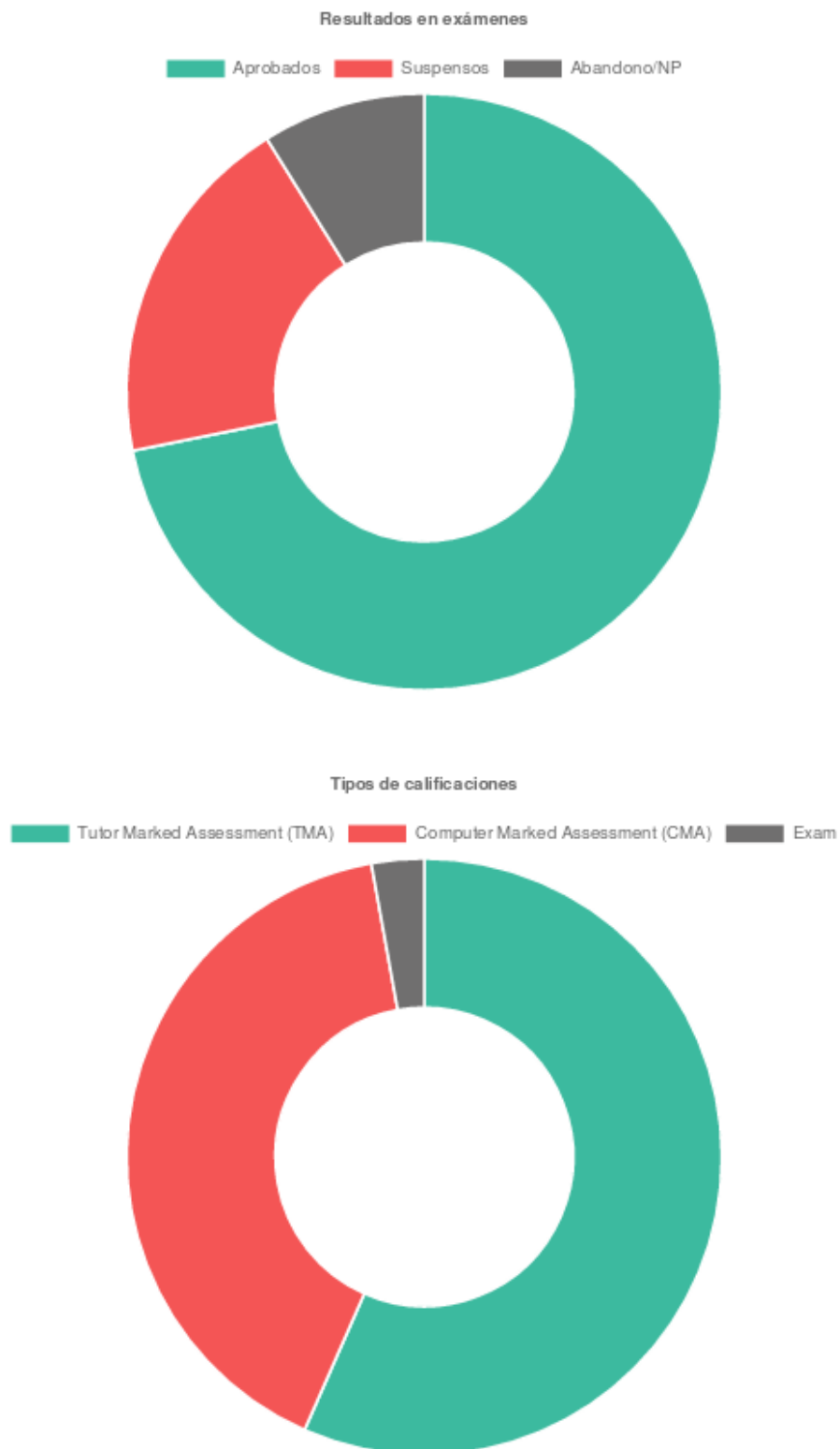
- [10] Información Hadoop <https://momentotic.com/2013/05/16/que-es-hadoop/>
- [11] Documentación Spark <https://spark.apache.org/docs/latest/>
- [12] Creación proyecto Scala + SBT <https://docs.scala-lang.org/getting-started-intellij-track/building-a-scala-project-with-intellij-and-sbt.html>
- [13] Formación en Scala <https://docs.scala-lang.org/es/tutorials/scala-for-java-programmers.html>
- [14] Formación en Scala <https://www.udemy.com/beginning-scala-programming/>
- [15] Lectura ficheros CSV <https://stackoverflow.com/questions/29704333/spark-load-csv-file-as-dataframe>
- [16] Parámetros Spark SQL <https://forums.databricks.com/questions/115/how-do-i-pass-parameters-to-my-sql-statements.html>
- [17] Join Spark Dataframes <https://stackoverflow.com/questions/40343625/joining-spark-dataframes-on-the-key/55286470#55286470>
- [18] Sentencia SQL vacía <https://stackoverflow.com/questions/29886290/sql-where-is-anything>
- [19] Información spark-submit [https://www.cloudera.com/documentation/enterprise/5-6-x/topics/cdh\\_ig\\_running\\_spark\\_apps.html](https://www.cloudera.com/documentation/enterprise/5-6-x/topics/cdh_ig_running_spark_apps.html)
- [20] Generar proyecto Node.js + express [https://developer.mozilla.org/es/docs/Learn/Server-side/Express\\_Nodejs/skeleton\\_website](https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/skeleton_website)
- [21] Documentación Bootstrap <https://getbootstrap.com/docs/4.1/getting-started/contents/>

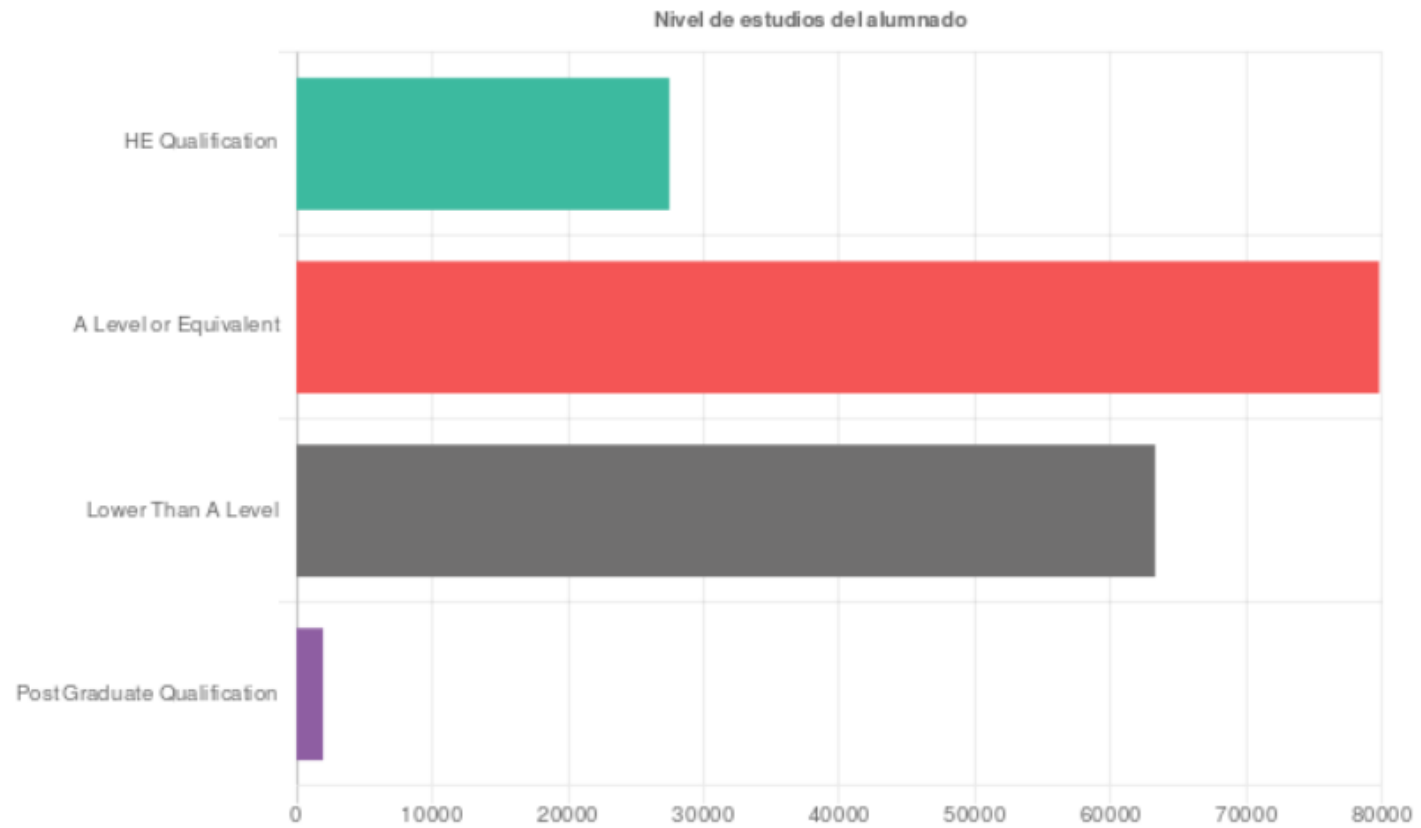


- [22] Documentación JavaScript <https://devdocs.io/javascript/>
- [23] Documentación jQuery <https://api.jquery.com/>
- [24] Documentación Chart.js <http://microbuilder.io/blog/2016/01/10/plottin-g-json-data-with-chart-js.html>
- [25] Lectura JSON en Chart.js <https://stackoverflow.com/questions/44990517/displaying-json-data-in-chartjs>
- [26] Chart.js Boxplot and Violin addon <https://github.com/datavisyn/chartjs-chart-box-and-violin-plot>
- [27] Bash Script Statements <https://www.thegeekstuff.com/2010/06/bash-if-statement-examples/>
- [28] Información comando sed <https://www.americati.com/doc/sed/sed.html>
- [29] Manual Operativo ULPGC [https://www.eii.ulpgc.es/tb\\_university\\_ex/sites/default/files/files/trabajos%20fin%20de%20grado/Formularios/Manual%20Operativo%20TFT%20EII%20\(R%20Rev%202010\).pdf](https://www.eii.ulpgc.es/tb_university_ex/sites/default/files/files/trabajos%20fin%20de%20grado/Formularios/Manual%20Operativo%20TFT%20EII%20(R%20Rev%202010).pdf)

## 8. ANEXOS

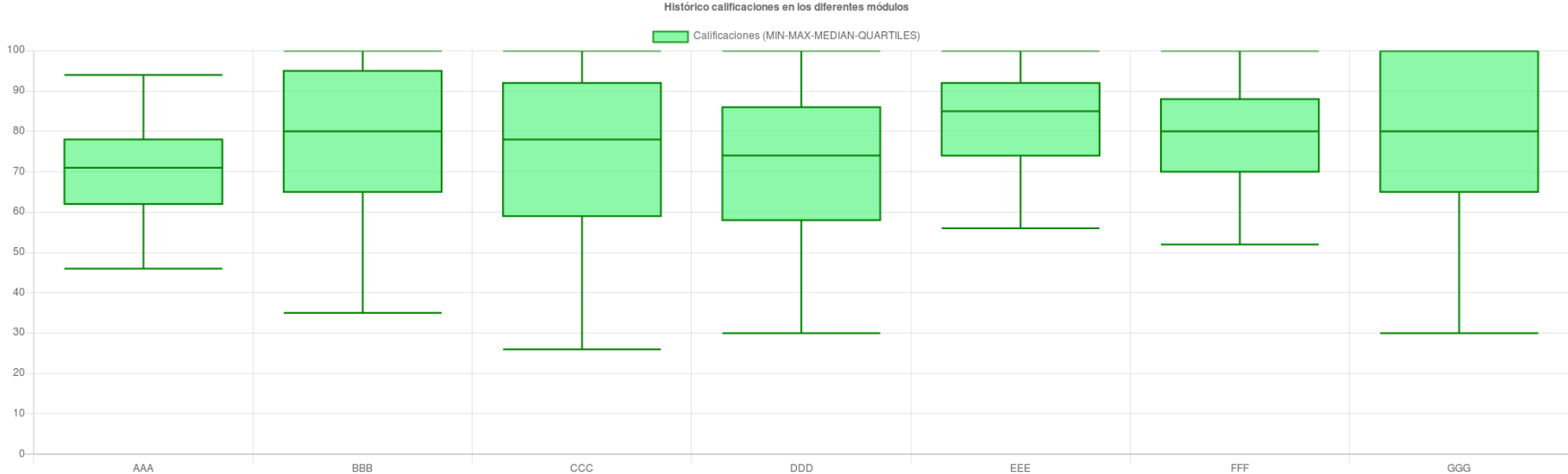
### 8.2 ANEXO 1 – GRÁFICA 1





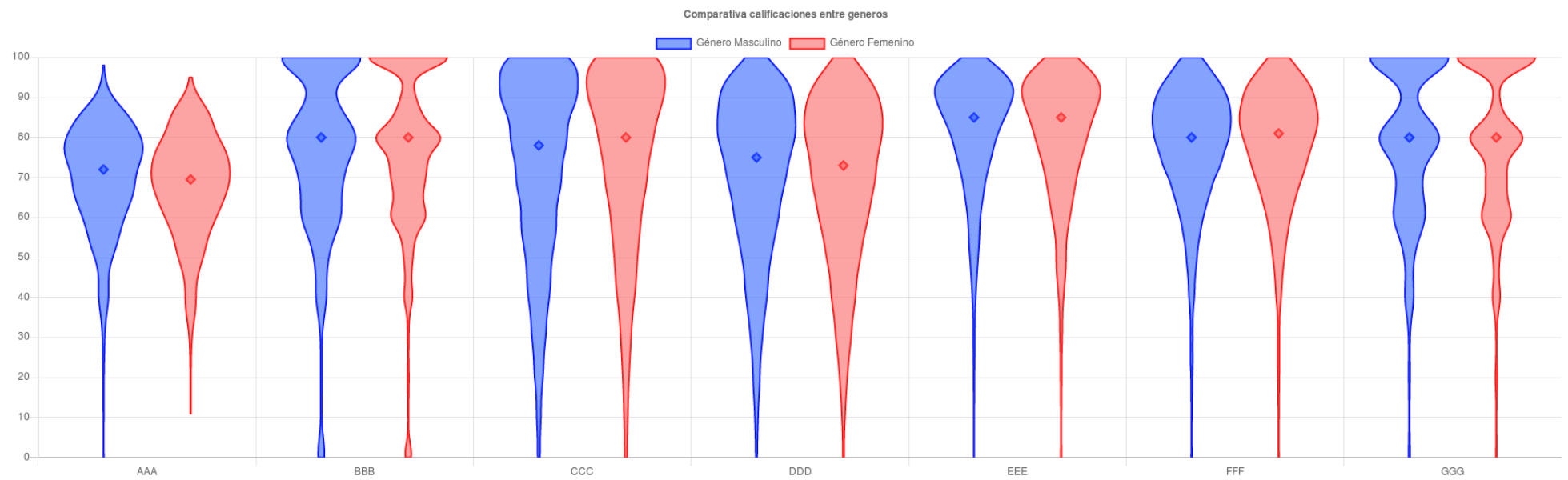
### 8.3 ANEXO 2 – GRÁFICA 2

#### Learning Analytics



## 8.4 ANEXO 3 – GRÁFICA 3

### Learning Analytics



## 8.5 ANEXO 4 – GRÁFICA 4

### Learning Analytics

