**Título:** Modelo predictor basado en datos reales de Spotify para decidir cuándo un usuario escuchará una canción completa

**Autor:** Alberto Romero Vázquez

Contenido

[Resumen 7](#_Toc46262989)

[Introducción 8](#_Toc46262990)

[Metodología CRISP-DM 8](#_Toc46262991)

[Caso de uso: Valor para el negocio 9](#_Toc46262992)

[Objetivo del proyecto 11](#_Toc46262993)

[Planificación: Gantt 11](#_Toc46262994)

[1. Requisitos 12](#_Toc46262995)

[1.1 Servicios 12](#_Toc46262996)

[1.2 Contenedores 13](#_Toc46262997)

[2. Datos 14](#_Toc46262998)

[3. Análisis exploratorio 18](#_Toc46262999)

[4. Del dato al valor 53](#_Toc46263000)

[5. Modelos de Machine Learning 62](#_Toc46263001)

[5.1. Regresión Logística 63](#_Toc46263002)

[5.2. Árbol de Decisión 65](#_Toc46263003)

[5.3. Random Forest 67](#_Toc46263004)

[5.4. Máquina de Vector Soporte (SVM) 69](#_Toc46263005)

[5.5. Análisis de Discriminante Gaussiano 71](#_Toc46263006)

[5.6. K-NN (K-Vecinos más próximos) 73](#_Toc46263007)

[6. Selección del modelo 75](#_Toc46263008)

[6.1. Curva de aprendizaje 75](#_Toc46263009)

[6.2. Serialización 76](#_Toc46263010)

[7. Arquitectura en Docker 77](#_Toc46263011)

[7.1 HDFS (Hadoop Distributed File System) 78](#_Toc46263012)

[7.2. Hive (con PostgreSQL) 81](#_Toc46263013)

[7.3. Lanzando HDFS y Hive 84](#_Toc46263014)

[7.4. Almacenando ficheros en Hive sobre HDFS 86](#_Toc46263015)

[7.5. PySpark 86](#_Toc46263016)

[7.6. Visualizaciones con ELK 88](#_Toc46263017)

[8. Evolución 94](#_Toc46263018)

[9. Conclusiones 95](#_Toc46263019)

[Documentos entregados 98](#_Toc46263020)

[Anexo 98](#_Toc46263021)

[Hola-Mundo en Docker 98](#_Toc46263022)

[Notebook de Jupyter en Docker 99](#_Toc46263023)

[Entorno del notebook de Jupyter del modelo 100](#_Toc46263024)

[Docker-compose para HDFS y Hive 104](#_Toc46263025)

[Trabajando con Hive y el modelo 108](#_Toc46263026)

[Referencias 120](#_Toc46263027)

[Ilustración 1: Metodología CRISP-DM 9](#_Toc46263125)

[Ilustración 2: Stack tecnológico 9](#_Toc46263126)

[Ilustración 3: Planificación Gantt 11](#_Toc46263127)

[Ilustración 4: Jupyter-notebook-Docker 13](#_Toc46263128)

[Ilustración 5: Histogramas de la tabla de canciones 23](#_Toc46263129)

[Ilustración 6: Histogramas de la tabla de sesión 24](#_Toc46263130)

[Ilustración 7: Densidad en tabla de canciones - duration 25](#_Toc46263131)

[Ilustración 8: Densidad en tabla de canciones - release\_year 25](#_Toc46263132)

[Ilustración 9: Densidad en tabla de canciones - us\_popularity\_estimate 26](#_Toc46263133)

[Ilustración 10: Densidad en tabla de canciones - acousticness 26](#_Toc46263134)

[Ilustración 11: Densidad en tabla de canciones - beat\_strength 27](#_Toc46263135)

[Ilustración 12: Densidad en tabla de canciones - bounciness 27](#_Toc46263136)

[Ilustración 13: Densidad en tabla de canciones - danceability 28](#_Toc46263137)

[Ilustración 14: Densidad en tabla de canciones - dyn\_range\_mean 28](#_Toc46263138)

[Ilustración 15: Densidad en tabla de canciones - energy 29](#_Toc46263139)

[Ilustración 16: Densidad en tabla de canciones - flatness 29](#_Toc46263140)

[Ilustración 17: Densidad en tabla de canciones - instrumentalness 30](#_Toc46263141)

[Ilustración 18: Densidad en tabla de canciones - key 30](#_Toc46263142)

[Ilustración 19: Densidad en tabla de canciones - liveness 31](#_Toc46263143)

[Ilustración 20: Densidad en tabla de canciones - loudness 31](#_Toc46263144)

[Ilustración 21: Densidad en tabla de canciones - mechanism 32](#_Toc46263145)

[Ilustración 22: Densidad en tabla de canciones - organism 32](#_Toc46263146)

[Ilustración 23: Densidad en tabla de canciones - speechiness 33](#_Toc46263147)

[Ilustración 24: Densidad en tabla de canciones - tempo 33](#_Toc46263148)

[Ilustración 25: Densidad en tabla de canciones - time\_signature 34](#_Toc46263149)

[Ilustración 26: Densidad en tabla de canciones - valence 34](#_Toc46263150)

[Ilustración 27: Densidad en tabla de canciones - acoustic\_vector\_0 35](#_Toc46263151)

[Ilustración 28: Densidad en tabla de canciones - acoustic\_vector\_1 35](#_Toc46263152)

[Ilustración 29: Densidad en tabla de canciones - acoustic\_vector\_2 36](#_Toc46263153)

[Ilustración 30: Densidad en tabla de canciones - acoustic\_vector\_3 36](#_Toc46263154)

[Ilustración 31: Densidad en tabla de canciones - acoustic\_vector\_4 37](#_Toc46263155)

[Ilustración 32: Densidad en tabla de canciones - acoustic\_vector\_5 37](#_Toc46263156)

[Ilustración 33: Densidad en tabla de canciones - acoustic\_vector\_6 38](#_Toc46263157)

[Ilustración 34: Densidad en tabla de canciones - acoustic\_vector\_7 38](#_Toc46263158)

[Ilustración 35: Densidad en tabla de sesión - session\_position 39](#_Toc46263159)

[Ilustración 36: Densidad en tabla de sesión - session\_length 39](#_Toc46263160)

[Ilustración 37: Densidad en tabla de sesión - context\_switch 40](#_Toc46263161)

[Ilustración 38: Densidad en tabla de sesión - no\_pause\_before\_play 40](#_Toc46263162)

[Ilustración 39: Densidad en tabla de sesión - short\_pause\_before\_play 41](#_Toc46263163)

[Ilustración 40: Densidad en tabla de sesión - long\_pause\_before\_play 41](#_Toc46263164)

[Ilustración 41: Densidad en tabla de sesión - hist\_user\_behavior\_n\_seekfwd 42](#_Toc46263165)

[Ilustración 42: Densidad en tabla de sesión - hist\_user\_behavior\_n\_seekback 42](#_Toc46263166)

[Ilustración 43: Densidad en tabla de sesión - hour\_of\_day 43](#_Toc46263167)

[Ilustración 44: Diagrama de cajas en canciones - duration, release\_year 43](#_Toc46263168)

[Ilustración 45: Diagrama de cajas en canciones - us\_popularity\_estimate, acousticness 44](#_Toc46263169)

[Ilustración 46: Diagrama de cajas en canciones - beat\_strength, bounciness 44](#_Toc46263170)

[Ilustración 47: Diagrama de cajas en canciones - danceability, dyn\_range\_mean 45](#_Toc46263171)

[Ilustración 48: Diagrama de cajas en canciones - energy, flatness 45](#_Toc46263172)

[Ilustración 49: Diagrama de cajas en canciones - instrumentalness, key 46](#_Toc46263173)

[Ilustración 50: Diagrama de cajas en canciones - liveness, loudness 46](#_Toc46263174)

[Ilustración 51: Diagrama de cajas en canciones - mechanism, organism 47](#_Toc46263175)

[Ilustración 52: Diagrama de cajas en canciones - speechniness, tempo 47](#_Toc46263176)

[Ilustración 53: Diagrama de cajas en canciones - time\_signature, valence 48](#_Toc46263177)

[Ilustración 54: Diagrama de cajas en canciones - acoustic\_vector\_0, acoustic\_vector\_1 48](#_Toc46263178)

[Ilustración 55: Diagrama de cajas en canciones - acoustic\_vector\_2, acoustic\_vector\_3 49](#_Toc46263179)

[Ilustración 56: Diagrama de cajas en canciones - acoustic\_vector\_4, acoustic\_vector\_5 49](#_Toc46263180)

[Ilustración 57: Diagrama de cajas en canciones - acoustic\_vector\_6, acoustic\_vector\_7 50](#_Toc46263181)

[Ilustración 58: Diagrama de cajas en sesión - session\_position, session\_length, context\_switch, no\_pause\_before\_play 51](#_Toc46263182)

[Ilustración 59: Diagrama de cajas en sesión – short\_pause\_before\_play, long\_pause\_before\_play, hist\_user\_behavior\_n\_seekfwd, hist\_user\_behavior\_n\_seekback 52](#_Toc46263183)

[Ilustración 60: Diagrama de cajas en sesión - hour\_of\_day 53](#_Toc46263184)

[Ilustración 61: Correlaciones mapa de calor con tablón balanceado 58](#_Toc46263185)

[Ilustración 62: Correlaciones mapa de calor con tablón balanceado - marcando outliers 59](#_Toc46263186)

[Ilustración 63: Correlaciones mapa de calor con tablón balanceado - sin outliers 60](#_Toc46263187)

[Ilustración 64: Regresión Logística 64](#_Toc46263188)

[Ilustración 65: Regresión Logística - métricas 64](#_Toc46263189)

[Ilustración 66: Árbol de decisión 66](#_Toc46263190)

[Ilustración 67: Árbol de Decisión – métricas 66](#_Toc46263191)

[Ilustración 68: Random Forest 68](#_Toc46263192)

[Ilustración 69: Random Forest – métricas 68](#_Toc46263193)

[Ilustración 70: Máquina de Vector Soporte 70](#_Toc46263194)

[Ilustración 71: Máquina de Vector Soporte – métricas 70](#_Toc46263195)

[Ilustración 72: Análisis de Discriminante Gaussiano 72](#_Toc46263196)

[Ilustración 73: Análisis de Discriminante Gaussiano – métricas 72](#_Toc46263197)

[Ilustración 74: K-Vecinos más próximos 73](#_Toc46263198)

[Ilustración 75: K-Vecinos más próximos – métricas 74](#_Toc46263199)

[Ilustración 76: Curva de aprendizaje 76](#_Toc46263200)

[Ilustración 77: HDFS - namenode 79](#_Toc46263201)

[Ilustración 78: HDFS - datanode 80](#_Toc46263202)

[Ilustración 79: HUE 81](#_Toc46263203)

[Ilustración 80: Docker-compose-Hive 83](#_Toc46263204)

[Ilustración 81: namenode - URL 84](#_Toc46263205)

[Ilustración 82: datanode - URL 85](#_Toc46263206)

[Ilustración 83: PySpark en Jupyter 87](#_Toc46263207)

[Ilustración 84: SparkSession 87](#_Toc46263208)

[Ilustración 85: PySpark join 88](#_Toc46263209)

[Ilustración 86: Elasticsearch - mapeo 89](#_Toc46263210)

[Ilustración 87: Logstash, tipos correctos 90](#_Toc46263211)

[Ilustración 88:Elasticsearch, tipos correctos 91](#_Toc46263212)

[Ilustración 89: Kibana, discover 92](#_Toc46263213)

[Ilustración 90:Kibana. índice paso 1 92](#_Toc46263214)

[Ilustración 91: Kibana, índice paso 2 93](#_Toc46263215)

[Ilustración 92:Dashboard en Kibana 94](#_Toc46263216)

# Resumen

Esta memoria describe los pasos seguidos para conseguir un clasificador que trabaje con datos reales de Spotify para detectar cuándo un usuario va a saltar la escucha de una canción.

El objetivo del proyecto es educativo, buscando reforzar los conocimientos teóricos llevados a la práctica.

Se busca realizar un modelo basado en un banco de modelos y a la vez desplegar una arquitectura Big Data basada en servicios en contenedores Docker. La arquitectura cubre HDFS, Hive con un metastore PostgreSQL, PySpark y la pila ELK para cubrir las visualizaciones.

La metodología aplicada es CRISP-DM y pese a haber sido realizado de forma individual, se han simulado distintos roles Big Data ajustándose a una planificación inicial.

Este trabajo cubre todas las secciones formales propias de una memoria completa.

Adicionalmente, si se desea más detalle se puede acudir a la sección de Anexos.

Todos los resultados se han obtenido ejecutando los ficheros que se adjuntan con esta memoria, incluida la presentación que acompaña a este trabajo.

# Introducción

En este proyecto vamos a tratar un caso real obtenido de la plataforma especializada en inteligencia artificial [www.aicrowd.com](http://www.aicrowd.com). En concreto vamos a estudiar un problema abierto a concurso en 2019 y esponsorizado por Spotify (aicrowd, 2019). Spotify es una compañía muy popular que trabaja ofreciendo un servicio de música en streaming (wikipedia Spotify, s.f.). Spotify buscaba conocer cuándo un usuario iba a saltar una canción; es decir, cuándo una canción no se iba a escuchar entera y se iba a saltar a otra canción antes de que se acabara toda la duración completa de la canción. El objetivo de Spotify es poder identificar qué canciones no van a interesar al usuario que está en su sesión de Spotify escuchando música, para no proponerle canciones que no va a acabar. Esto es muy importante porque de este modo Spotify se preocupa de la experiencia de usuario. Se busca que un usuario conectado a Spotify escuche, tenga alrededor y descubra canciones que le van a gustar, para que el usuario asocie Spotify a un buen servicio. Si un usuario recibiese solo propuestas de canciones que acabará saltando, el usuario no disfrutaría con la experiencia de Spotify y podría darse de baja y no recomendar el servicio, afectando esto a la imagen de la marca.

## Metodología CRISP-DM

Con el objetivo de trabajar siguiendo una metodología estandarizada, vamos a trabajar con el paradigma del proceso CRISP-DM (Cross-Industry Standard Process for Data Mining). Esta elección se debe a que este trabajo se va a fundamentar completamente en la información que extraigamos de los datos.

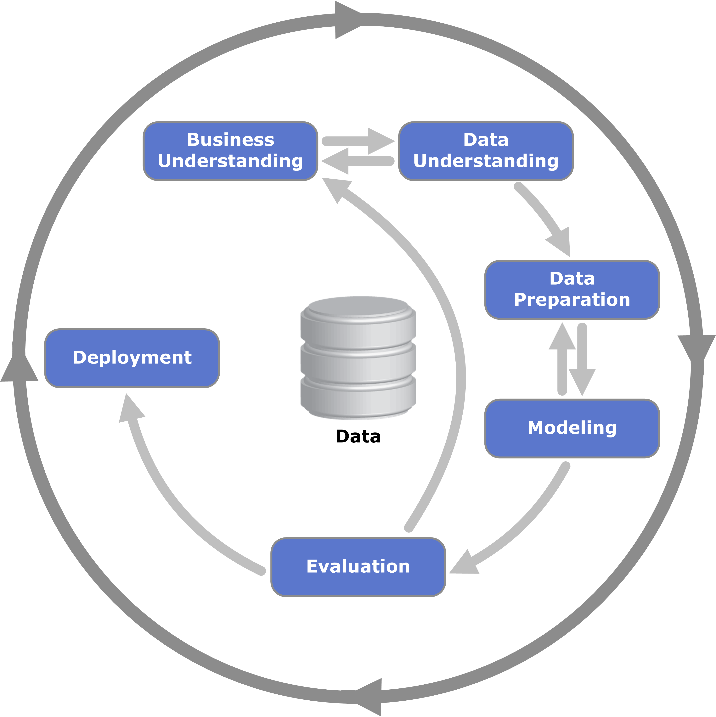


Ilustración : Metodología CRISP-DM

## Caso de uso: Valor para el negocio

El concurso ya está cerrado, pero usaremos los datos porque nos interesa utilizar datos cuyo origen es real. Nuestro objetivo no será tanto optimizar continuamente el algoritmo predictor, así como aplicar los conocimientos adquiridos teóricos, incluyendo un despliegue asociado al mundo Big Data, para simular un entorno de trabajo más completo y general.

En este proyecto vamos a abarcar una primera fase de despliegue tecnológico Big Data, utilizando la plataforma de contenedores Docker (Docker, 2020) para arrancar los servicios que emplearemos.

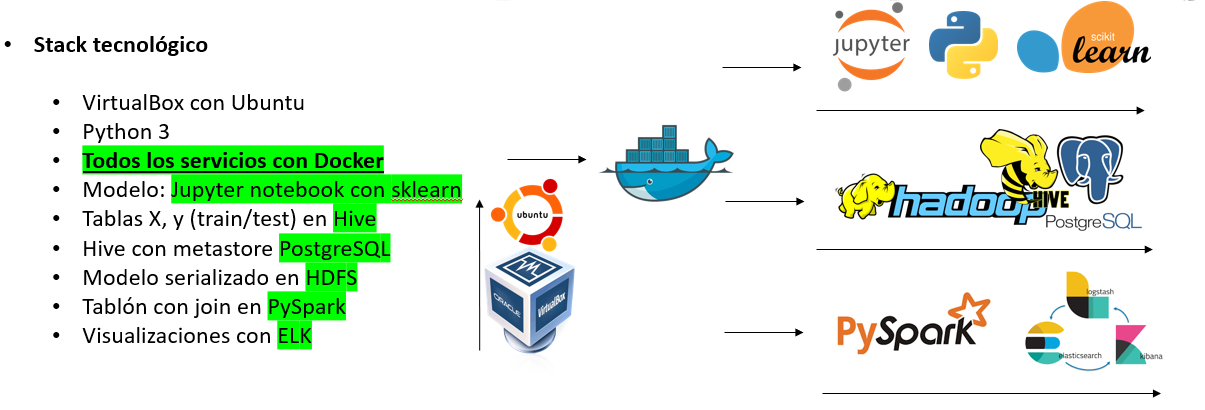


Ilustración : Stack tecnológico

Dentro de estos servicios abriremos un notebook Jupyter (Jupyter.org, 2020) donde podremos trabajar con el modelo: entenderemos las fuentes, procesaremos los datos, obtendremos información de ellos, aplicaremos varios modelos y elegiremos el mejor con base en ciertas métricas. Una vez hayamos obtenido el modelo que mejor prediga cuándo un usuario de Spotify va o no a escuchar una canción entera, serializaremos el modelo.

Almacenaremos en una infraestructura Big Data (Hive (Apache.org, Hive, 2020)) los cuatro ficheros con formato csv procesados en el modelo, siendo estos:

* La entrada preprocesada del modelo en entrenamiento
* La entrada preprocesada del modelo en test
* La salida del modelo en entrenamiento
* La salida del modelo en test

También hemos obtenido el propio modelo serializado, que a su vez guardaremos en la infraestructura Big Data (HDFS, *Hadoop Distributed File System*) (Apache.org, Hadoop, 2020).

En este punto, se emplea la infraestructura desplegada en contenedores Docker para acudir al servicio de Hive sobre HDFS. Decidimos almacenar los ficheros csv en Hive puesto que Hive guarda los datos en forma tabular.

Posteriormente, accederemos directamente al HDFS para almacenar ahí el modelo serializado (cuyo formato es una ristra binaria).

Cada apartado de este proyecto explicará en detalle y profundidad los aspectos tanto del modelo como de la arquitectura desplegada en servicios que corren en contenedores Docker.

Por último, deseamos comentar que también se incluyen servicios de Spark y de ELK (Elastic – Logstash – Kibana) para profundizar en esta tecnología escalable tan demandada.

## Objetivo del proyecto

La gran meta final es poder aplicar de forma académica los conocimientos adquiridos teóricos, pero con un sentido lógico. Por ejemplo, la propia naturaleza de los datos de entrada es tabular, por lo que tiene sentido utilizar almacenamiento en HDFS con una capa de Hive por encima. También vemos muy interesante poder hacer uso de los contenedores Docker, porque facilita mucho el entorno de desarrollo, a la vez que afianzamos conocimientos obtenidos en la preparación del proyecto.

Por otro lado, no es necesario tener que abstraerse en Hive para almacenar el modelo serializado, porque en este caso es más sencillo aplicar una transacción en HDFS a nivel de sistema de ficheros (fs) que prepare el directorio en el cluster y de dfs para la propia transacción.

Se completan las tecnologías con un procesado extra en Spark SQL y con visualizaciones en Kibana apoyada en Elasticsearch y Logstash. De nuevo, estos servicios corren en contenedores Docker.

Todos los desarrollos se han realizado en modo stand alone (usando una única máquina), pero simulando un entorno cluster de Big Data.

## Planificación: Gantt

En el siguiente esquema se puede ver la planificación seguida en el proyecto.

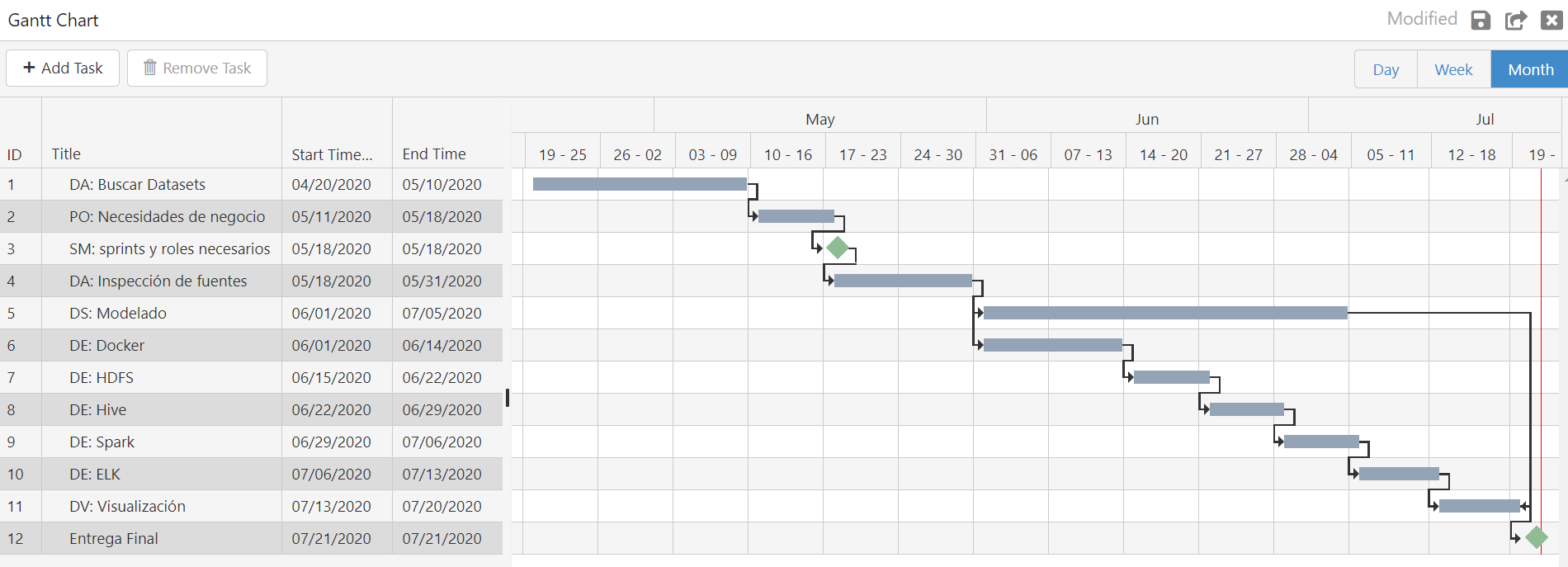


Ilustración : Planificación Gantt

Pese a ser un trabajo individual, se han aplicado los siguientes roles:

* Rol: Business Owner (BO) - las necesidades de negocio
* Rol: Product Owner (PO) - la definición de las tareas e hitos para DA, DS, DE y DV
* Rol: Scrum Master (SM) - ver la evolución de los sprints con las tareas y relaciones
* Rol: Data Analyst (DA) - entender las fuentes y el estado actual, trasladar el proyecto de negocio a técnico
* Rol: Data Scientist (DS) - aplicación de matemáticas y programación para extraer valor de los datos
* Rol: Data Engineer (DE) - preparar los entornos Big Data a nivel informático
* Rol: Data Visualization (DV) - preparar las visualizaciones útiles para negocio

# 1. Requisitos

Para crear y ejecutar este proyecto nos vamos a instalar una máquina virtual Ubuntu alojada en una plataforma OracleVirtualBox. Acudimos a Ubuntu para tener la seguridad de que Docker va a funcionar sin ningún problema y para tener también detrás el respaldo de una distribución de Linux, siendo este un sistema operativo ampliamente utilizado en entornos de desarrollo (Ubuntu.com, 2020).

## 1.1 Servicios

En los grandes proyectos de BigData se tiende a trabajar con arquitecturas por servicios, de forma que el mantenimiento y gestión de un servicio no afecte a otros. Además, así también optimizamos los recursos utilizados, puesto que los ajustamos a las necesidades de cada servicio.

Por este motivo y también para que el despliegue sea más sencillo e independiente de los entornos locales, vamos a emplear Docker como plataforma de contenedores.

Instalamos Docker y comprobamos que lo tenemos instalado siguiendo los pasos que se citan en el tutorial recogido en la sección de referencias (S., s.f.).

Una vez hemos acabado de instalar Docker en la máquina virtual, visualizamos la versión que hemos instalado:

**ubuntu@ubuntu-VirtualBox**:**~**$ docker --version

Docker version 19.03.10, build 9424aeaee9[[1]](#footnote-1)

Docker se ha instalado correctamente.

Como cualquier nueva tecnología que acabemos de instalar, vamos a lanzar un Hola Mundo para ver desde un nivel básico e inicial cómo funciona (detalle en Anexos).

Hemos conseguido lanzar nuestro primer Hola Mundo con Docker.

Nuestro siguiente objetivo es lanzar un contenedor Docker que corra un notebook de Jupyter donde podremos trabajar con los datos del problema.

## 1.2 Contenedores

Creamos el directorio para trabajar con el notebook de Jupyter.

**ubuntu@ubuntu-VirtualBox**:**~**$ mkdir ~/notebooks

Descargamos en /notebooks el notebook de Jupyter guardado en la carpeta notebooks.zip incluida en los ficheros adjuntos a esta memoria donde se incluyen todos los documentos necesarios para la ejecución de este proyecto.

El notebook de Jupyter se llama Alberto\_Romero\_Proyecto.ipynb. Lanzamos un notebook de Jupyter desde Docker (ver Anexo).

Accedemos al Jupyter.

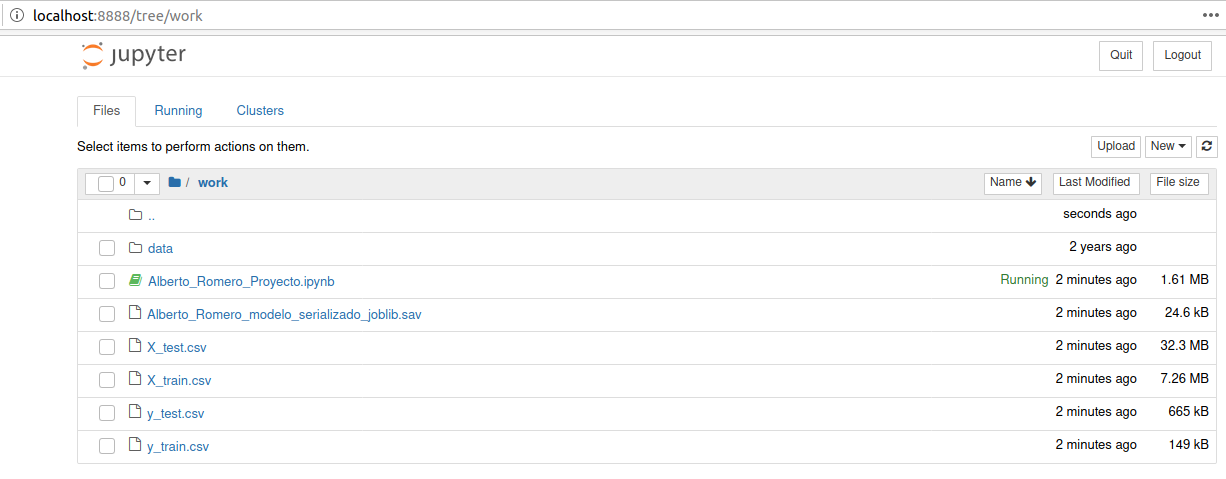


Ilustración 4: Jupyter-notebook-Docker

Abrimos el documento Alberto\_Romero\_modelo.ipynb para llegar al documento que describe el modelo.

El modelo recorrerá todas las etapas que se definen en el marco CRISP-DM, desde la toma de las fuentes, pasando por el procesado adaptado a las necesidades de negocio, hasta llegar a la extracción de información útil de los datos para la toma de decisiones.

# 2. Datos

Como ya hemos visto, el proyecto se basa en analizar el comportamiento de los usuarios de Spotify, estudiando datos reales. La fuente de datos se encuentra en la URL del reto de Spotify (aicrowd, 2019).

Pasos:

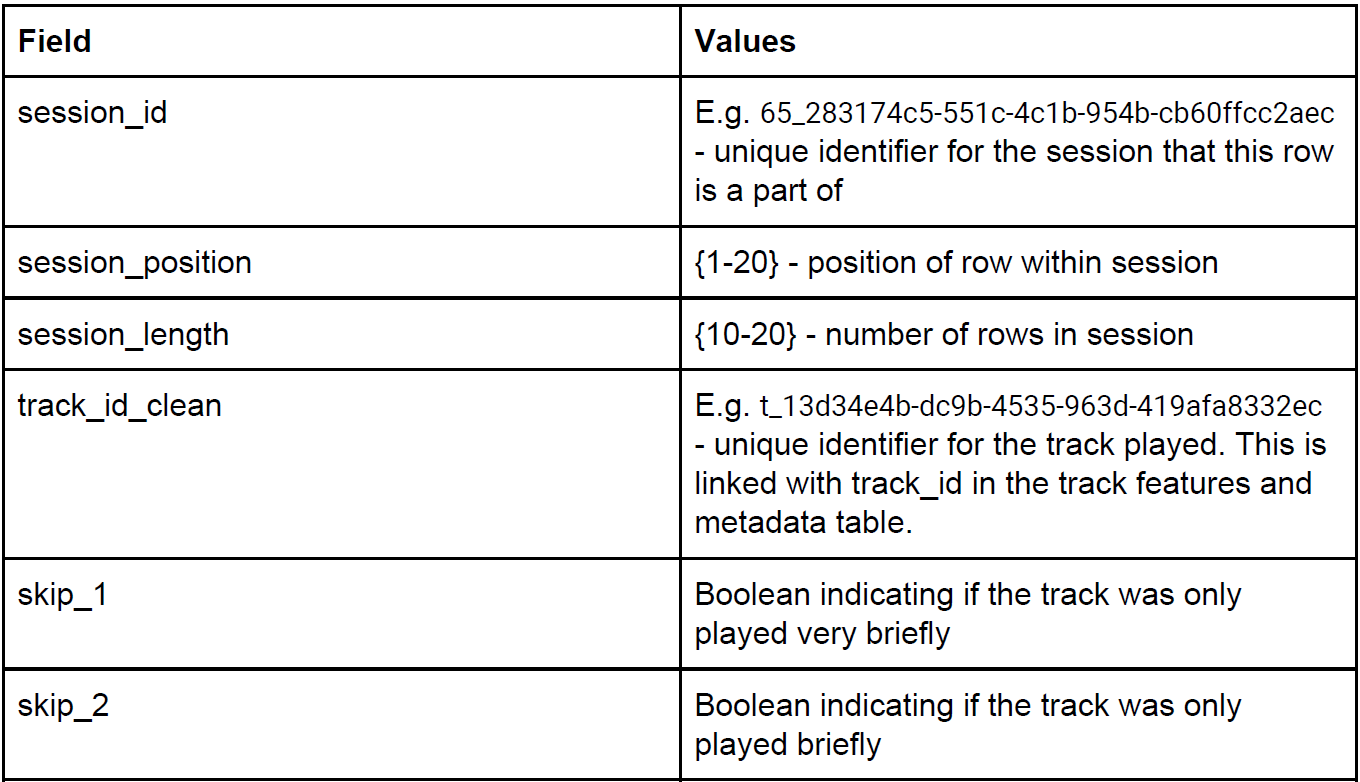
1. Se necesita una cuenta en www.aicrowd.com, web de IA y de Data Science. El registro es gratuito. Los datos son de un problema que se planteó previamente (y que se cerró en enero de 2019). Las puntuaciones y soluciones de los participantes no se hicieron públicas en la web. En esa URL, vamos a resources y tenemos todos los enlaces.
2. Primero nos interesa Dataset Description (172 K), es un documento con extensión tipo pdf que describe las variables (features). Vemos que existen dos ficheros, uno de sesiones y otro de canciones. Las sesiones tienen variables del comportamiento de un usuario dentro de su cuenta de Spotify, por ejemplo: id de sesión, si ha cambiado de contexto (si la ha cambiado de playlist, se llama context\_switch), si ha pausado entre canción y canción (no\_pause\_before\_play), etc.

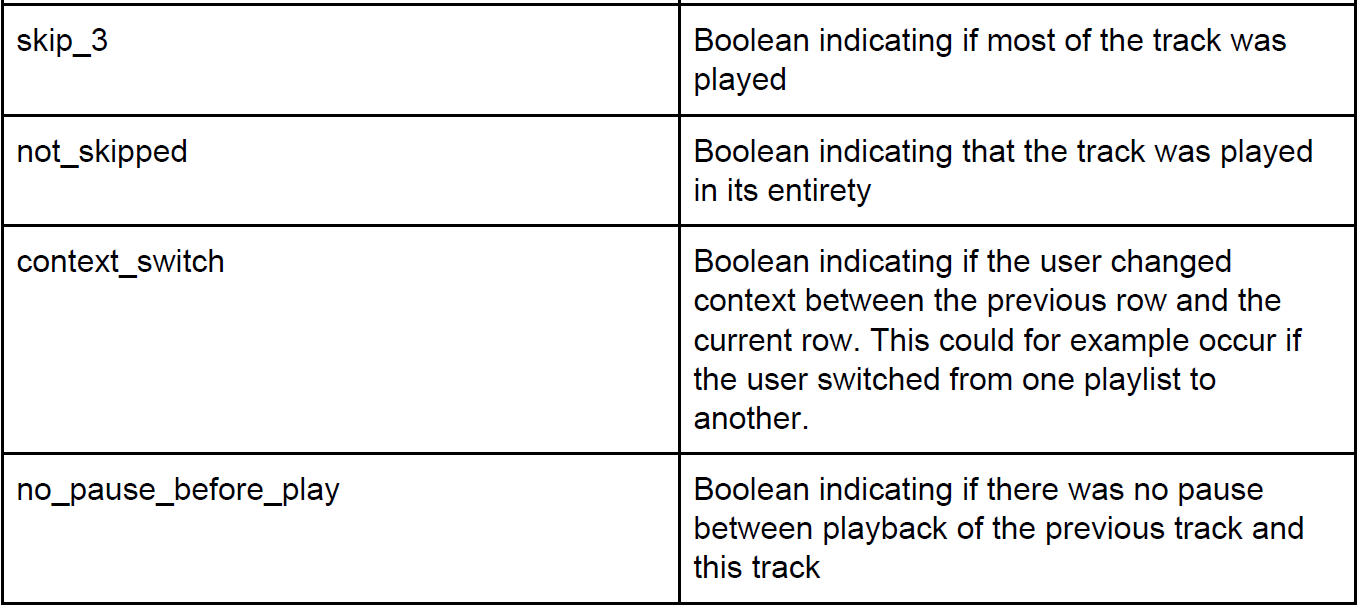
El otro fichero es el de canciones (tracks), que tiene variables propias de una canción: la duración, el año en el que se lanzó, etc.

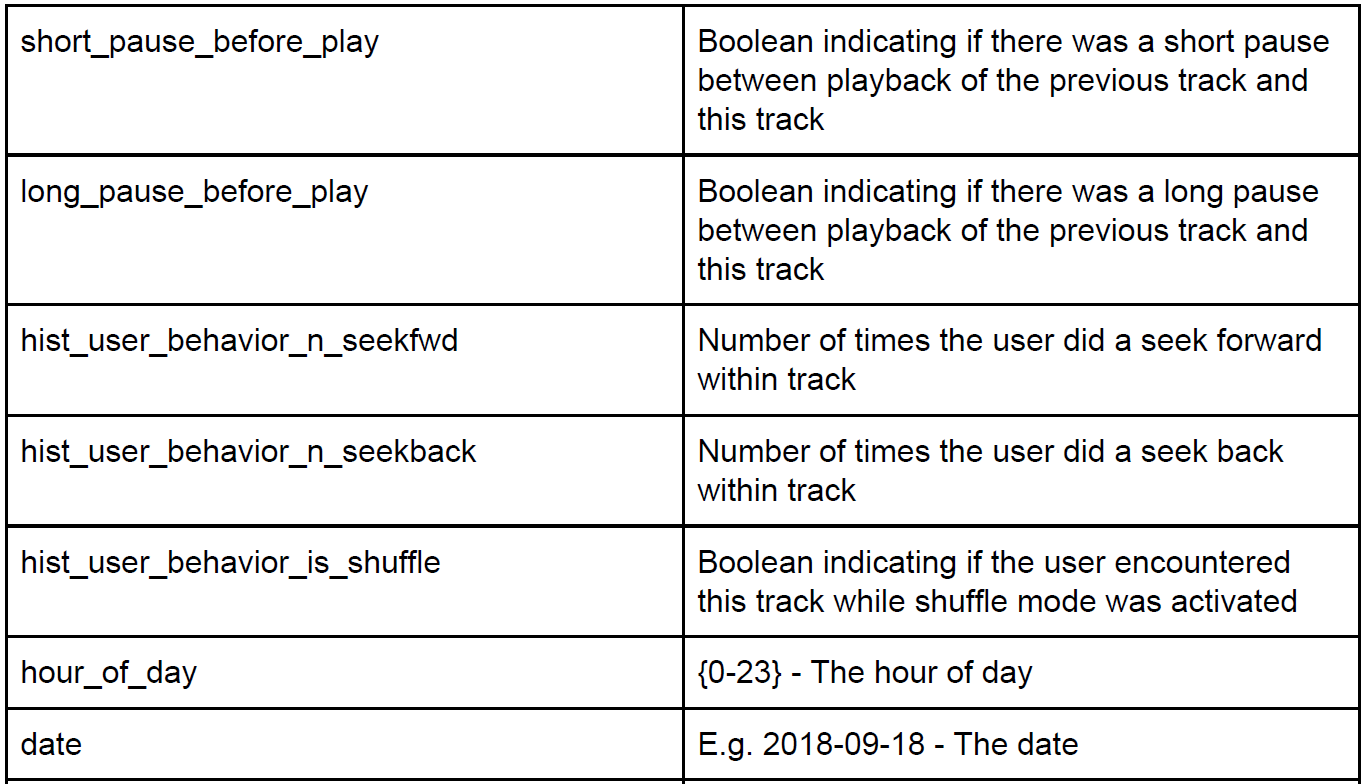
1. Ahora el siguiente paso es descargar el dataset para analizarlo. Aquí hay varias opciones: tenemos unos enlaces para descargar las partes que nos interesan, otro enlace descarga todo el dataset de entrenamiento (56 GB) y otro enlace decarga un dataset de entrenamiento "mini" de 17,2 MB, para probar en local. Como vamos a hacer estas pruebas en un contenedor Docker en local, elegimos el dataset "mini" de 17,2 MB, que es un muestreo del dataset grande.

En el enlace se descarga un documento con extensión ‘.tar.gz’ que contiene a su vez dos ficheros. Recomendamos utilizar 7zip para descomprimirlos. Como ya hemos visto del punto 2, un fichero corresponde con las sesiones de usuarios y otro con un listado de canciones.

Fichero de sesión:







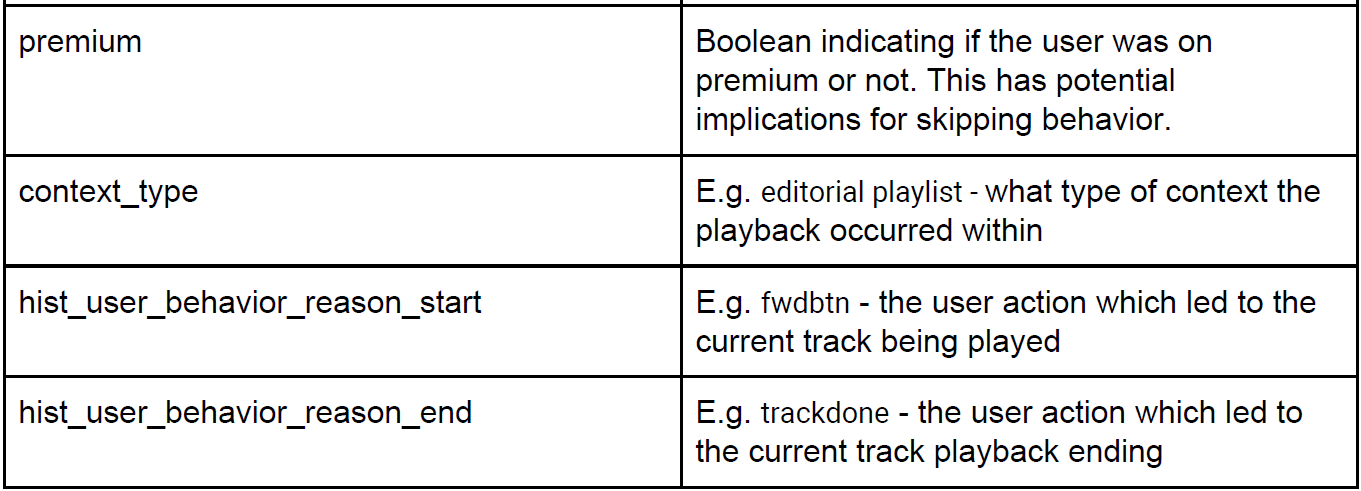
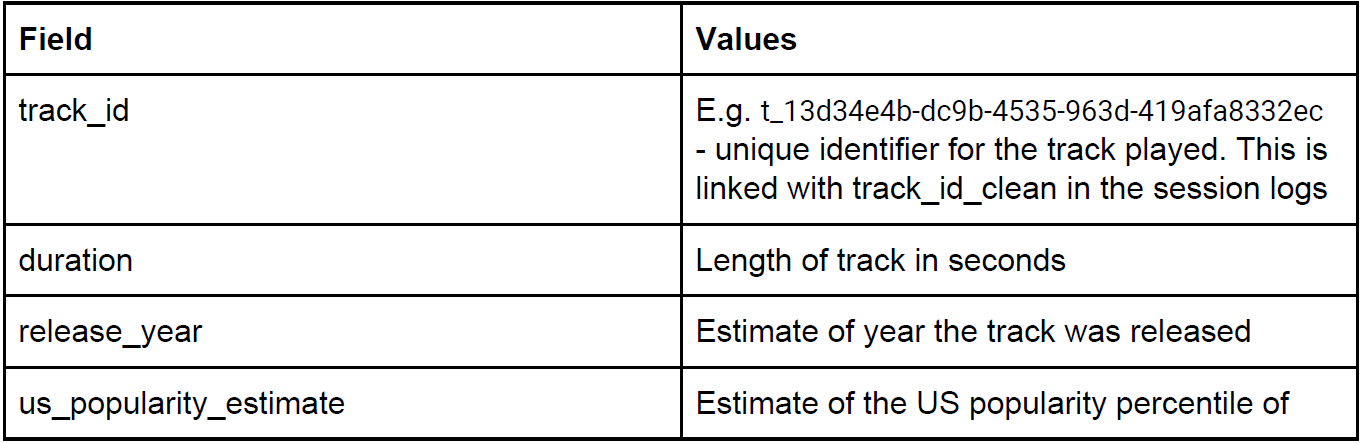
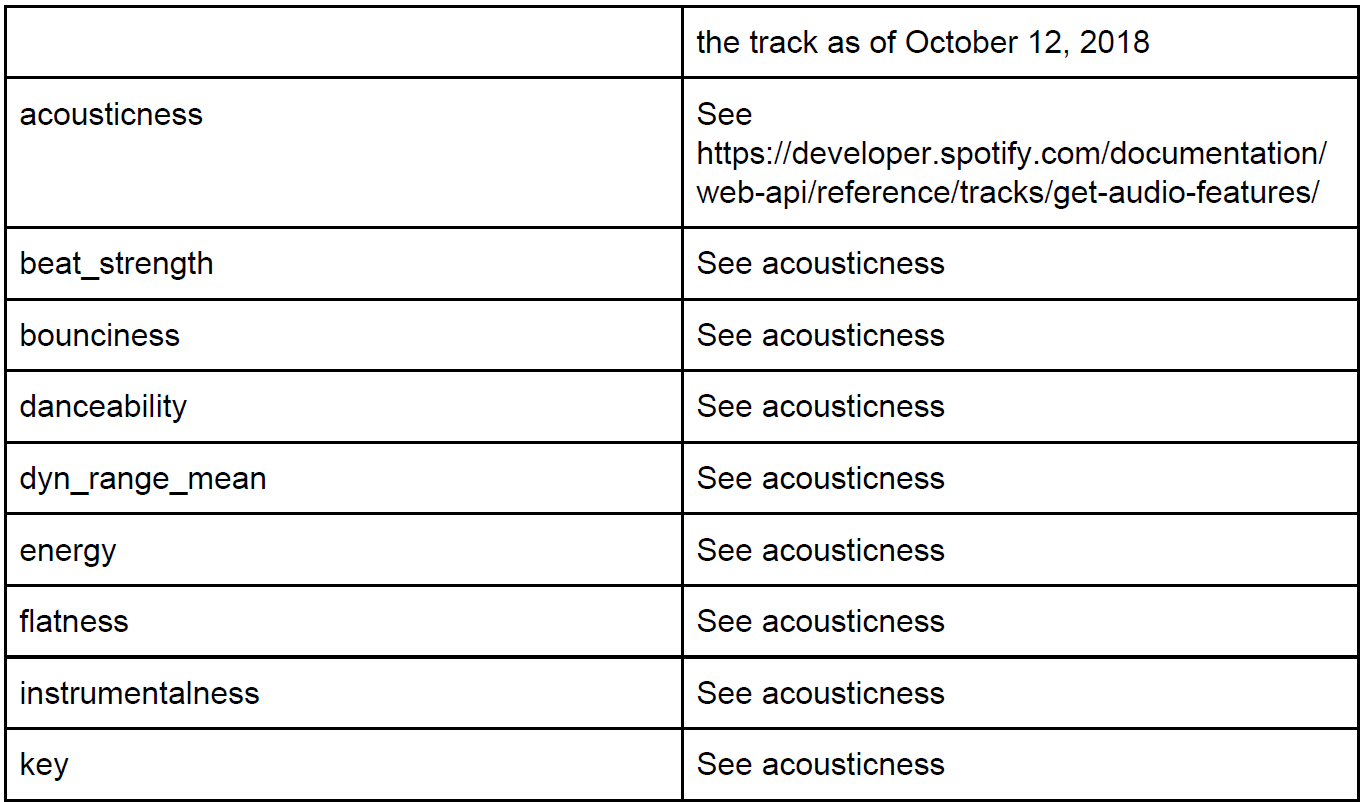
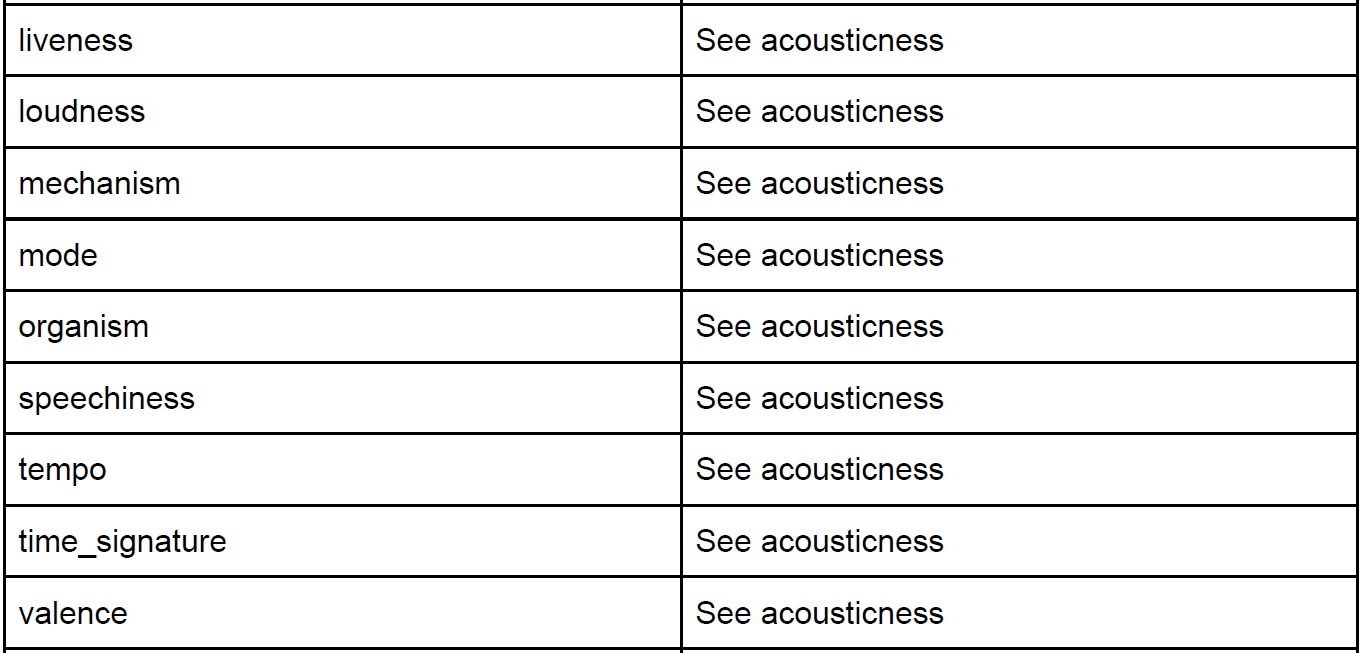


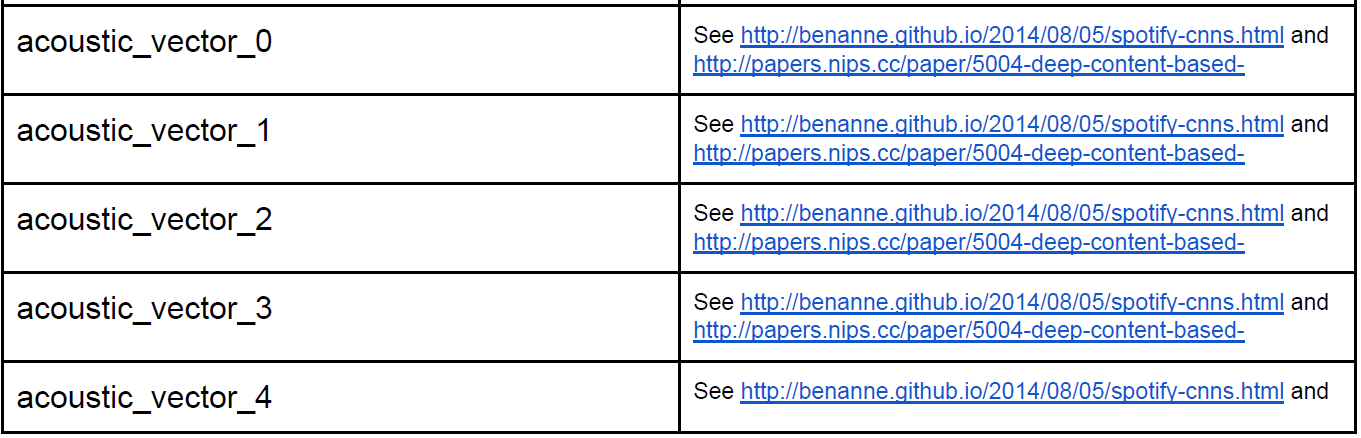
Tabla 1: Fichero de sesión en Spotify

Fichero de canción:









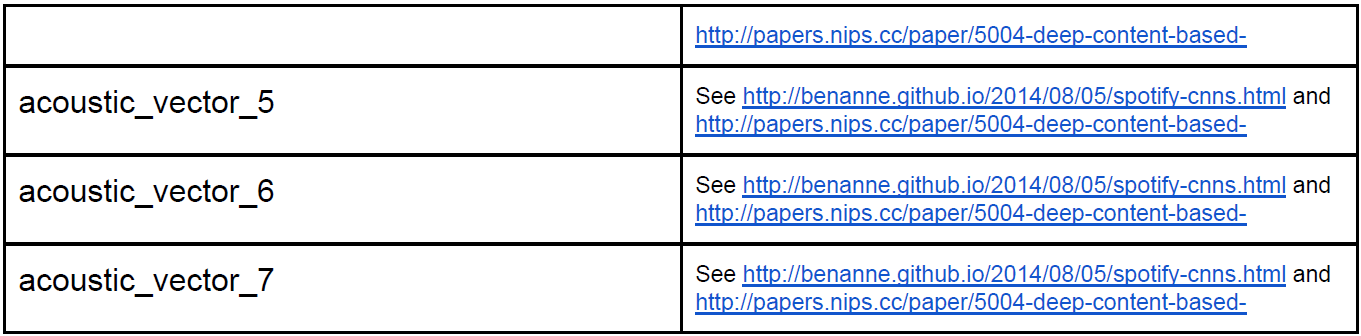


Tabla 2: Fichero de canción de Spotify

En caso de desear más detalle, las propias definiciones de las variables apuntan a GitHub. Las variables sintéticas fueron creadas dentro de un proyecto interno de Spotify usando redes neuronales.

Ya tenemos acceso a los datos de entrada, que son la base para empezar a estudiar el proyecto. En el siguiente capítulo empezamos con el análisis exploratorio.

# 3. Análisis exploratorio

Dentro de un análisis exploratorio siempre suele haber sucesivos pasos que van puliendo y transformando los datos de entrada para que sean útiles como datos de entrada de algún modelo.

Sin embargo, el análisis exploratorio no es siempre idéntico, pues depende de la propia naturaleza del proyecto. Según la cantidad de datos, el tipo de datos, el ruido en los datos, los datos faltantes, los datos duplicados…, según cada caso, las decisiones que se van tomando varían. Lo importante es ir entendiendo los datos para extraer información de ellos y de esta forma que se puedan adaptar correctamente a la entrada del modelo.

De manera inicial, empezaremos leyendo todos los datos.

Importamos las librerías en las que nos apoyamos. Vamos a programar en Python 3 (Python3, 2020), que es el lenguaje de referencia en el proyecto.

Vamos a explicar las librerías en las que nos apoyaremos.

Hay librerías numéricas (Numpy), de Dataframes (Pandas), científicas (SciPy, Stat), de preprocesado (Preprocessing, MinMaxScaler, SelectKBest, Chi2), de visualización (Seaborn, PyPlot), de pipeline (Pipeline), para entrenamiento y test (Train\_test\_split), para validación cruzada (Cross\_val\_score, GridSearchCV, StratifiedKFold), de métricas (Confusion\_matrix, Classification\_report, roc\_auc\_score).

También usaremos las librerías propias de cada modelo, pues nuestra idea es explorar para entender el estado inicial, preprocesar los datos para mejorar la extracción de información y adecuarlos al estudio, volver a explorarlos para ir descubriendo qué información ya es accesible desde los datos, y procesarlos de nuevo para tener un estadio previo a la entrada del modelo, donde introduciremos los datos en un banco de modelos y elegiremos el mejor en función de las métricas. Probaremos distintos modelos que tienen aplicación en el campo de la predicción.

Todo esto se va a explicar paso a paso.

Leemos las canciones, usamos la decodificación utf-8 para leer los caracteres españoles y vemos qué atributos tiene.

df\_track\_features.columns

Index(['track\_id', 'duration', 'release\_year', 'us\_popularity\_estimate','acousticness', 'beat\_strength', 'bounciness', 'danceability','dyn\_range\_mean', 'energy', 'flatness', 'instrumentalness', 'key', 'liveness', 'loudness', 'mechanism', 'mode', 'organism', 'speechiness', 'tempo', 'time\_signature', 'valence', 'acoustic\_vector\_0', 'acoustic\_vector\_1', 'acoustic\_vector\_2', 'acoustic\_vector\_3', 'acoustic\_vector\_4', 'acoustic\_vector\_5', 'acoustic\_vector\_6', 'acoustic\_vector\_7'], dtype='object')

Observamos que las variables que tenemos son las mismas que existían en la lista de variables que hemos definido en el fichero de canción del punto 2. **Así nos aseguramos de que no hemos perdido ninguna fuente.**

Leemos las sesiones y vemos qué atributos tiene.

df\_training\_set.columns

Index(['session\_id', 'session\_position', 'session\_length', 'track\_id\_clean', 'skip\_1', 'skip\_2', 'skip\_3', 'not\_skipped', 'context\_switch', 'no\_pause\_before\_play', 'short\_pause\_before\_play', 'long\_pause\_before\_play', 'hist\_user\_behavior\_n\_seekfwd', 'hist\_user\_behavior\_n\_seekback', 'hist\_user\_behavior\_is\_shuffle', 'hour\_of\_day', 'date', 'premium', 'context\_type', 'hist\_user\_behavior\_reason\_start', 'hist\_user\_behavior\_reason\_end'], dtype='object')

Observamos que las variables que tenemos son las mismas que existían en la lista de variables que hemos definido **en el fichero de sesión de usuario del punto anterior.**

Vamos a programar un predictor para saber si dada una sesión, el usuario va a escuchar toda la canción. La variable que queremos predecir (target, como se suele denominar en el mundo Data Science) es not\_skipped, es decir, que el usuario ha escuchado toda la canción. Los estudios que hagamos aquí, también se podrían aplicar a skip\_1, a skip\_2 y a skip\_3, que son variables también binarias que miden si la canción se ha dejado de escuchar nada más comenzar la canción (skip\_1), cuando lleva unos segundos (skip\_2) o hacia el final de la canción, pero sin acabar de escucharse entera (skip\_3).

Exploramos, queremos ver todas las columnas de todas las tablas fuente (para más detalle se puede consultar el notebook en adjuntos a esta memoria).

Ahora ya tenemos una idea del tipo de datos almacenado, conociendo además del tipo del dato, los valores que presentan.

Comprobamos a continuación el tamaño de la tabla de canciones.

df\_track\_features.shape

(50704, 30)

Vemos que en canciones tenemos 50704 muestras (filas) con 30 atributos (columnas).

Comprobamos ahora el tamaño de la tabla de sesión.

df\_training\_set.shape

(167880, 21)

Vemos que en las sesiones de usuario tenemos 167880 muestras (filas) con 21 atributos (columnas).

Miramos los tipos que aparecen en la tabla de canciones.

Así hemos visto la diferencia en la precisión de cada tipo de dato en la tabla de canciones. Comprobamos el track\_id para conocer su aspecto, el año para ver exactamente cómo es el formato de fecha, el tono para ver cuál es el conjunto que lo define, así como el modo.

Debemos observar que aquí se debe acceder al atributo con corchetes y no con punto para evitar ejecutar la moda, que en inglés se dice igual y es una palabra del lenguaje.

Vemos que hay dos modos: mayor y menor.

Comprobamos los tiempos, que son el compás musical. El resto de los campos son claramente mediciones numéricas relacionadas con la canción.

Miramos los tipos de la sesión para conocer esta información, al igual que anteriormente hemos consultado los tipos de la tabla de canciones.

En este momento ya conocemos los tipos de las variables. No obstante, todavía desconocemos si existen valores nulos en las variables, puesto que ese dato no viene explícito en el tipo del dato, además de que el realizar un volcado de las primeras y de las últimas filas no muestra la existencia de atributos con valores nulos.

Estudiamos también los nulos en la tabla de canciones y vemos que no existen nulos entre los datos de la tabla de canciones.

Respecto al balanceo de clases, resumimos la distribución de instancias entre clases en nuestro dataset. En este caso, solo nos interesa el dataframe de sesiones.

not\_skipped

False 111996

True 55884

dtype: int64

Tabla 12: Balanceo de clases

Estudiamos ahora la proporción de target positivo.

0.33288062902072907

Estudiamos la proporción de target negativo.

0.6671193709792709

Observamos que las clases **no** están balanceadas.

En la siguiente sección vamos a estudiar las relaciones que existen entre las variables.

Analizamos parejas de correlaciones de Pearson, puesto que queremos ver todas las columnas de las correlaciones dentro de la tabla de canciones.

Observamos el nivel de correlación entre las variables. Por ejemplo, vemos que casi todos los datos son de usuarios premium, puesto que la correlación es prácticamente unitaria. Esto implica que casi todos los datos provienen de sesiones de usuarios con un servicio premium, lo que es lógico, puesto que así se pueden conocer muchas más variables de entrada (al tener que registrarse el usuario, además de que ya no es únicamente una cuestión de cantidad de variables, sino que Spotify lo que busca es que un usuario premium esté satisfecho con el servicio).

Las correlaciones ofrecen mucha información, en secciones posteriores representaremos esta información de una manera más visual.

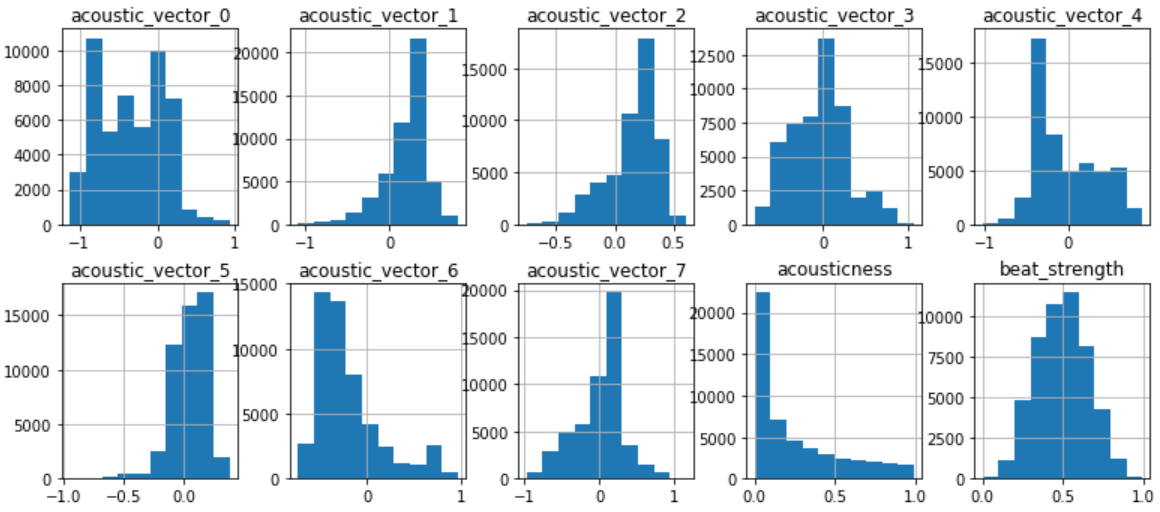
Sin embargo, antes de entrar en representaciones más visuales, vamos a estudiar en la siguiente sección la asimetría estadística para conocer qué tan simétricos son nuestros datos de partida.

Estudiamos la asimetría estadística por cada atributo en la tabla de canciones.

La asimetría también se puede apreciar observando los histogramas de las variables (las pendientes que caen a cada lado de los valores centrales de las funciones de densidad de probabilidad).

En la siguiente sección vamos a pintar los histogramas para entrar en la representación gráfica de las variables. Así podemos intuir también la forma que tienen las funciones de densidad de probabilidad, que se conseguirían calculando los histogramas para una población infinita y normalizada.

Miramos los histogramas de la tabla de canciones.



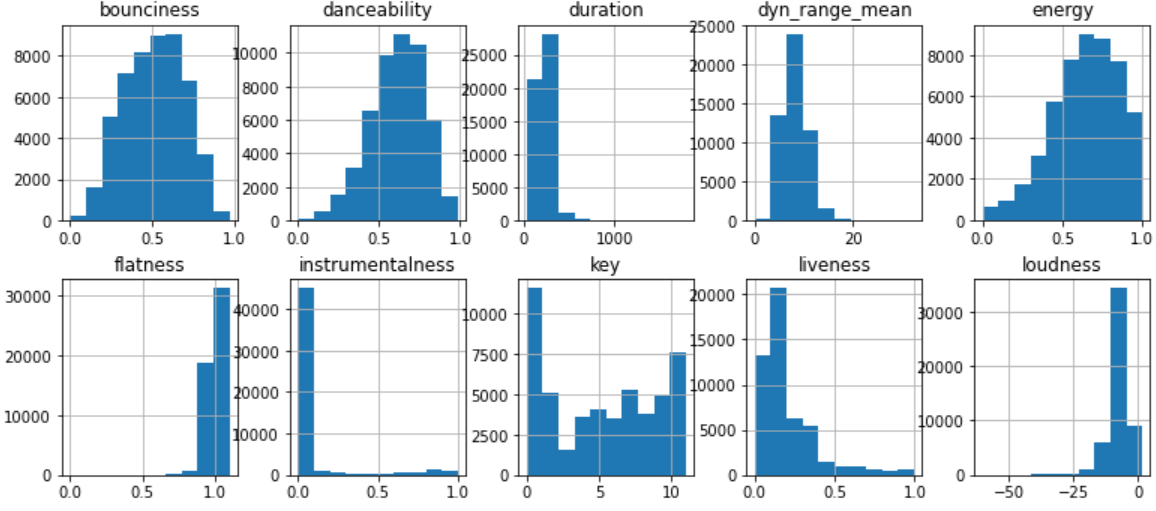
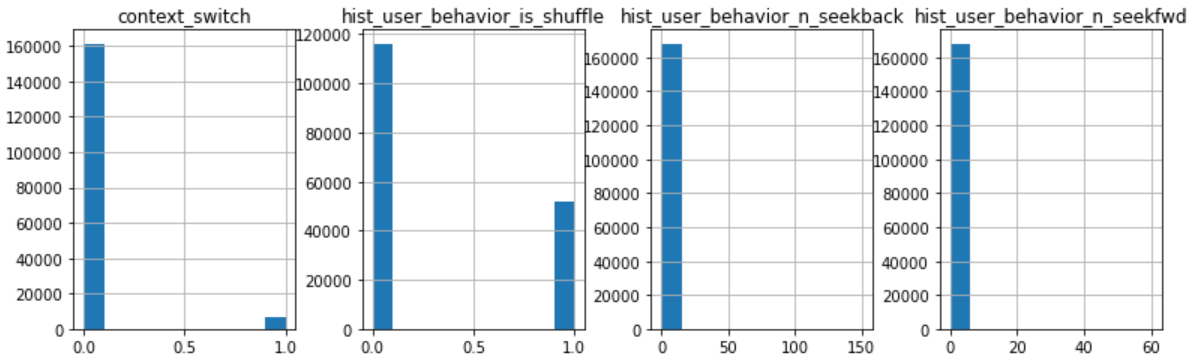
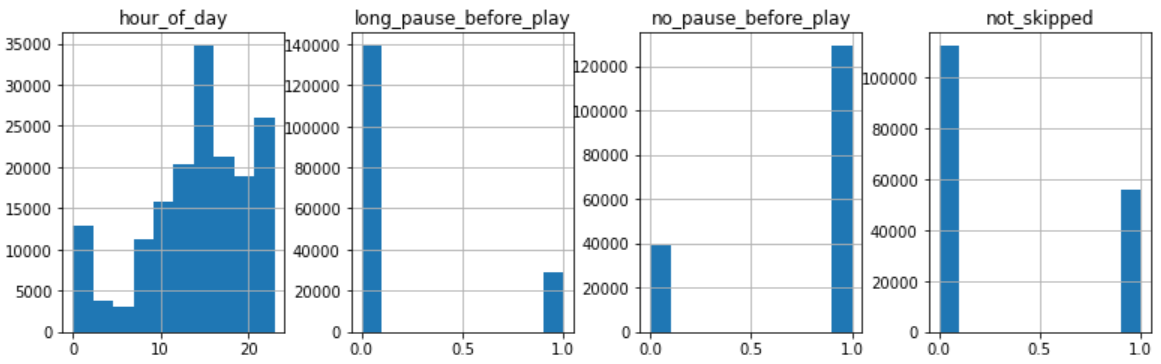


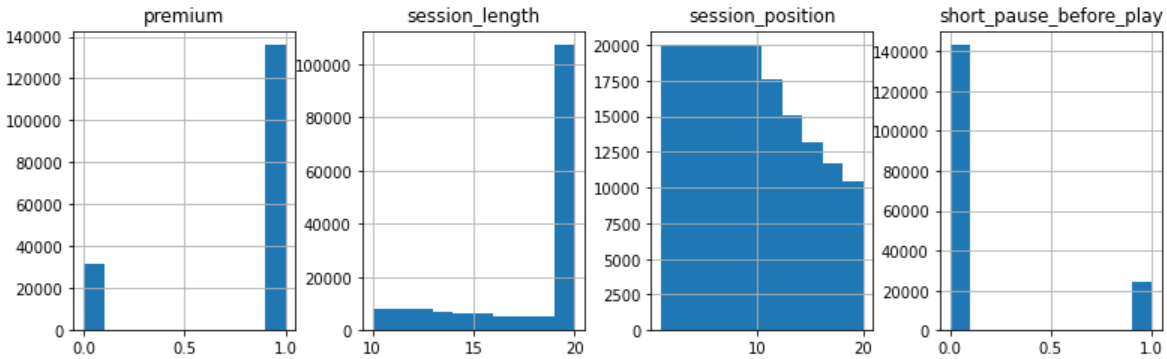


Ilustración 5: Histogramas de la tabla de canciones

Cambiamos el formato para la matriz de dispersión.







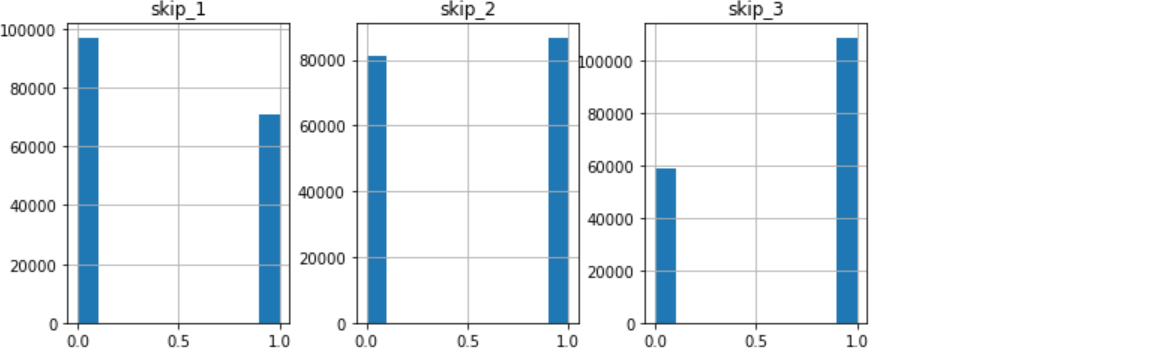


Ilustración 6: Histogramas de la tabla de sesión

Observamos los valores centrales, la distribución de los datos y las variables binarias.

El siguiente paso natural es estudiar los diagramas de densidad para extrapolar desde los histogramas hacia las funciones de densidad de probabilidad.

Diagramas de densidad univariante de la tabla de canciones.

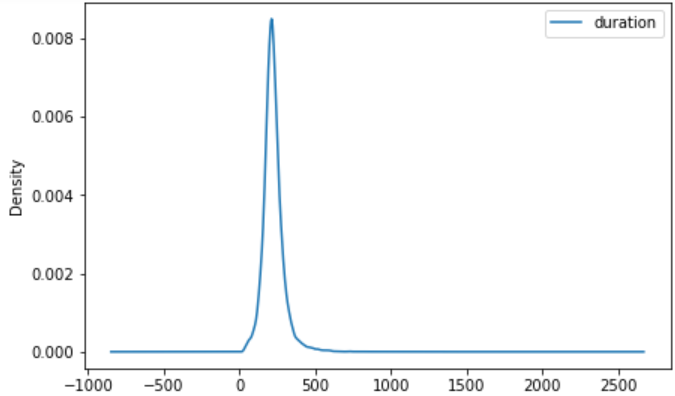


Ilustración 7: Densidad en tabla de canciones - duration

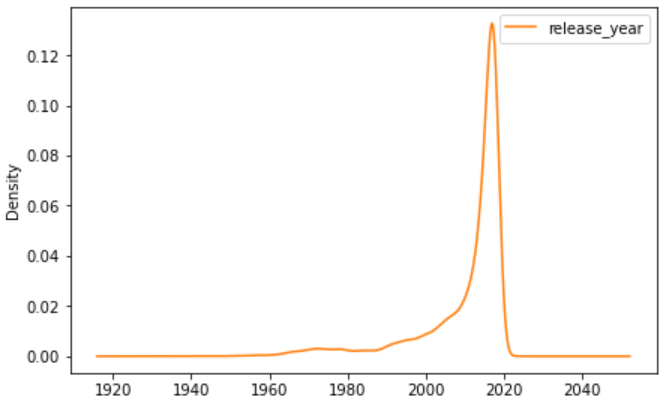


Ilustración 8: Densidad en tabla de canciones - release\_year

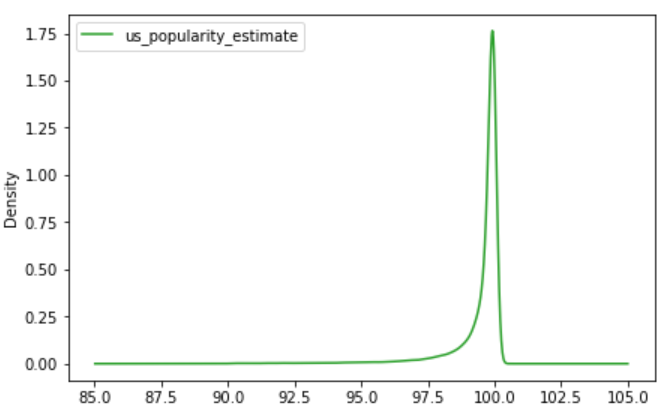


Ilustración 9: Densidad en tabla de canciones - us\_popularity\_estimate

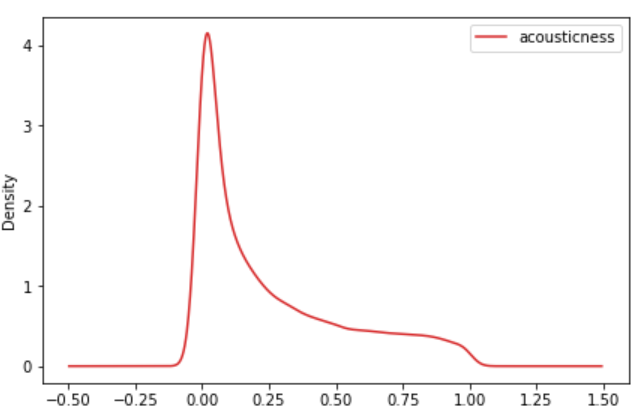


Ilustración 10: Densidad en tabla de canciones - acousticness

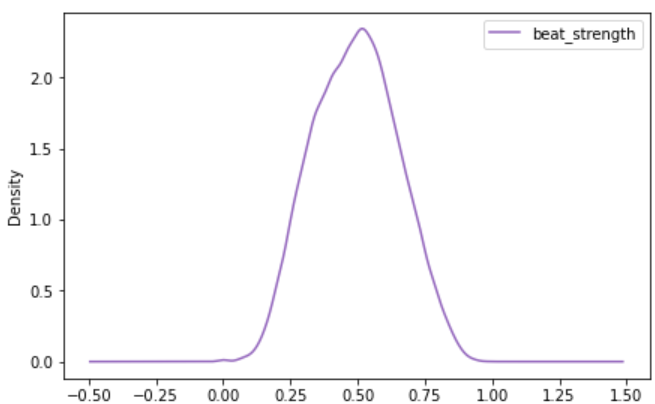


Ilustración 11: Densidad en tabla de canciones - beat\_strength

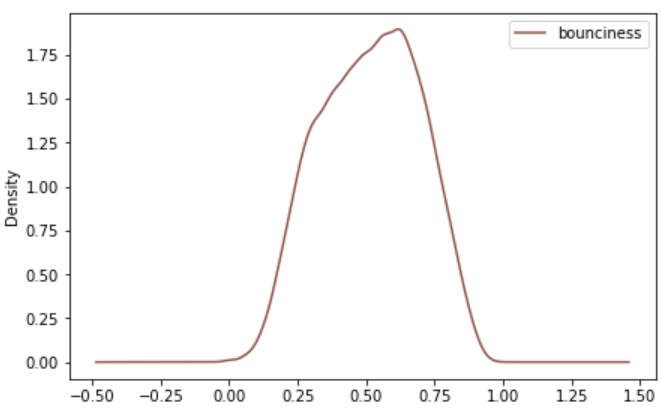


Ilustración 12: Densidad en tabla de canciones - bounciness

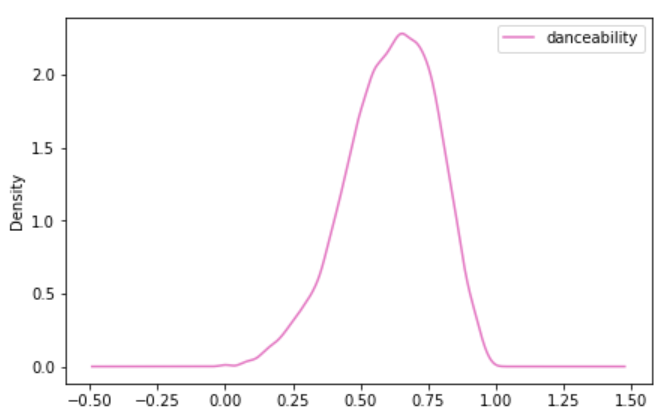


Ilustración 13: Densidad en tabla de canciones - danceability

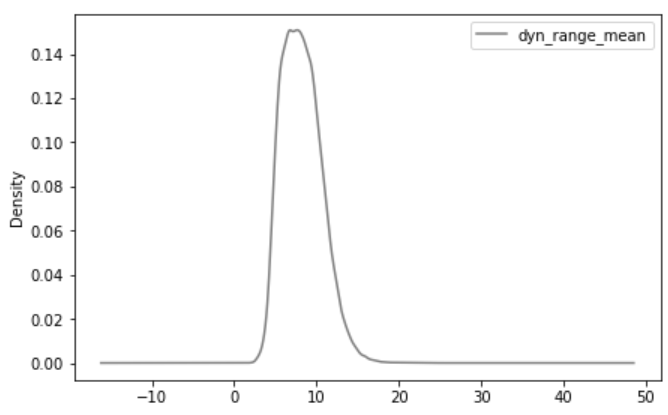


Ilustración 14: Densidad en tabla de canciones - dyn\_range\_mean

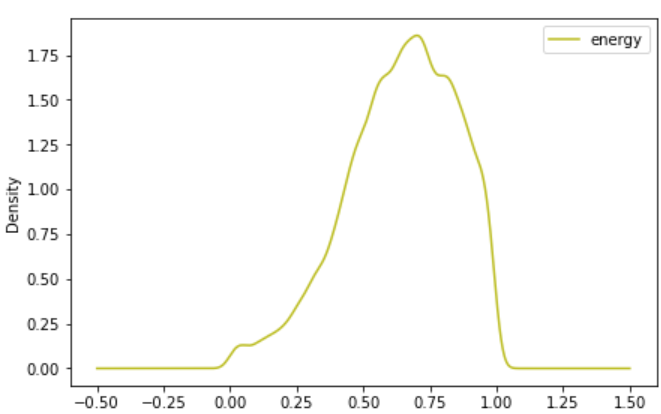


Ilustración 15: Densidad en tabla de canciones - energy

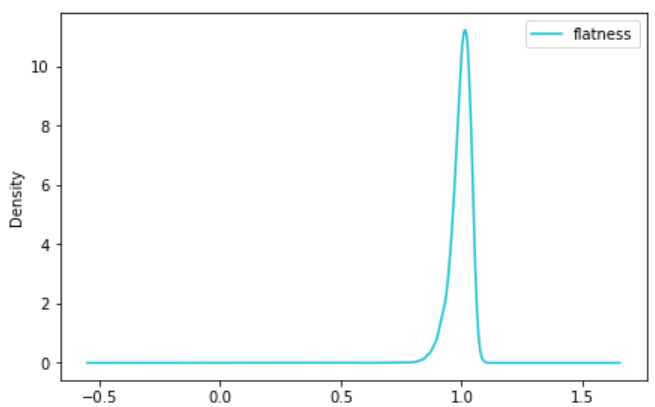


Ilustración 16: Densidad en tabla de canciones - flatness

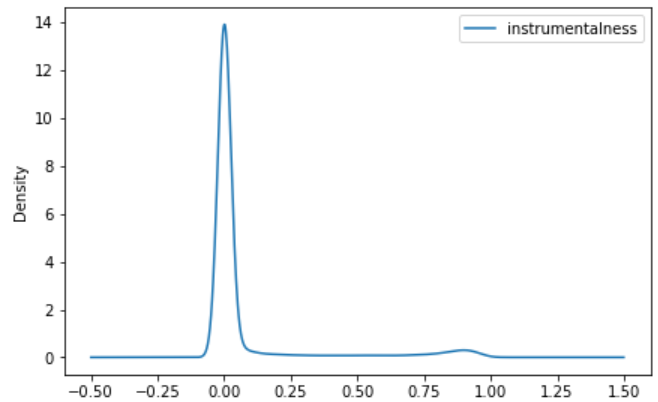


Ilustración 17: Densidad en tabla de canciones - instrumentalness

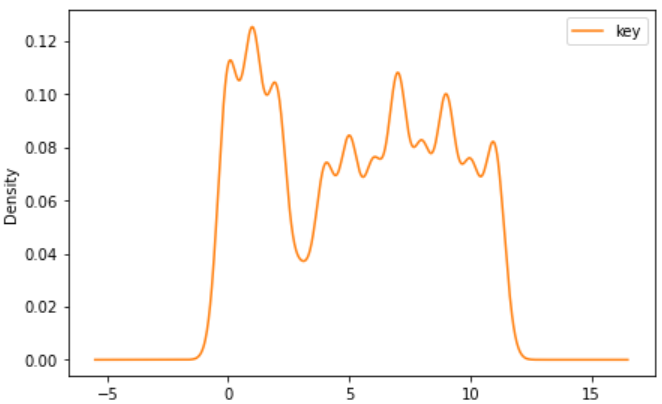


Ilustración 18: Densidad en tabla de canciones - key

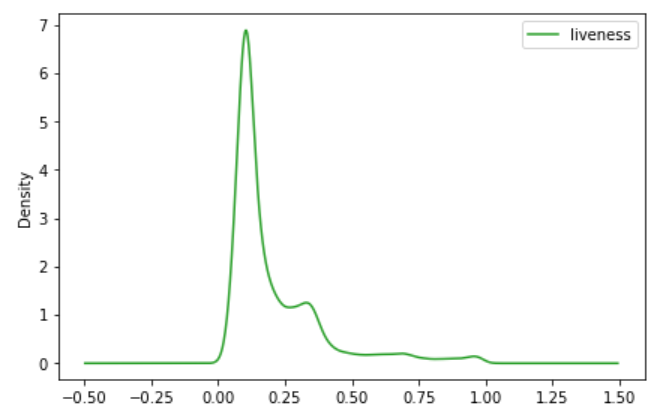


Ilustración 19: Densidad en tabla de canciones - liveness

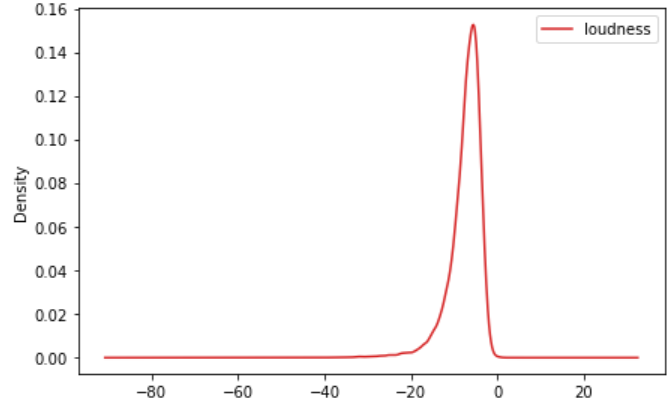


Ilustración 20: Densidad en tabla de canciones - loudness

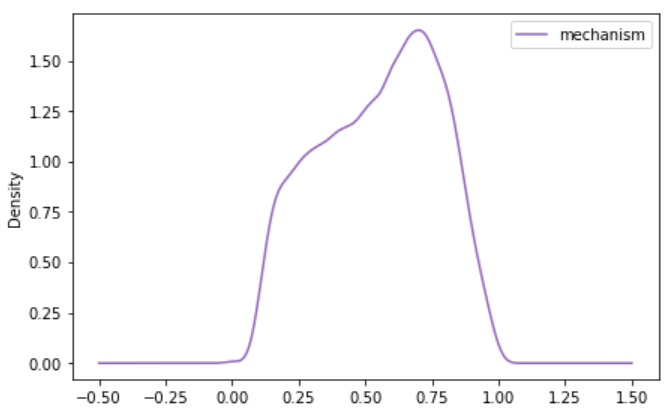


Ilustración 21: Densidad en tabla de canciones - mechanism

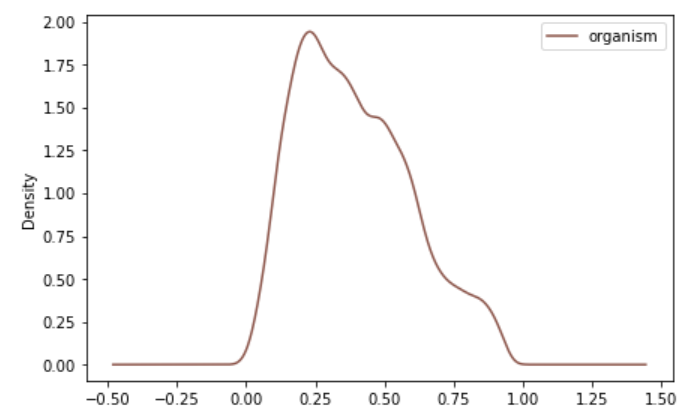


Ilustración 22: Densidad en tabla de canciones - organism

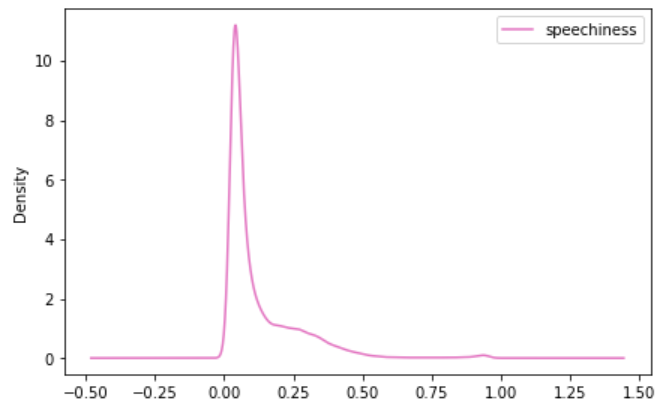


Ilustración 23: Densidad en tabla de canciones - speechiness

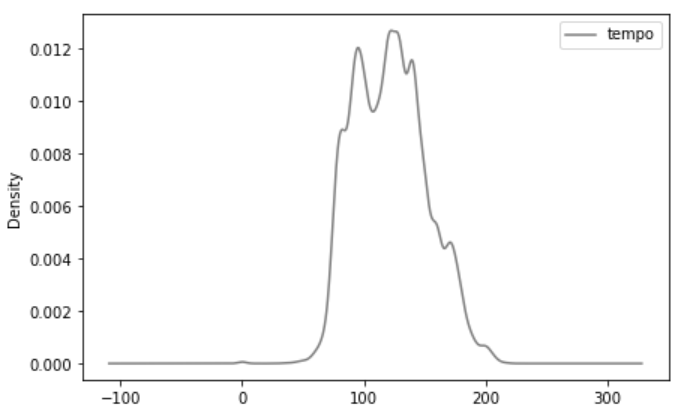


Ilustración 24: Densidad en tabla de canciones - tempo

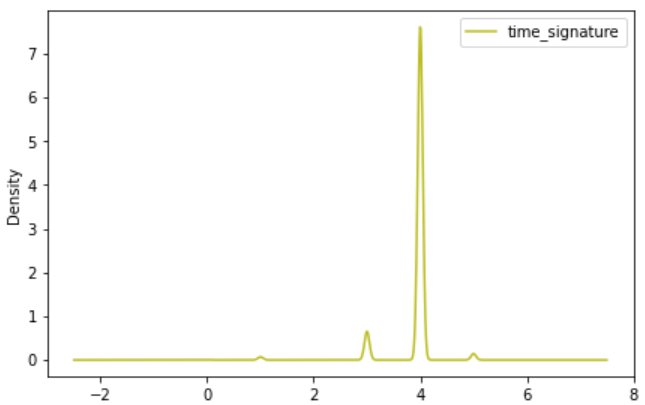


Ilustración 25: Densidad en tabla de canciones - time\_signature

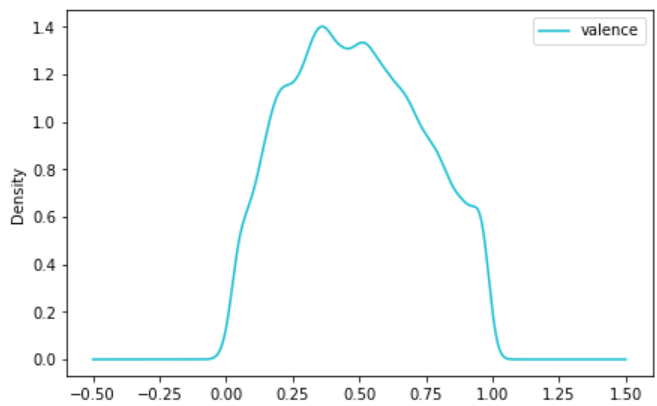


Ilustración 26: Densidad en tabla de canciones - valence

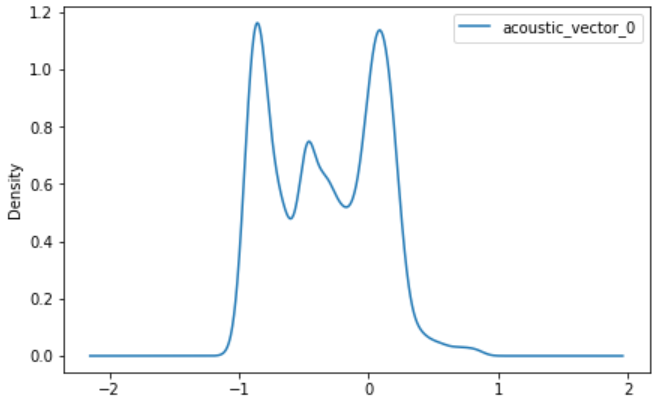


Ilustración 27: Densidad en tabla de canciones - acoustic\_vector\_0

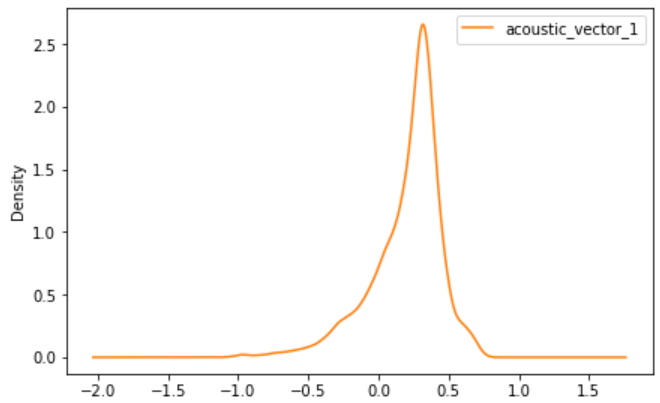


Ilustración 28: Densidad en tabla de canciones - acoustic\_vector\_1

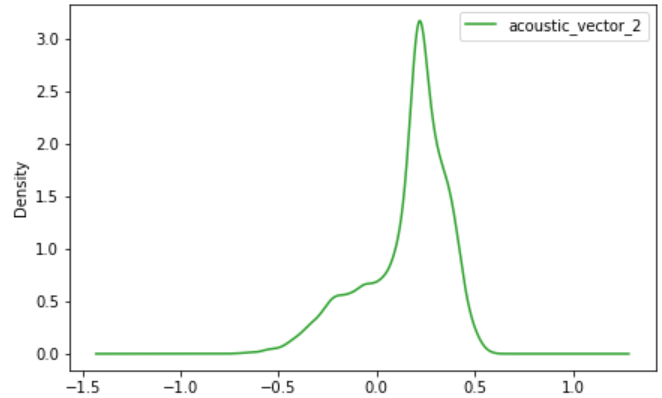


Ilustración 29: Densidad en tabla de canciones - acoustic\_vector\_2

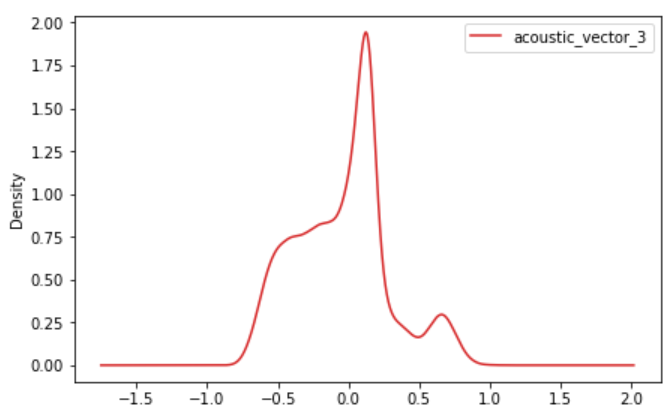


Ilustración 30: Densidad en tabla de canciones - acoustic\_vector\_3

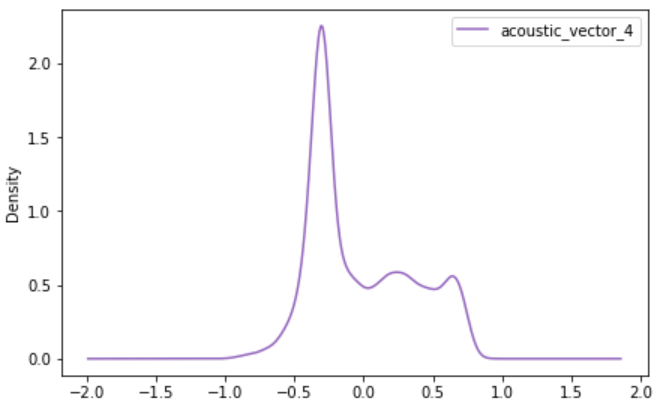


Ilustración 31: Densidad en tabla de canciones - acoustic\_vector\_4

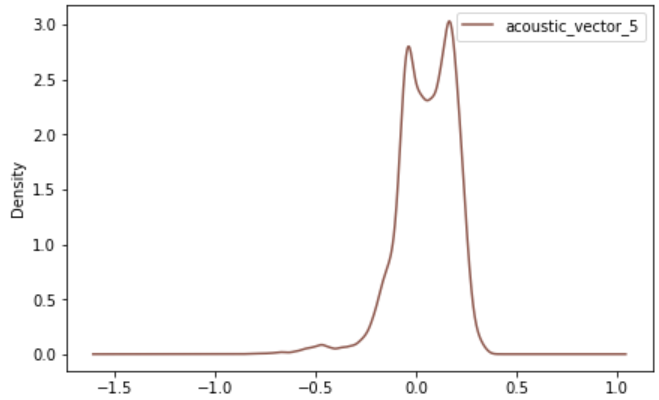


Ilustración 32: Densidad en tabla de canciones - acoustic\_vector\_5

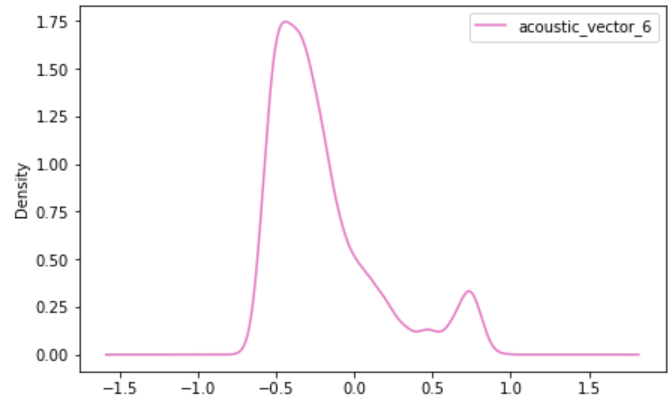


Ilustración 33: Densidad en tabla de canciones - acoustic\_vector\_6

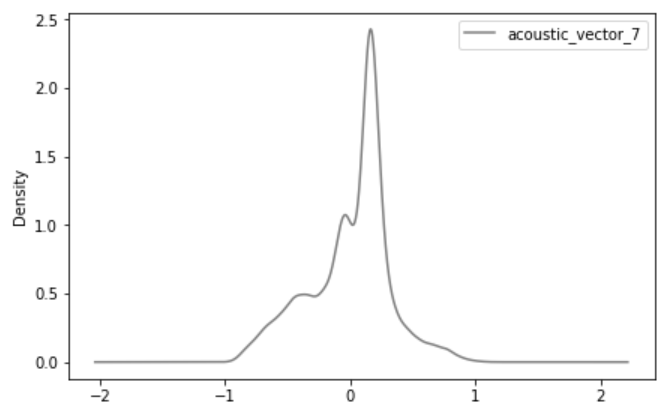


Ilustración 34: Densidad en tabla de canciones - acoustic\_vector\_7

Diagramas de densidad univariante de la tabla de sesión.

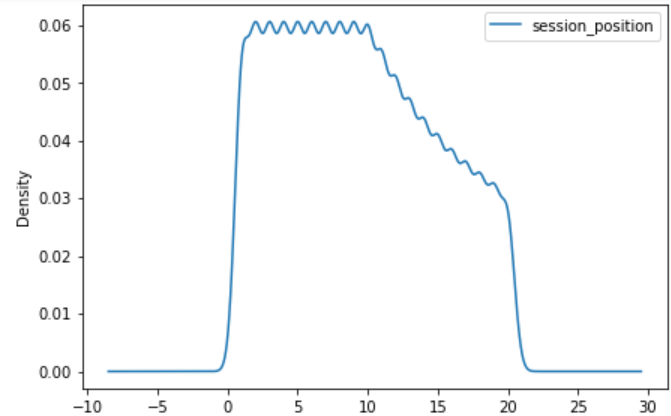


Ilustración 35: Densidad en tabla de sesión - session\_position

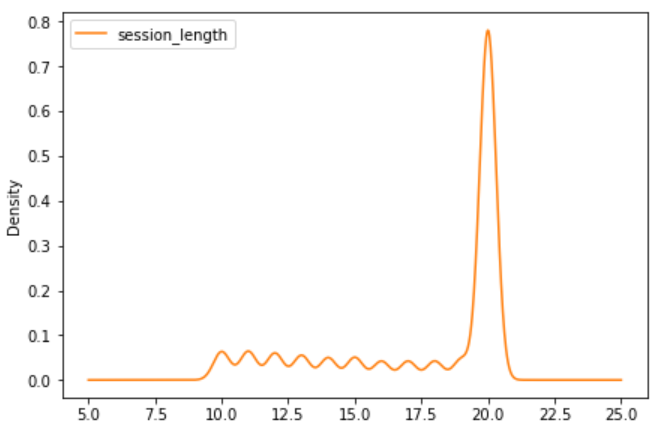


Ilustración 36: Densidad en tabla de sesión - session\_length

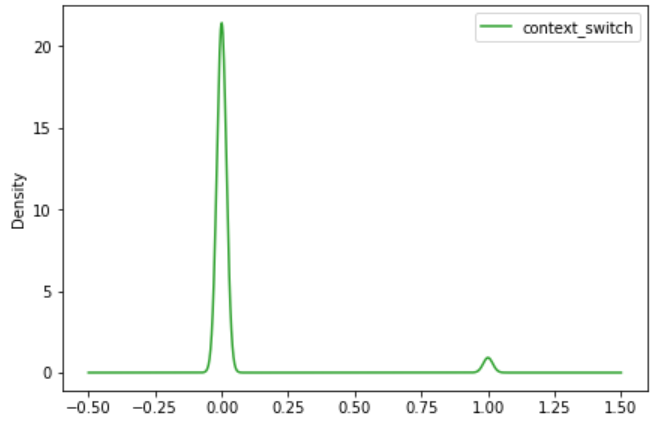


Ilustración 37: Densidad en tabla de sesión - context\_switch

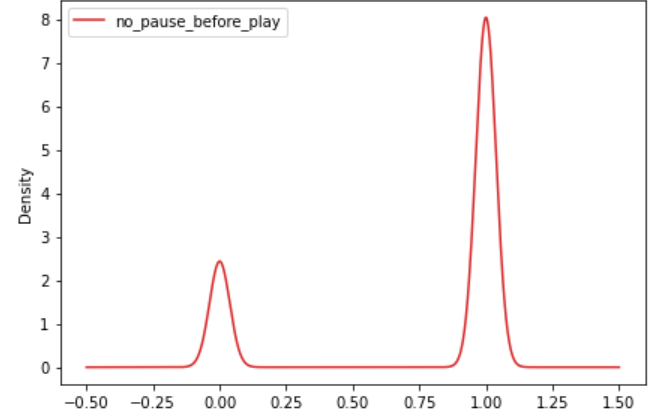


Ilustración 38: Densidad en tabla de sesión - no\_pause\_before\_play

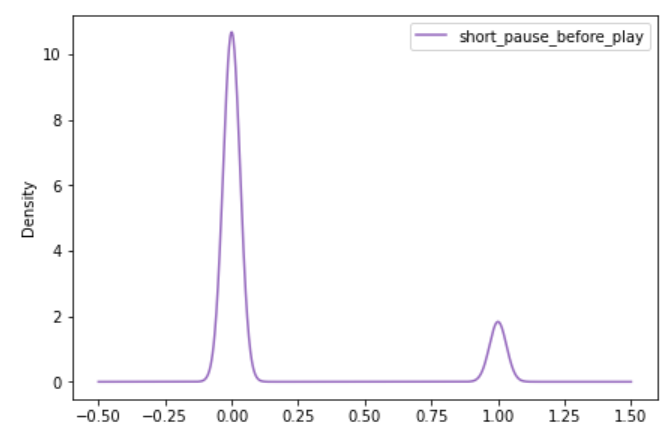


Ilustración 39: Densidad en tabla de sesión - short\_pause\_before\_play

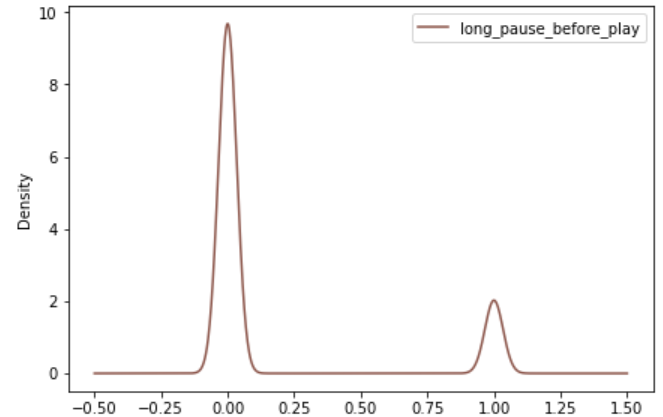


Ilustración 40: Densidad en tabla de sesión - long\_pause\_before\_play

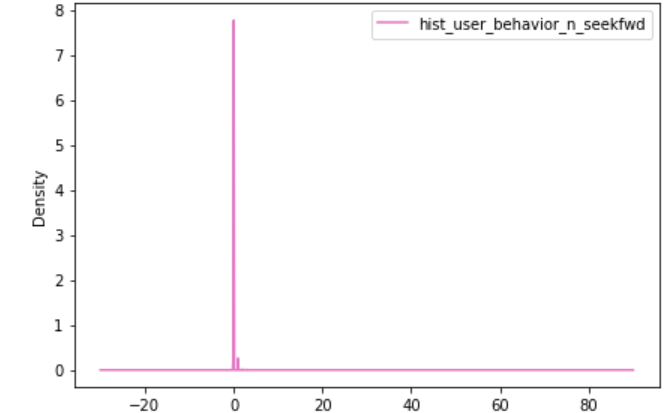


Ilustración 41: Densidad en tabla de sesión - hist\_user\_behavior\_n\_seekfwd

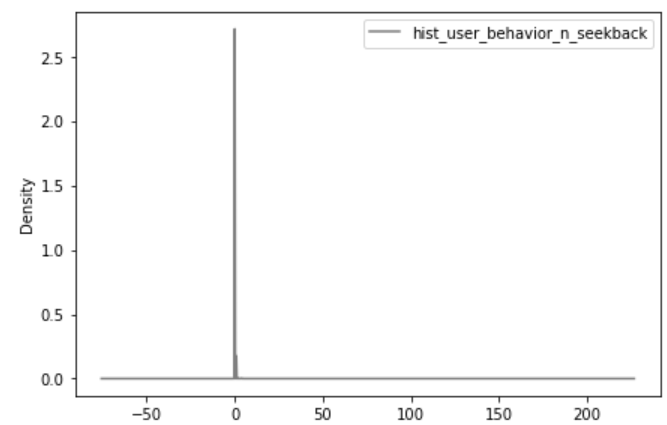


Ilustración 42: Densidad en tabla de sesión - hist\_user\_behavior\_n\_seekback

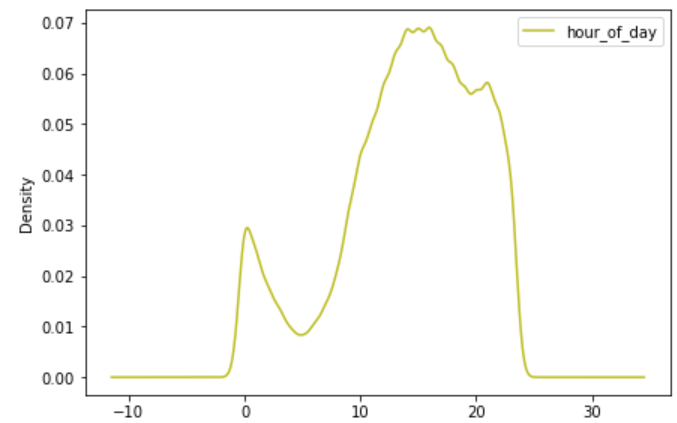


Ilustración 43: Densidad en tabla de sesión - hour\_of\_day

En esta sección hemos podido ver gráficamente cómo se distribuyen los datos.

Para conocer los outliers (valores anómalos por su baja aparición en la distribución) se puede recurrir a los diagramas de densidad, pero es más fácil tratarlos con los diagramas de cajas, pues estos los marcan muy claramente.

En la siguiente sección vamos a estudiar los diagramas de cajas, con el objetivo de poder filtrar posteriormente los outliers.

Diagrama de caja y bigotes de la tabla de canciones.

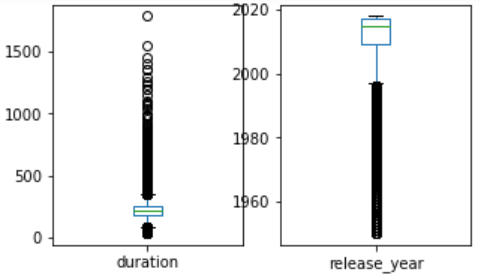


Ilustración 44: Diagrama de cajas en canciones - duration, release\_year

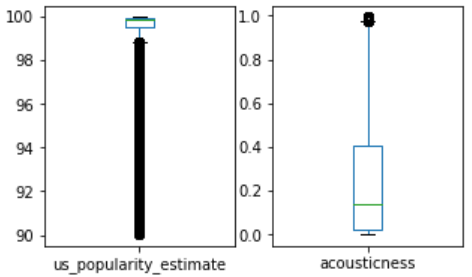


Ilustración 45: Diagrama de cajas en canciones - us\_popularity\_estimate, acousticness

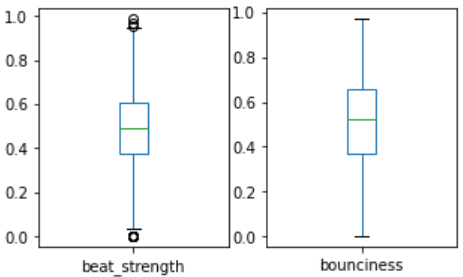


Ilustración 46: Diagrama de cajas en canciones - beat\_strength, bounciness

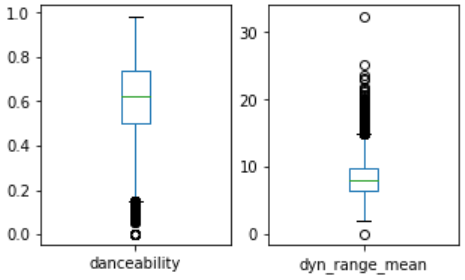


Ilustración 47: Diagrama de cajas en canciones - danceability, dyn\_range\_mean

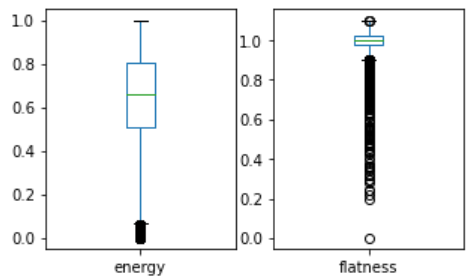


Ilustración 48: Diagrama de cajas en canciones - energy, flatness

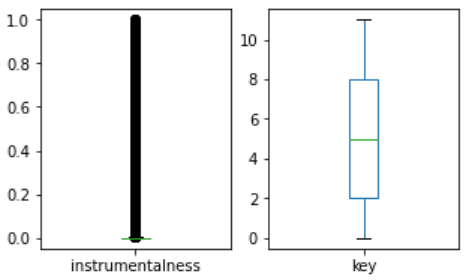


Ilustración 49: Diagrama de cajas en canciones - instrumentalness, key

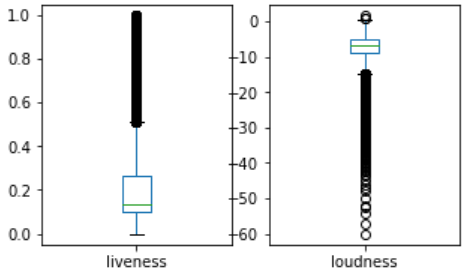


Ilustración 50: Diagrama de cajas en canciones - liveness, loudness

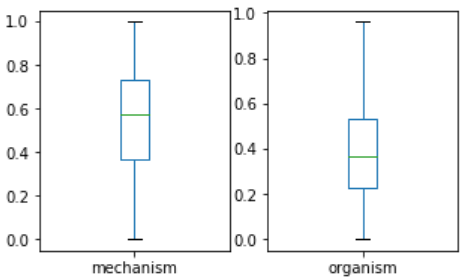


Ilustración 51: Diagrama de cajas en canciones - mechanism, organism

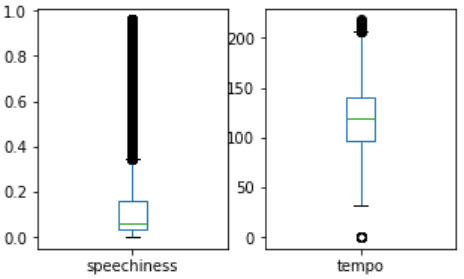


Ilustración 52: Diagrama de cajas en canciones - speechniness, tempo

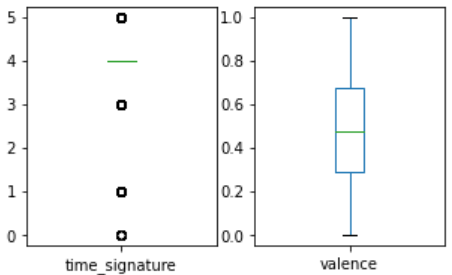


Ilustración 53: Diagrama de cajas en canciones - time\_signature, valence

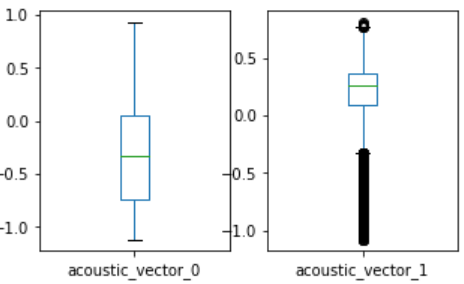


Ilustración 54: Diagrama de cajas en canciones - acoustic\_vector\_0, acoustic\_vector\_1

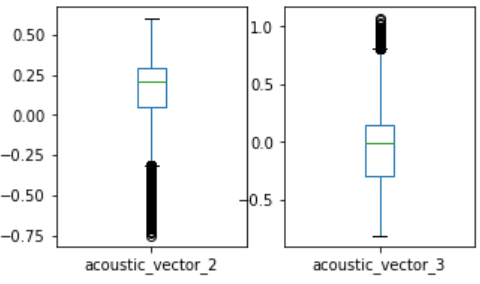


Ilustración 55: Diagrama de cajas en canciones - acoustic\_vector\_2, acoustic\_vector\_3

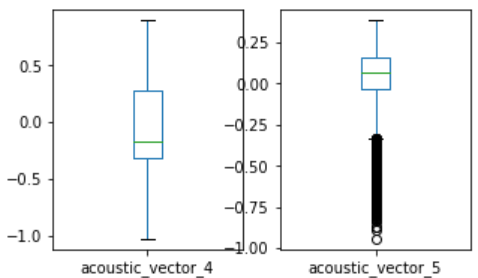


Ilustración 56: Diagrama de cajas en canciones - acoustic\_vector\_4, acoustic\_vector\_5

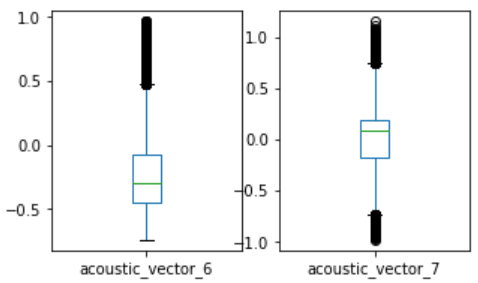


Ilustración 57: Diagrama de cajas en canciones - acoustic\_vector\_6, acoustic\_vector\_7

Diagrama de caja y bigotes de la tabla de sesión.

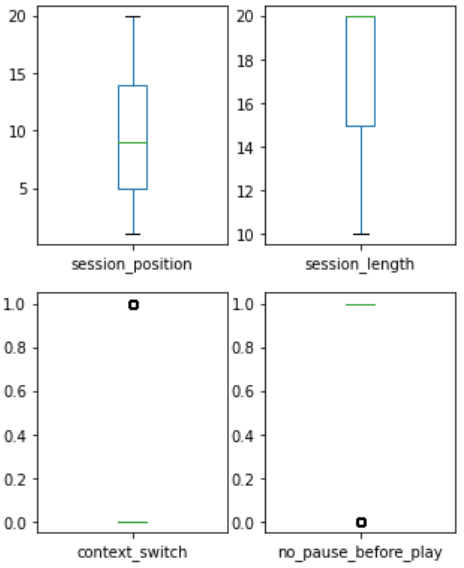


Ilustración 58: Diagrama de cajas en sesión - session\_position, session\_length, context\_switch, no\_pause\_before\_play

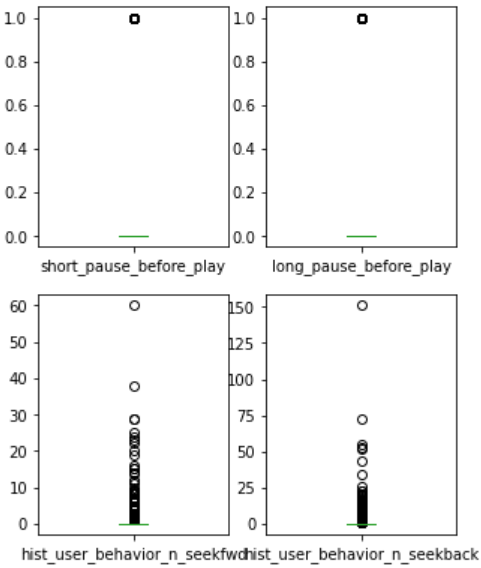


Ilustración 59: Diagrama de cajas en sesión – short\_pause\_before\_play, long\_pause\_before\_play, hist\_user\_behavior\_n\_seekfwd, hist\_user\_behavior\_n\_seekback

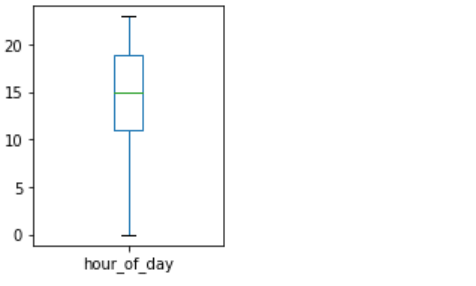


Ilustración 60: Diagrama de cajas en sesión - hour\_of\_day

Ahora ya estamos preparados para el tratamiento de outliers cuando tengamos montado todo el tablón.

En la siguiente sección vamos a poner en contexto toda la información que hemos reunido hasta el momento.

# 4. Del dato al valor

Comprobando el id de sesión, el id de la canción, las horas del día cuando se escucharon las canciones y la fecha, que sigue el formato YYYY-DD-MM, formateamos la fecha para poder operar con ella posteriormente.

Tras ese análisis, los motivos que han llevado a que comience una canción son:

* + *trackdone* es que la canción anterior llegó hasta el final de la canción y ha cambiado de canción, esto no implica que haya llegado hasta el final del fichero de la canción, pero sí que ha llegado hasta el aviso de final de canción.
  + *fwdbtn* es *forward button*, el usuario le ha dado al botón de siguiente de la lista.
  + *backbtn* es *backbutton*, el usuario le ha dado al botón de anterior de la lista.
  + *clickrow* es que el usuario ha seleccionado específicamente esa canción buscándola.
  + *appload* es que la canción fue subida por el usuario, aunque desde julio de 2019 Spotify no permite esa opción.
  + *playbtn* es que el usuario le ha dado al botón de play.
  + *remote* es que se ha lanzado en remoto desde otro dispositivo.
  + *trackerror* es que antes hubo un error.
  + *endplay* es que se ha parado temporalmente (stop), pero luego ha seguido.

Analizaremos las variables más en detalle posteriormente.

**Ahora comprobamos que una sesión puede tener varias canciones escuchadas.**

Se comprueba también que un mismo id de sesión tiene asociadas varias *track\_id\_clean*, que son los id de canción. Se v**erifica a su vez que los formatos id de las canciones coinciden con los formatos id de las sesiones.**

Todas las canciones individuales del dataset de canciones coinciden con todas las canciones individuales del dataset de sesiones.

La cantidad de todas las canciones individuales de las sesiones, guardadas en el dataset de canciones, coincide por tanto con la cantidad de todas las canciones individuales de las sesiones (es decir, no hay canciones en la fuente de canciones que no se hayan escuchado en alguna sesión).

Comprobamos también que todas las canciones que aparecen en las sesiones tienen información guardada en el dataset de canciones.

Como curiosidad, también podemos ver de cuántas canciones diferentes tenemos información. Esto significa de nuevo que disponemos exactamente de todas las canciones escuchadas en todas las sesiones.

**Observamos entre qué fechas se están estudiando las sesiones.**

Fecha mínima: ('2018-05-12 00:00:00')

Fecha máxima: ('2018-07-16 00:00:00')

En el dataset de sesiones aparecen 3 variables objeto que sí que queremos transformar en dummies para tenerlas en cuenta numéricamente.

|  |  |
| --- | --- |
| Context\_type | object |
| Hist\_user\_behavior\_reason\_start | object |
| Hist\_user\_behaviour\_reason\_end | object |

Tabla 17: Variables categóricas transformadas a dummies en tabla de sesión

La variable date no necesitamos transformarla (suponemos los sucesos invariantes en el tiempo), ni los campos id, que sirven para identificar muestras.

Al ser los sucesos invariantes en el tiempo no debe afectar desde qué instante se toman las medidas. Al muestrear la población total de forma aleatoria tampoco debe importar el orden de las muestras.

A continuación, vamos a realizar un join (intersección de tablas) entre el dataframe de sesiones y el de canciones. Ya tenemos un dataframe (el de sesión) que almacena en cada fila una sesión que escucha una canción. De este modo el dataframe de sesión guarda por fila una canción que se ha escuchado en una sesión de Spotify. Ahora además queremos que en esa fila (muestra) se tenga también información de la canción escuchada, y no solo de la sesión. Para lograrlo vamos a hacer un join entre el dataframe de sesión y el de canciones. Como ya hemos visto que el conjunto de canciones coincide exactamente con todas las canciones escuchadas en las sesiones, aplicar un join left o un join inner nos debería dar el mismo resultado.

Lo comprobamos.

Left join.

df\_sesion\_y\_track\_left = df\_training\_set.merge(right= df\_track\_features, how='left', left\_on='track\_id\_clean', right\_on='track\_id')

Inner join.

df\_sesion\_y\_track\_inner = df\_training\_set.merge(right= df\_track\_features, how='inner', left\_on='track\_id\_clean', right\_on='track\_id')

df\_sesion\_y\_track\_left.shape == df\_sesion\_y\_track\_inner.shape

True

En este punto ya estamos preparados para generar las variables dummies necesarias para que los modelos posteriores puedan procesarlas (modelos que esperan variables de entrada numérica o binaria, pero no categórica).

Nos preparamos eliminando las variables categóricas de id que no queremos que se generen como dummies porque no entrarán como variables del modelo

Hay que entender que en el conjunto {'not\_skipped', 'skip\_1', 'skip\_2', 'skip\_3'} solo uno puede ser el target y es excluyente con los otros tres atributos, puesto que las variables son excluyentes entre sí (los estados de cuándo acaba de reproducirse la canción). Por esto, no se podrán usar skip\_1, skip\_2 y skip\_3 como variables de entrada del algoritmo para predecir not\_skipped (esos valores no están disponibles en la entrada porque son todos estados de salida)

Igualmente, eliminamos 'hist\_user\_behavior\_reason\_end' porque ese dato solo lo sabríamos al acabar la reproducción de la canción, cuando ya sabríamos si la canción se ha reproducido entera o no. El problema que queremos resolver es predecir, cuando una canción empieza (con sus valores propios de la canción y los valores propios de la sesión del usuario), si el usuario escuchará la canción entera.

Vemos que queremos eliminar los registros tipo id.

No es necesario hacer una copia del dataframe df\_sesion\_y\_track\_left porque el drop no lo modifica.

Vemos las razones por las que puede empezar una canción.

{'appload',

'backbtn',

'clickrow',

'endplay',

'fwdbtn',

'playbtn',

'remote',

'trackdone',

'trackerror'}

**Eliminamos la primera dummy de cada conjunto para evitar la trampa de la variable dummy. Si no, fallaría por colinealidad.**

Eliminamos la fecha porque presuponemos que los atributos son invariantes en el tiempo y además fechas pasadas no van a ser una palanca para nuestro modelo con futuras predicciones. Si se necesitase la fecha, se podría consultar con los datos de dataframes anteriores.

Hasta ahora todas las transformaciones que hemos aplicado son externas al valor de los propios datos (podemos aplicarlas en otro dataset que contenga los mismos atributos sin conocer el valor de los atributos). En este paso separamos los datos de entrenamiento de los datos de test porque los drops y dummies se pueden aplicar a un nuevo conjunto de test para ver las métricas de eficiencia en datos que no hayamos usado para entrenar ni validar.

Recordamos que el target es 'not\_skipped'. Elegimos un 33% para test y la semilla de 42 (lo importante no es qué semilla fijamos, sino que siempre esté fijada al mismo valor para que al repetir iteraciones siempre obtengamos los mismos resultados).

Eliminamos los outliers del dataframe de canciones quedándonos con el rango intercuartílico (las muestras caen entre el Q1 y el Q3) (Sharma, 2018).

Nos quedamos con el target asociado a X train sin outliers y comprobamos que hemos tomado bien el target asociado a X train sin outliers.

Hemos eliminado los outliers. Dada la gran cantidad de datos que tenemos, podemos aplicar la medida de quedarnos con el rango intercuartílico, puesto que seguimos teniendo suficientes muestras para realizar el estudio.

Ahora que hemos quitado los outliers y que tenemos las muestras con datos con un valor útil con las que queremos trabajar, vamos a inspeccionar el balanceo de las clases para comprobar lo representativo que es el target.

Además, ya podemos indicar que cuando en los modelos apliquemos métricas de scoring, la exactitud es la métrica que en las librerías de Python se aplica por defecto, pero dicha métrica solo funciona bien si las clases están balanceadas (que sea equiprobable obtener un target positivo o negativo aleatoriamente). Veremos que vamos a analizar varias métricas e incluso balanceando las clases, elegiremos otra métrica a la que no le afecta en su rendimiento el balanceo para poder generalizar con nuevos datos de entrada que no estén balanceados.

Por todos estos motivos, en la siguiente sección nos centraremos en el balanceo de clases.

Como las clases no están balanceadas y tenemos muchas muestras, vamos a quedarnos con un subconjunto de target = 0 (mayoritario) de la misma longitud que el target positivo para introducir un sesgo menor y además poder trabajar con la exactitud, que es una métrica que funciona bien cuando las clases están balanceadas.

Basándonos en el artículo de TowardsDataScience (Badr, 2019), vamos a crear una función que ejecute internamente el código propuesto, para poder generalizar en otro momento.

def balancea(df,target,L):

shuffled\_df = df.sample (frac=1, random\_state=4) # frac=1 es para cogerlo todo

target\_1\_df = shuffled\_df.loc[shuffled\_df[target]==1] # Todo el target=1

target\_0\_df = shuffled\_df.loc[shuffled\_df[target]==0].sample(n=L, random\_state=42)

return pd.concat([target\_1\_df, target\_0\_df])

Ecuación 1: Función balancea

Ejecutamos la función que hemos creado.

Vemos las columnas del dataframe final balanceado.

Ya tenemos un tablón balanceado. Ahora nos podemos preguntar cómo han quedado las correlaciones entre las variables, al eliminar outliers, balancear las clases y disponer de un único tablón con todas las variables en lugar de las dos tablas de entrada (de canciones y de sesión).

En la siguiente sección vamos a estudiar las correlaciones actuales.

Dibujamos la matriz de correlación basándonos en el artículo de Medium incluido en las referencias (Anita, 2019).

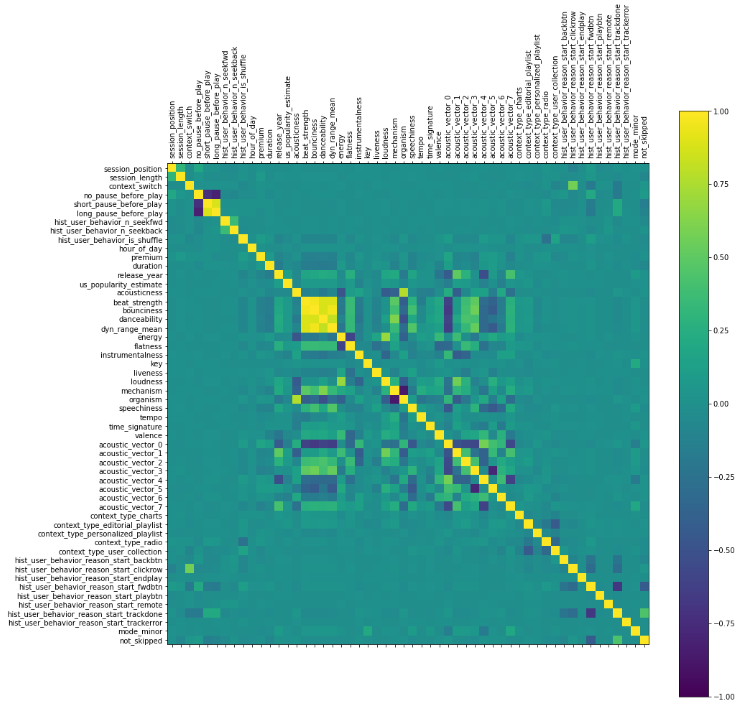


Ilustración 61: Correlaciones mapa de calor con tablón balanceado

A simple vista ya vemos que 'hist\_user\_behaviour\_reason\_start\_trackdone' es la variable con mayor correlación con el target 'not\_skipped', viendo el cuadrado verde claro que cruza en la matriz de correlaciones con el target 'not\_skipped'.

Tiene sentido que si el usuario ha escuchado la canción anterior entera sea probable que escuche esta nueva canción entera. Vemos también que los parámetros de canción más "dinámicos" presentan bastante correlación: beat\_strength, bounciness, danceability, dyn\_range\_mean.

Veamos ahora qué sucede al eliminar los outliers.

Dibujamos la matriz de correlación.

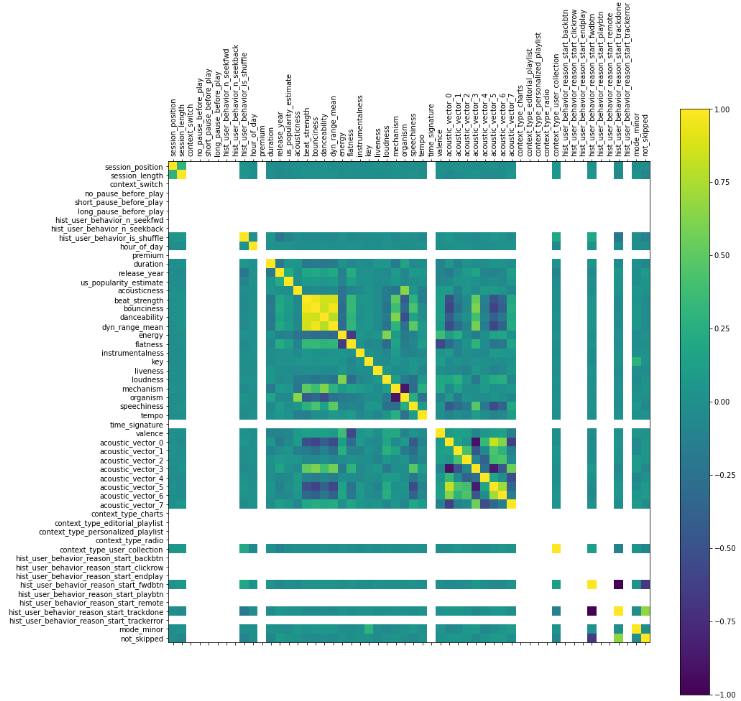


Ilustración 62: Correlaciones mapa de calor con tablón balanceado - marcando outliers

Vemos el efecto de haber eliminado outliers. Los atributos que tras eliminar sus outliers no presentan variación aparecen vacíos en la matriz de correlación.

Encontramos los atributos sin outliers que no varían. El y train sin outliers asociado al X train sin outliers que tenga datos útiles, sigue siendo el mismo y porque solo hemos eliminado columnas de la X.

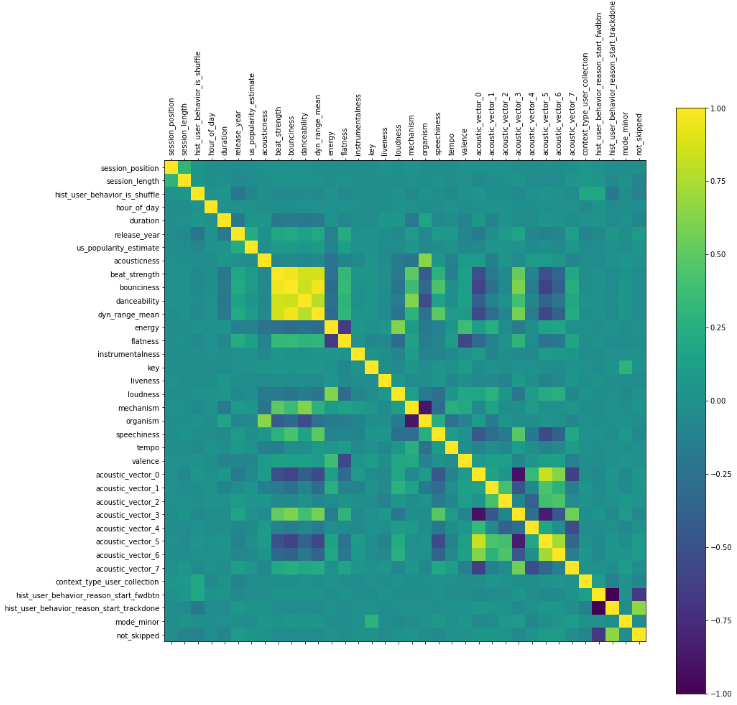


Ilustración 63: Correlaciones mapa de calor con tablón balanceado - sin outliers

Hemos descubierto más información:

Mirando la matriz de correlación en la última columna que correla respecto a 'not\_skipped', si el usuario acabó de escuchar la canción anterior entera ('hist\_user\_behaviour\_reason\_start\_trackdone') lo más probable es que no salte la canción actual. Por el contrario, si ha llegado a la canción actual después de dar al botón de 'siguiente', es muy probable que el usuario también salte la canción actual.

Escalamos para calcular la selección de variables con K-Best basado en chi cuadrado. Así todos los valores son positivos y podemos emplear el K-Best. De este modo reducimos dimensiones para representar la matriz de dispersión y además ya nos quedan los atributos preparados para ser independientes de su escala.

Cuando queramos representar gráficos donde no podemos tener en cuenta todas las variables, necesitamos seleccionar un conjunto menor de variables que contenga las variables más importantes. El método que vamos a usar es la selección basada en la prueba de chi cuadrado. Lo estudiamos en la siguiente sección.

Seleccionamos las 3 variables más representativas para el target (Gajawad, 2019) para poder representar la matriz de dispersión.

Mostramos las variables con get\_support del KBest, el dataframe escalado y reducido.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **hist\_user\_behavior\_is\_shuffle** | **hist\_user\_behavior\_reason\_start\_fwdbtn** | **hist\_user\_behavior\_reason\_start\_trackdone** |
| 14619 | 0.0 | 1.0 | 0.0 |
| 156991 | 0.0 | 1.0 | 0.0 |
| 9598 | 0.0 | 0.0 | 1.0 |

Tabla 24: X train escalado y reducido

Es una referencia del dataframe a las columnas que nos interesan.

En la siguiente sección vamos a ver la matriz de dispersión.

Como estamos manejando muchas variables, nos apoyamos en la selección del K-Best para disponer de un conjunto más reducido de variables y conseguir unas dimensiones que sí se puedan representar bien.

Preparamos el formato del dataframe para la función de seaborn.

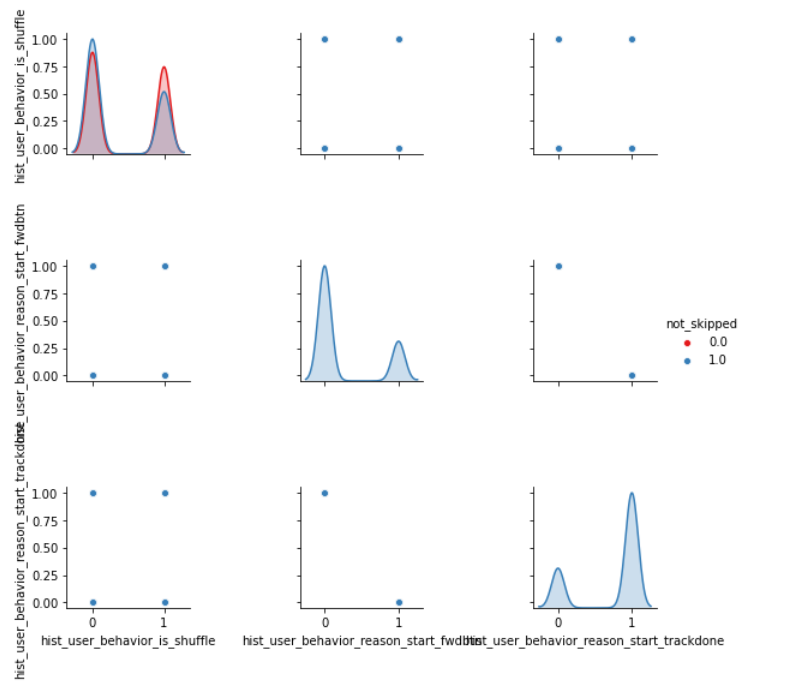


Tabla 25: Matriz de dispersión

Al ser las variables más importantes binarias, la matriz de dispersión no ofrece tanta claridad como si las variables fueran continuas.

Tras toda esta exploración ya estamos preparados para entrar en el estudio de los modelos.

# 5. Modelos de Machine Learning

Recordamos que partimos de X\_train, X\_test, y\_train, y\_test y con transformaciones obtenidas de X train hemos llegado a:

**X\_train\_escalado: (12487, 35)**

**X\_test\_escalado: (55401, 35)**

**y\_train: (12487,)**

**y\_test: (55401,)**

Cambiamos la nomenclatura por claridad (para que al final se obtengan X, y, X\_test, y\_test).

**X = X\_train\_escalado;**

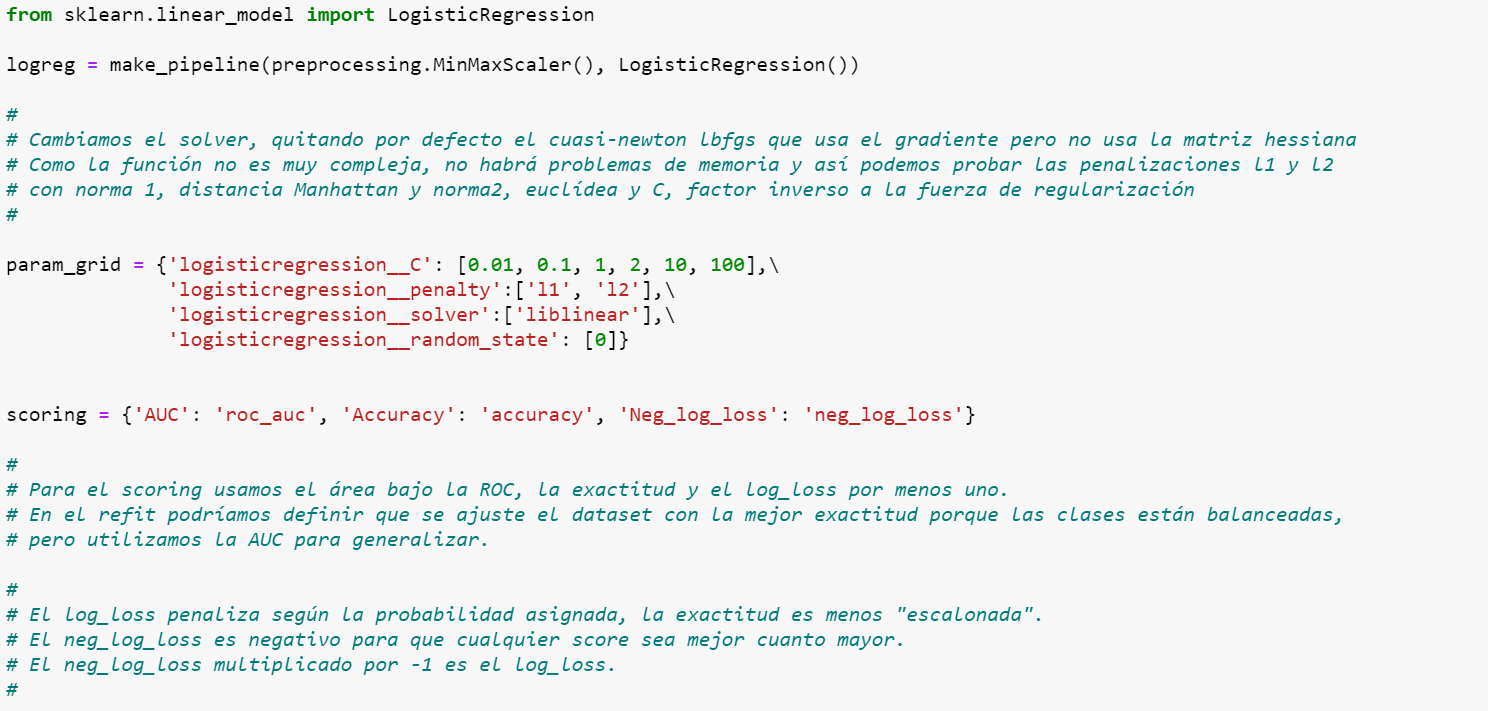
**y = y\_train\_sin\_outliers;**

**X\_test = X\_test\_escalado;**

Ya tenemos X, y, X\_test, y\_test. Esto significa que ya podemos empezar a probar los modelos.

## 5.1. Regresión Logística

El primer modelo que vamos a probar es la Regresión Logística. Es de tipo supervisado (conocemos el target), clasificador y lineal.





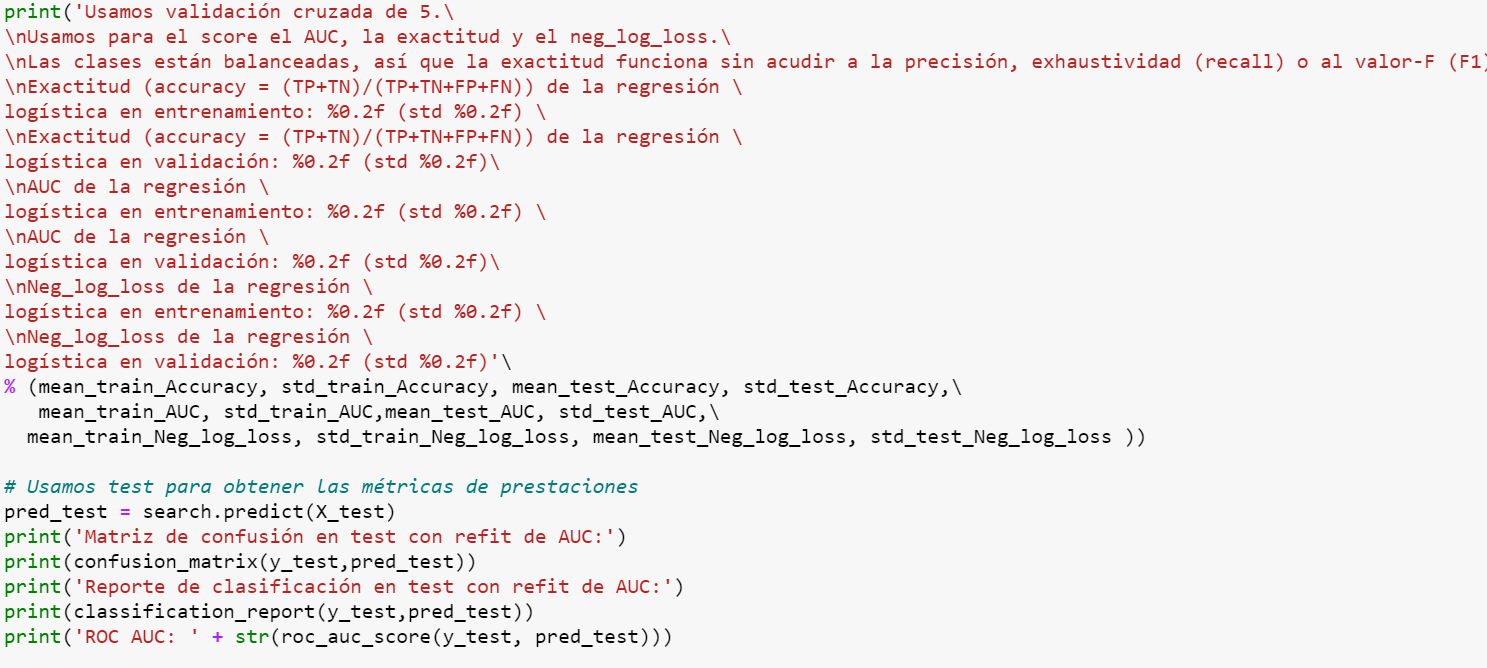




Ilustración 64: Regresión Logística

Parámetros óptimos de la regresión logística con entrenamiento y validación:

C: 2

penalty: l2

solver: liblinear

Usamos validación cruzada de 5.

Usamos para el score el AUC, la exactitud y el neg\_log\_loss.

Las clases están balanceadas, así que la exactitud funciona sin acudir a la precisión, exhaustividad (recall) o al valor-F (F1).

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) de la regresión logística en entrenamiento: 0.85 (std 0.00)

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) de la regresión logística en validación: 0.85 (std 0.00)

AUC de la regresión logística en entrenamiento: 0.86 (std 0.00)

AUC de la regresión logística en validación: 0.85 (std 0.00)

Neg\_log\_loss de la regresión logística en entrenamiento: -0.42 (std 0.00)

Neg\_log\_loss de la regresión logística en validación: -0.42 (std 0.01)

Matriz de confusión en test con refit de AUC:

[[23365 13587]

[ 5116 13333]]

Reporte de clasificación en test con refit de AUC:

precision recall f1-score support

False 0.82 0.63 0.71 36952

True 0.50 0.72 0.59 18449

accuracy 0.66 55401

macro avg 0.66 0.68 0.65 55401

weighted avg 0.71 0.66 0.67 55401

ROC AUC: 0.6775008866886639

Ilustración 65: Regresión Logística - métricas

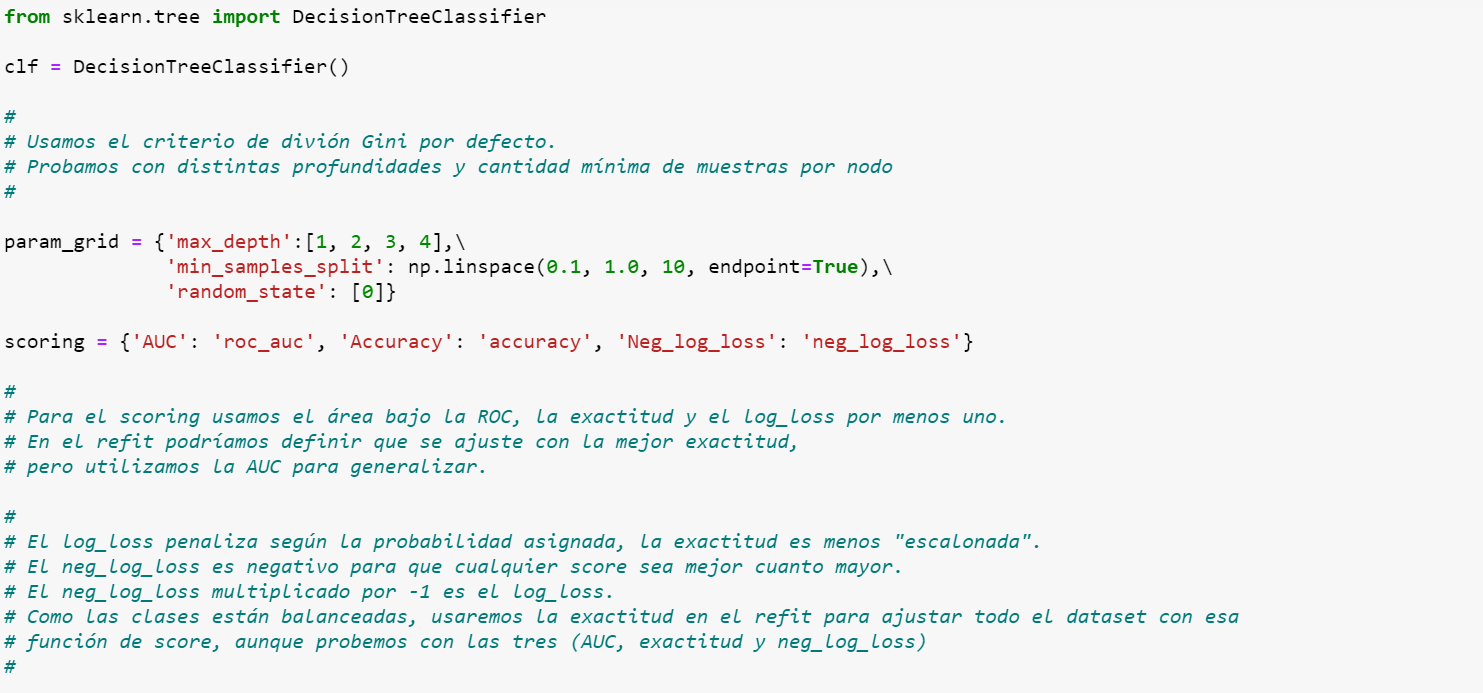
En la Ilustración 62: Regresión Logística y en la Ilustración 63: Regresión Logística – métricas podemos ver todo el detalle del modelo.

Se explica claramente la elección de la validación cruzada estratificada, del ajuste de hiperparámetros, las métricas y por generalización la ROC AUC (Narkhede, 2018).

## 

## 5.2. Árbol de Decisión

El siguiente modelo que vamos a estudiar es un Árbol de Decisión. Es de tipo supervisado (conocemos el target), clasificador y no es lineal porque no se basa en distancias euclídeas.





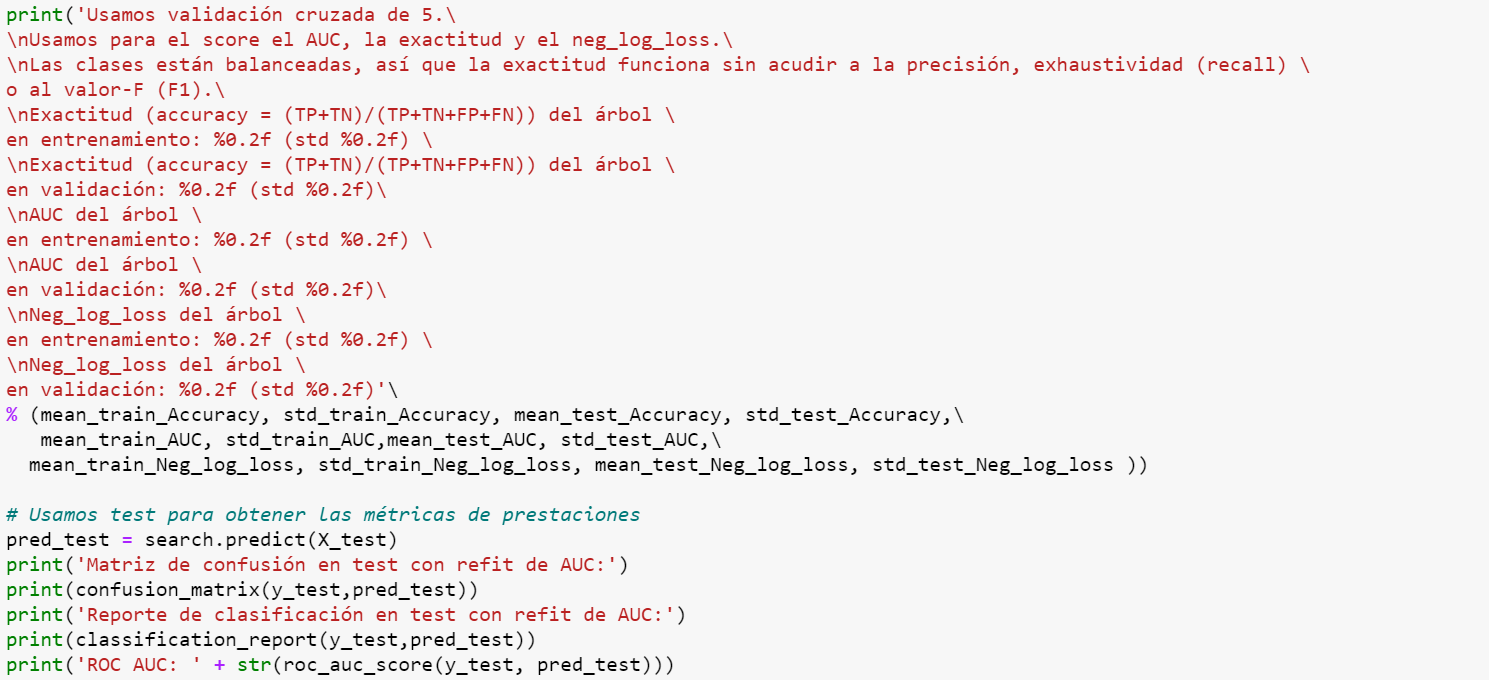




Ilustración 66: Árbol de decisión

Parámetros óptimos del árbol con entrenamiento y validación:

max\_depth: 4

min\_samples\_split: 0.10

Usamos validación cruzada de 5.

Usamos para el score el AUC, la exactitud y el neg\_log\_loss.

Las clases están balanceadas, así que la exactitud funciona sin acudir a la precisión, exhaustividad (recall) o al valor-F (F1).

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del árbol en entrenamiento: 0.85 (std 0.00)

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del árbol en validación: 0.85 (std 0.00)

AUC del árbol en entrenamiento: 0.86 (std 0.00)

AUC del árbol en validación: 0.85 (std 0.00)

Neg\_log\_loss del árbol en entrenamiento: -0.41 (std 0.00)

Neg\_log\_loss del árbol en validación: -0.43 (std 0.01)

Matriz de confusión en test con refit de AUC:

[[22641 14311]

[ 3738 14711]]

Reporte de clasificación en test con refit de AUC:

precision recall f1-score support

False 0.86 0.61 0.72 36952

True 0.51 0.80 0.62 18449

accuracy 0.67 55401

macro avg 0.68 0.71 0.67 55401

weighted avg 0.74 0.67 0.68 55401

ROC AUC: 0.7050505915672035

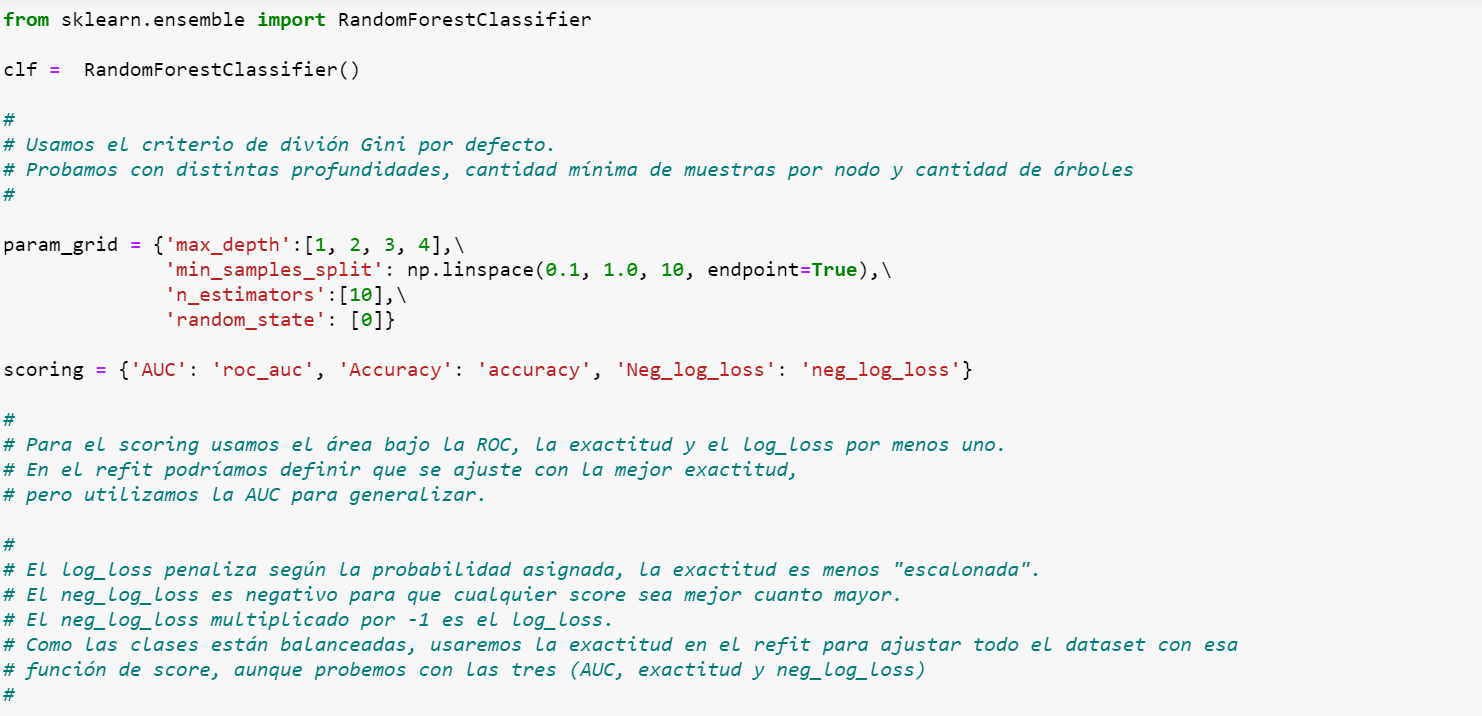
Ilustración 67: Árbol de Decisión – métricas

En la Ilustración 66: Árbol de decisión y en la Ilustración 67: Árbol de decisión – métricas podemos ver todo el detalle del modelo.

Se explica claramente la elección de la validación cruzada estratificada, del ajuste de hiperparámetros, las métricas y por generalización la ROC AUC.

## 5.3. Random Forest

El siguiente modelo que vamos a estudiar es un Random Forest. Es de tipo supervisado (conocemos el target), clasificador y no es lineal (porque es una bolsa de árboles de los que se elige el mejor).





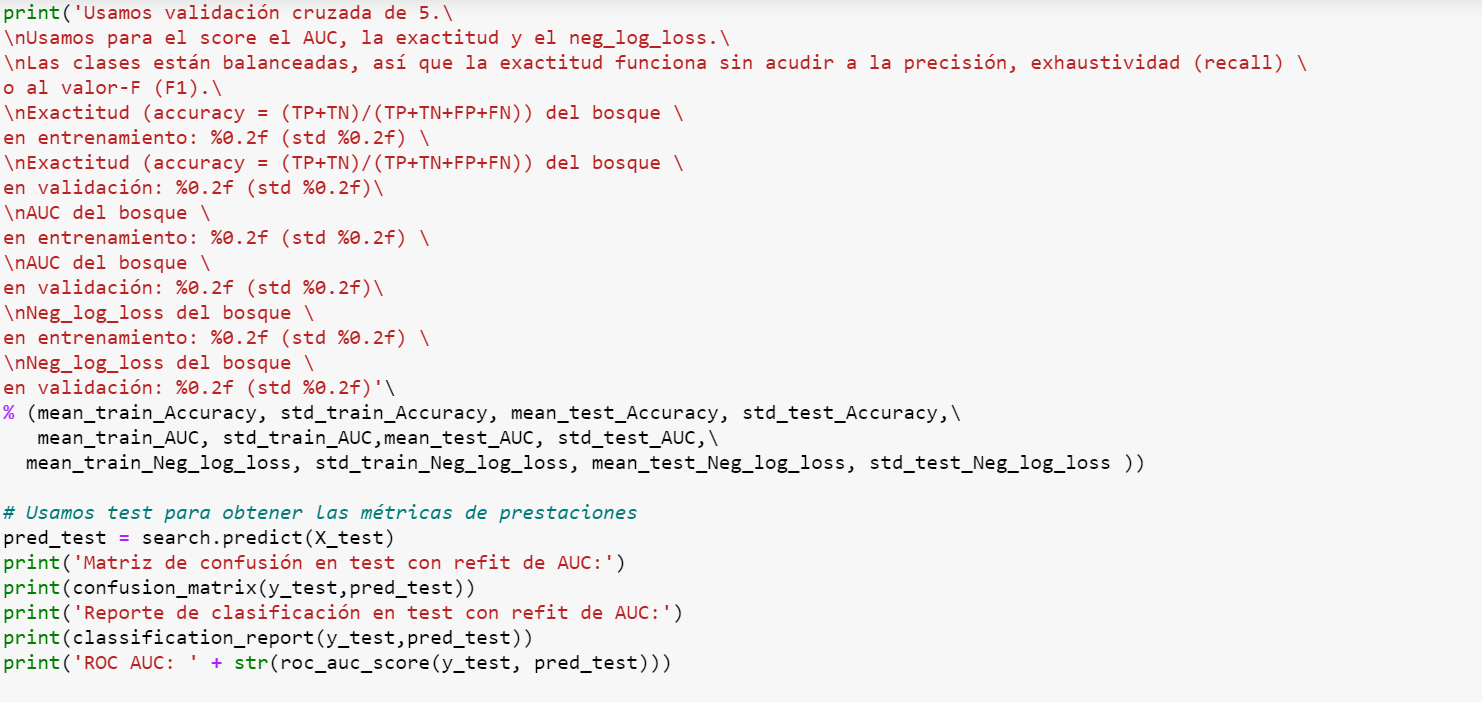




Ilustración 68: Random Forest

Parámetros óptimos del árbol con entrenamiento y validación:

max\_depth: 3

min\_samples\_split: 0.10

n\_estimators: 10

Usamos validación cruzada de 5.

Usamos para el score el AUC, la exactitud y el neg\_log\_loss.

Las clases están balanceadas, así que la exactitud funciona sin acudir a la precisión, exhaustividad (recall) o al valor-F (F1).

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del bosque en entrenamiento: 0.83 (std 0.01)

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del bosque en validación: 0.83 (std 0.01)

AUC del bosque en entrenamiento: 0.85 (std 0.00)

AUC del bosque en validación: 0.84 (std 0.00)

Neg\_log\_loss del bosque en entrenamiento: -0.50 (std 0.01)

Neg\_log\_loss del bosque en validación: -0.50 (std 0.01)

Matriz de confusión en test con refit de AUC:

[[30263 6689]

[ 7631 10818]]

Reporte de clasificación en test con refit de AUC:

precision recall f1-score support

False 0.80 0.82 0.81 36952

True 0.62 0.59 0.60 18449

accuracy 0.74 55401

macro avg 0.71 0.70 0.71 55401

weighted avg 0.74 0.74 0.74 55401

ROC AUC: 0.7026773132068462

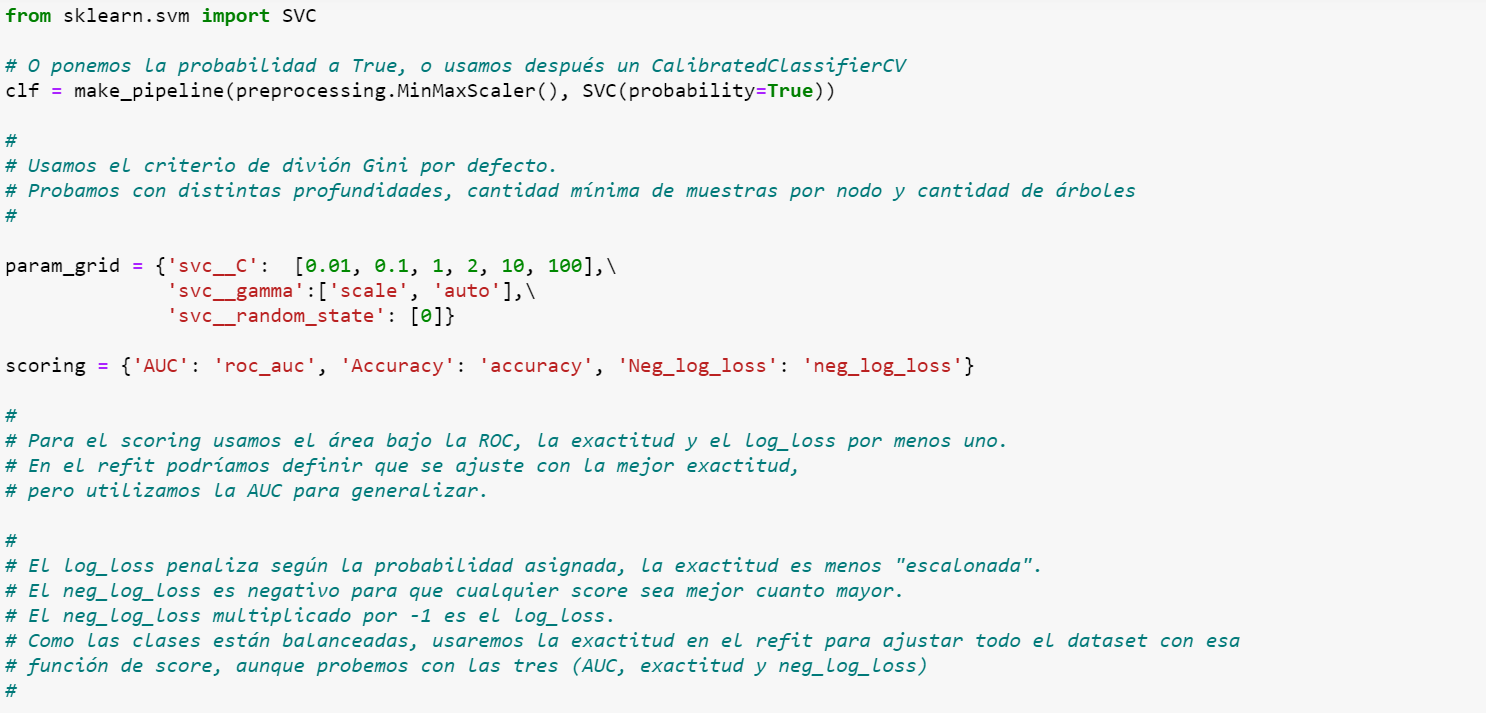
Ilustración 69: Random Forest – métricas

En la Ilustración 68: Random Forest y en la Ilustración 69: Random Forest – métricas podemos ver todo el detalle del modelo.

Se explica claramente la elección de la validación cruzada estratificada, del ajuste de hiperparámetros, las métricas y por generalización la ROC AUC.

## 5.4. Máquina de Vector Soporte (SVM)

El siguiente modelo que vamos a estudiar es una Máquina de Vector Soporte (Supported Vector Machine). Es de tipo supervisado (conocemos el target), clasificador y no es lineal (tiene fronteras Gaussianas cuando es de kernel Gaussiano). El SVM, cuando se fuerza a que aplique una función lineal, se convierte en un regresor lineal.





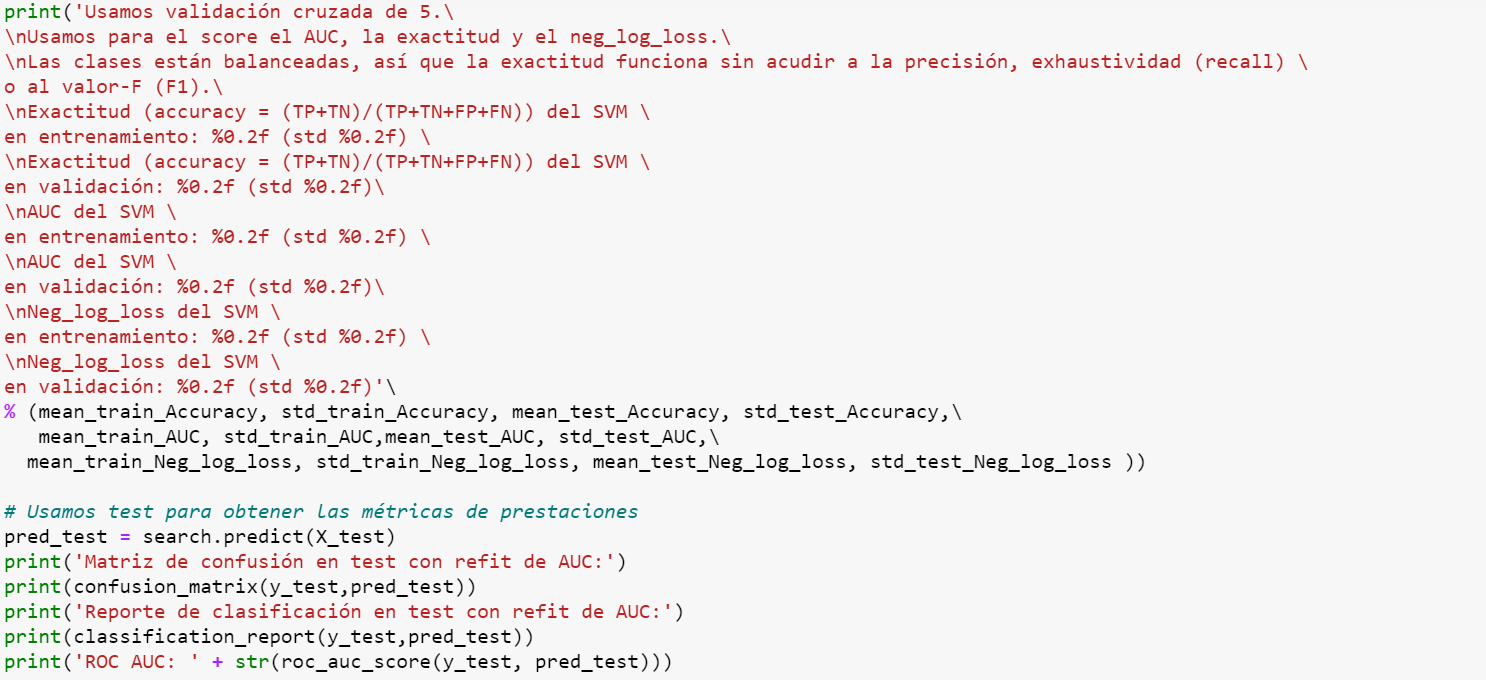




Ilustración 70: Máquina de Vector Soporte

Parámetros óptimos de SVM con entrenamiento y validación:

C: 10

gamma: scale

Usamos validación cruzada de 5.

Usamos para el score el AUC, la exactitud y el neg\_log\_loss.

Las clases están balanceadas, así que la exactitud funciona sin acudir a la precisión, exhaustividad (recall) o al valor-F (F1).

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del SVM en entrenamiento: 0.86 (std 0.00)

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del SVM en validación: 0.85 (std 0.00)

AUC del SVM en entrenamiento: 0.95 (std 0.00)

AUC del SVM en validación: 0.84 (std 0.00)

Neg\_log\_loss del SVM en entrenamiento: -0.35 (std 0.00)

Neg\_log\_loss del SVM en validación: -0.43 (std 0.00)

Matriz de confusión en test con refit de AUC:

[[28365 8587]

[ 8671 9778]]

Reporte de clasificación en test con refit de AUC:

precision recall f1-score support

False 0.77 0.77 0.77 36952

True 0.53 0.53 0.53 18449

accuracy 0.69 55401

macro avg 0.65 0.65 0.65 55401

weighted avg 0.69 0.69 0.69 55401

ROC AUC: 0.6488095378844126

Ilustración 71: Máquina de Vector Soporte – métricas

En la Ilustración 70: Máquina de Vector Soporte y en la Ilustración 71: Máquina de Vector Soporte – métricas podemos ver todo el detalle del modelo.

Se explica claramente la elección de la validación cruzada estratificada, del ajuste de hiperparámetros, las métricas y por generalización la ROC AUC.

## 5.5. Análisis de Discriminante Gaussiano

El siguiente modelo que vamos a estudiar es un Análisis de Discriminante Gaussiano. Es de tipo supervisado (conocemos el target), clasificador y no es lineal, porque trabaja con fronteras Gaussianas.

Seleccionamos las 2 variables más representativas para el target.

Mostramos las variables con get\_support del KBest.

'hist\_user\_behavior\_reason\_start\_fwdbtn' está muy correlada con respecto a 'hist\_user\_behavior\_reason\_start\_trackdone', que es el atributo más importante. Estos casos son prácticamente opuestos al eliminar el resto de 'hist\_user'.

Como el discriminante Gaussiano se basa en un discriminante, si hay dos atributos muy correlados, el discriminante para la inversa de la matriz es próximo a cero y se calcula con dificultad. Por ese motivo eliminamos en este modelo 'hist\_user\_behavior\_reason\_start\_fwdbtn'.

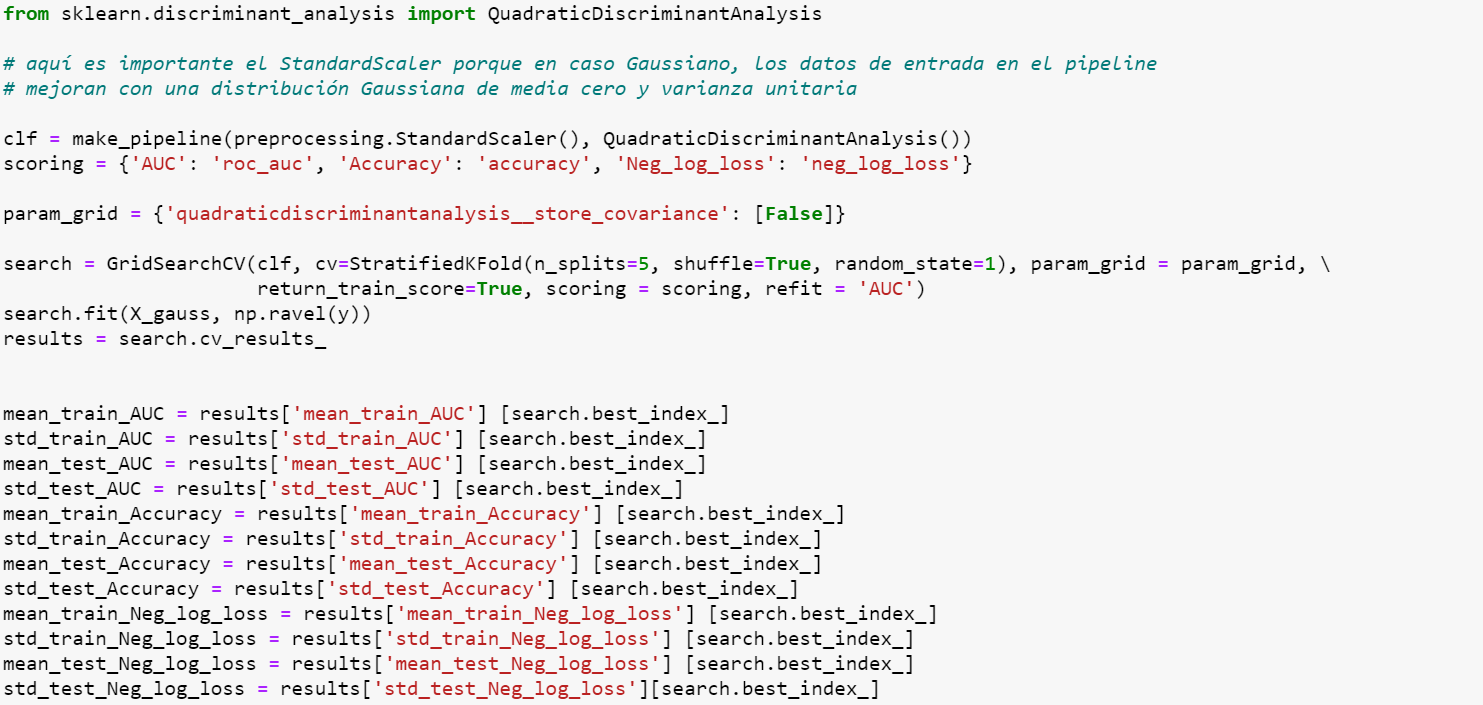






Ilustración 72: Análisis de Discriminante Gaussiano

Usamos validación cruzada de 5.

Usamos para el score el AUC, la exactitud y el neg\_log\_loss.

Las clases están balanceadas, así que la exactitud funciona sin acudir a la precisión, exhaustividad (recall) o al valor-F (F1).

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del discriminante Gaussiano en entrenamiento: 0.84 (std 0.00)

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del discriminante Gaussiano en validación: 0.83 (std 0.00)

AUC del discriminante Gaussiano en entrenamiento: 0.87 (std 0.00)

AUC del discriminante Gaussiano en validación: 0.85 (std 0.00)

Neg\_log\_loss del discriminante Gaussiano en entrenamiento: -0.51 (std 0.01)

Neg\_log\_loss del discriminante Gaussiano en validación: -0.55 (std 0.03)

Matriz de confusión en test con refit de AUC:

[[21876 15076]

[ 6148 12301]]

Reporte de clasificación en test con refit de AUC:

precision recall f1-score support

False 0.78 0.59 0.67 36952

True 0.45 0.67 0.54 18449

accuracy 0.62 55401

macro avg 0.61 0.63 0.61 55401

weighted avg 0.67 0.62 0.63 55401

ROC AUC: 0.6293841318238956

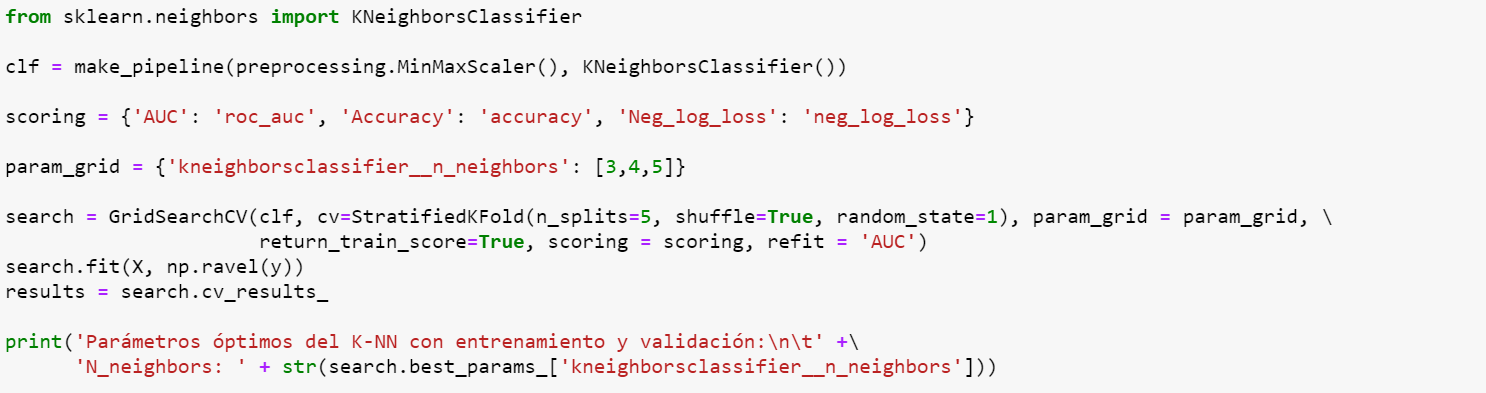
Ilustración 73: Análisis de Discriminante Gaussiano – métricas

En la Ilustración 72: Análisis de Discriminante Gaussiano y en la Ilustración 73: Análisis de Discriminante Gaussiano – métricas podemos ver todo el detalle del modelo.

Se explica claramente la elección de la validación cruzada estratificada, del ajuste de hiperparámetros, las métricas y por generalización la ROC AUC.

## 5.6. K-NN (K-Vecinos más próximos)

El siguiente modelo que vamos a estudiar es el K-Vecinos más próximos (K-Nearest Neighbors). Es de tipo supervisado (conocemos el target), clasificador y no es lineal porque la frontera depende de las distancias euclídeas, pero no se obtiene una frontera lineal al ponderar.





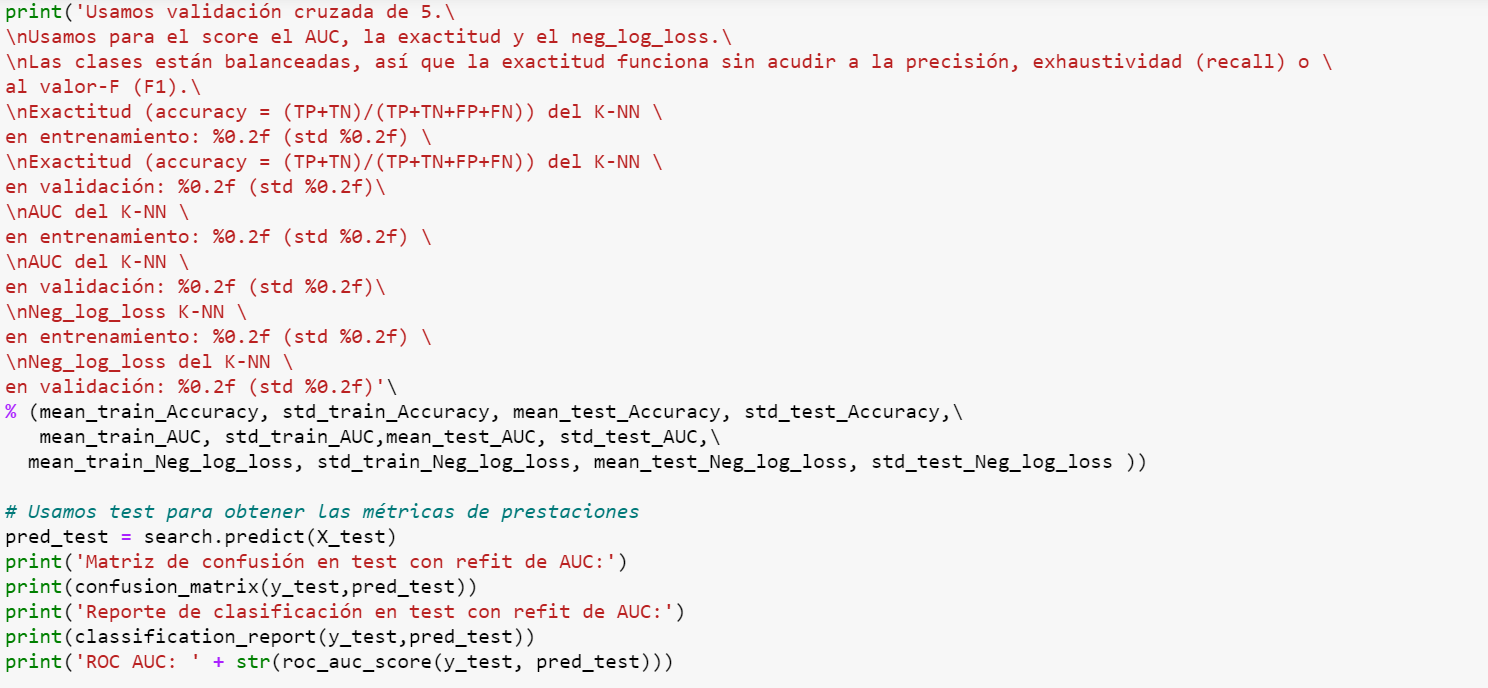




Ilustración 74: K-Vecinos más próximos

Parámetros óptimos del K-NN con entrenamiento y validación:

N\_neighbors: 5

Usamos validación cruzada de 5.

Usamos para el score el AUC, la exactitud y el neg\_log\_loss.

Las clases están balanceadas, así que la exactitud funciona sin acudir a la precisión, exhaustividad (recall) o al valor-F (F1).

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del K-NN en entrenamiento: 0.86 (std 0.00)

Exactitud (accuracy = (TP+TN)/(TP+TN+FP+FN)) del K-NN en validación: 0.83 (std 0.00)

AUC del K-NN en entrenamiento: 0.93 (std 0.00)

AUC del K-NN en validación: 0.83 (std 0.00)

Neg\_log\_loss K-NN en entrenamiento: -0.30 (std 0.00)

Neg\_log\_loss del K-NN en validación: -2.47 (std 0.08)

Matriz de confusión en test con refit de AUC:

[[24966 11986]

[ 5810 12639]]

Reporte de clasificación en test con refit de AUC:

precision recall f1-score support

False 0.81 0.68 0.74 36952

True 0.51 0.69 0.59 18449

accuracy 0.68 55401

macro avg 0.66 0.68 0.66 55401

weighted avg 0.71 0.68 0.69 55401

ROC AUC: 0.6803555179723378

Ilustración 75: K-Vecinos más próximos – métricas

En la Ilustración 74: K-Vecinos más próximos y en la Ilustración 75: K-Vecinos más próximos – métricas podemos ver todo el detalle del modelo.

Se explica claramente la elección de la validación cruzada estratificada, del ajuste de hiperparámetros, las métricas y por generalización la ROC AUC.

# 6. Selección del modelo

Veamos de manera resumida y conjunta todas las prestaciones para compararlas entre sí.

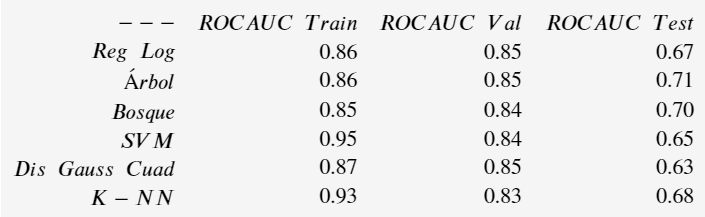


Tabla 26: Prestaciones de modelos

Observaciones del modelo:

a) El árbol de decisión es el modelo que mejor funciona en test (0.71).

b) Usar este modelo mejora la clasificación un 21 % respecto a una predicción aleatoria (es una buena mejoría y coherente con las mejoras de los modelos que emplean datos reales).

c) Observamos que los árboles son algoritmos que funcionan bien (como hemos estudiado en la teoría).

d) El bosque no mejora el árbol porque el árbol ya estaba bien optimizado (sus parámetros en el GridSearchCV).

Ya tenemos el modelo elegido y los ficheros que se han generado también guardados en local. En secciones posteriores usaremos estos ficheros para almacenarlos en una estructura HDFS y en Hive a través de contenedores Docker persistentes con volúmenes.

## 

## 6.1. Curva de aprendizaje

Tenemos una gran cantidad de datos, por lo que no ha sido un problema tener que crear datos sintéticos o comprobar si con nuestro conjunto de datos ya habíamos alcanzado el aprendizaje (sin entrar en sobreaprendizaje). Sin embargo, queremos saber cuánto ha tardado el modelo en aprender y si hemos entrado en sobreaprendizaje (caso que no parece que sea probable por los scorings de entrenamiento, validación y test).

Una buena forma de ver la rapidez en el aprendizaje y el estado en el aprendizaje del modelo es recurriendo a la curva de aprendizaje. La curva de aprendizaje muestra el score de entrenamiento y de validación en función del tamaño muestral que se va tomando (Albon, 2017).

Importamos la librería necesaria.

Creamos scores de CV de entrenamiento y test para diferentes tamaños.

Pintamos.

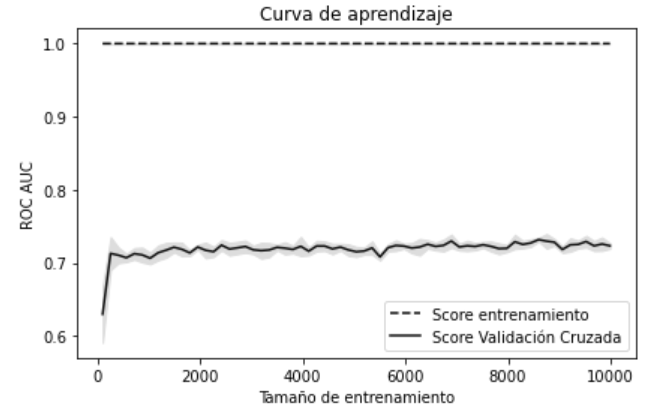


Ilustración 76: Curva de aprendizaje

Observamos que rápidamente alcanzamos una situación estable y que luego, aunque se aumente el tamaño muestral en el entrenamiento, la métrica ROC AUC no mejora significativamente.

El score de la validación cruzada no se va acercando al score de entrenamiento, lo que implica que no hemos entrado en sobreaprendizaje (las curvas se tendrían que estar acercando mucho).

También apreciamos la varianza sombreada alrededor del score.

## 

## 6.2. Serialización

Recordamos que hemos seleccionado el árbol de decisión.

GridSearchCV(cv=StratifiedKFold(n\_splits=5, random\_state=1, shuffle=True),

error\_score=nan,

estimator=DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None,

max\_features=None,

max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0,

min\_impurity\_split=None,

min\_samples\_leaf=1,

min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0,

presort='deprecated',

random\_state=None,

splitter='best'),

iid='deprecated', n\_jobs=None,

param\_grid={'max\_depth': [1, 2, 3, 4],

'min\_samples\_split': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),

'random\_state': [0]},

pre\_dispatch='2\*n\_jobs', refit='AUC', return\_train\_score=True,

scoring={'AUC': 'roc\_auc', 'Accuracy': 'accuracy',

'Neg\_log\_loss': 'neg\_log\_loss'},

verbose=0)

Guardamos el modelo en el disco, cuando lo vaya a recuperar en otro momento cargamos el modelo del disco.

Comprobamos que al serializar el modelo y al volver a cargarlo, si ejecutamos de nuevo el dataset de test, que obtenemos la ROC AUC para el árbol de decisión, que como hemos visto previamente es 0.72

Es hora de encaminarnos de nuevo a Docker para desplegar la infraestructura Big Data sobre la que almacenaremos nuestro modelo y ficheros resultantes.

# 

# 7. Arquitectura en Docker

En este capítulo vamos a analizar cómo desplegar un HDFS con una capa de Hive por encima, lanzando todos los servicios con Docker. De este modo podremos almacenar en Hive los cuatro ficheros cvs obtenidos en el modelo. Después almacenaremos el propio modelo serializado en HDFS.

Posteriormente utilizaremos un notebook Jupyter con PySpark para volver a cruzar las fuentes iniciales creando un tablón en csv. La idea es ver cómo se puede pasar procesado del Jupyter inicial a un Jupyter con PySpark.

Por último, veremos cómo ese tablón csv global se puede leer y mostrar basándonos en la pila ELK.

En secciones posteriores trataremos también los caminos abiertos para seguir evolucionando esta memoria.

## 7.1 HDFS (Hadoop Distributed File System)

Para lanzar este entorno hemos buscado en GitLab código compartido que ya tuviera el entorno montado (Thibaut, 2017). Por coherencia, hemos buscado entornos que además de HDFS y Hive, tuvieran Spark. Es más lógico que un entorno puesto a disposición de los usuarios tenga Spark antes que Hive (o adicional). Además, así podremos tener el entorno ya preparado en caso de querer extender el proyecto hacia Spark, como ha sido el caso. Sin embargo, no se utiliza en el Docker-compose de referencia la imagen de Spark porque dicha imagen viene con Spark en Scala, y buscamos tener PySpark.

Por la propia ejecución del proyecto, aplicamos primero las tecnologías de HDFS y de Hive, pues las hemos estudiado primero en el temario del curso.

El fichero Docker-compose.yml con los servicios de HDFS se encuentra en la carpeta BigDataFiles de los adjuntos.

Un fichero Docker-compose.yml es muy útil cuando trabajamos con Docker porque nos permite definir varios contenedores Docker en el mismo fichero, que se podrán levantar posteriormente a la vez y además habiendo definido la relación que hay entre los contenedores.

Veamos qué contenedores Docker se necesitan para poder tener levantado y ejecutándose un entorno HDFS. Con este objetivo analizamos el documento Docker-compose.yml:

version: '2'

Es la versión del propio documento Docker-compose.yml para que Docker entienda las etiquetas y estructura del documento.

services:

Entramos en la sección de los servicios que vamos a tener disponibles.

namenode:

image: bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8

container\_name: namenode

volumes:

- ./data/namenode:/hadoop/dfs/name

environment:

- CLUSTER\_NAME=test

env\_file:

- ./hadoop.env

ports:

- 50070:50070

Ilustración 77: HDFS - namenode

Servicio del namenode del HDFS (*Hadoop Distributed File System*, sistema de ficheros distribuido de Hadoop). El namenode se encarga de conocer dónde está la información de los datanodes.

Seleccionamos una imagen Docker del namenode, usa Java 8.

La etiqueta del container\_name es un nombre que le damos para referirnos al contenedor Docker después.

Los volúmenes son muy importantes, ahí es donde se monta el espacio para que los datos del namenode persistan cuando se apague el contenedor. Vemos que los datos del contenedor se almacenan montándose en un directorio *data/namenode* del directorio local actual.

La etiqueta de entorno describe valores que se le pasan al contenedor. En este caso es el nombre del cluster.

Además de esa variable de entorno, se pueden definir muchas más en un fichero de entorno, en este caso es *hadoop.env*. Dicho fichero de entorno describe variables de sistema de ficheros, usuarios, proxies, permisos dfs y configuración YARN.

Si se desea más detalle, el documento adjunto a esta entrega del proyecto define cada valor necesario. El punto más importante es resaltar que gracias a YARN vamos a poder trabajar después con Hive, puesto que los procesos Java del namenode y de los datanodes son propios de HDFS, pero YARN es propio de Hive.

Por último, la etiqueta de los puertos asigna el mapeo entre los puertos del propio contenedor y de nuestra máquina host (en este caso la máquina virtual Ubuntu).

datanode:

image: bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8

depends\_on:

- namenode

volumes:

- ./data/datanode:/hadoop/dfs/data

env\_file:

- ./hadoop.env

ports:

- 50075:50075

Ilustración 78: HDFS - datanode

Servicio del datanode del HDFS (*Hadoop Distributed File System*, sistema de ficheros distribuido de Hadoop). El datanode se encarga de almacenar los bloques de datos del HDFS.

Seleccionamos una imagen Docker del datanode, usa Java 8.

Evidentemente, el proceso Java del datanode depende del proceso Java del namenode, puesto que los datanodes tienen los datos que el namenode registra. Es decir, son los dos procesos necesarios para poder levantar un HDFS.

Los volúmenes son muy importantes, ahí es donde se monta el espacio para que los datos del datanode persistan cuando se apague el contenedor. Vemos que los datos del contenedor se almacenan montándose en un directorio *data/datanode* del directorio local actual.

Las variables de entorno se definen en un fichero, en este caso es *hadoop.env*. Dicho fichero de entorno describe variables de sistema de ficheros, usuarios, proxies, permisos dfs y configuración YARN.

Si se desea más detalle, el documento adjunto a esta entrega del proyecto define cada valor necesario. El punto más importante es resaltar que gracias a YARN vamos a poder trabajar después con Hive, puesto que los procesos Java del namenode y de los datanodes son propios de HDFS, pero YARN es propio de Hive.

Por último, la etiqueta de los puertos asigna el mapeo entre los puertos del propio contenedor y de nuestra máquina host (en este caso la máquina virtual Ubuntu).

hue:

image: bde2020/hdfs-filebrowser:3.11

ports:

- 8088:8088

environment:

- NAMENODE\_HOST=namenode

Ilustración 79: HUE

Servicio del Hue. Llamamos Hue al navegador de ficheros del HDFS.

Seleccionamos una imagen Docker.

Se define la variable de entorno de cuál es el namenode del HDFS.

La etiqueta de los puertos asigna el mapeo entre los puertos del propio contenedor y de nuestra máquina host (en este caso la máquina virtual Ubuntu).

## 7.2. Hive (con PostgreSQL)

Veamos ahora qué servicios están incluidos en el Docker-compose.yml de Hive:

version: '2'

De nuevo, la versión del documento Docker-compose.yml. Esto sirve para que Docker sepa qué etiquetas pueden aparecer y la estructura del propio documento.

services:

Entramos en los servicios (cada contenedor Docker contiene un servicio, así se trabaja de forma orientada a servicios y se interfiere lo mínimo posible entre servicios que no tengan dependencias directas).

Vamos a ver primero todos los servicios que se describen para darnos cuenta de que coincide con el Docker-compose del HDFS que ya hemos tratado en el punto anterior, con la diferencia de que ahora hay que definir el metastore de Hive (la base de datos que contiene la metainformación de los esquemas Hive) y añadir las variables de entorno necesarias en el fichero que contiene las variables de entorno.

Veamos como el documento cumple estas condiciones:

namenode:

image: bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8

container\_name: namenode

volumes:

- ./data/namenode:/hadoop/dfs/name

environment:

- CLUSTER\_NAME=test

env\_file:

- ./hadoop-hive.env

ports:

- 50070:50070

datanode:

image: bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8

depends\_on:

- namenode

volumes:

- ./data/datanode:/hadoop/dfs/data

env\_file:

- ./hadoop-hive.env

ports:

- 50075:50075

hive-server:

image: bde2020/hive:2.1.0-postgresql-metastore

container\_name: hive-server

env\_file:

- ./hadoop-hive.env

environment:

- "HIVE\_CORE\_CONF\_javax\_jdo\_option\_ConnectionURL=jdbc:postgresql://hive-metastore/metastore"

ports:

- "10000:10000"

hive-metastore:

image: bde2020/hive:2.1.0-postgresql-metastore

container\_name: hive-metastore

env\_file:

- ./hadoop-hive.env

command: /opt/hive/bin/hive --service metastore

hive-metastore-postgresql:

image: bde2020/hive-metastore-postgresql:2.1.0

hue:

image: bde2020/hdfs-filebrowser:3.11

ports:

- 8088:8088

environment:

- NAMENODE\_HOST=namenode

Ilustración 80: Docker-compose-Hive

Hemos comprobado como las únicas diferencias son que ahora necesitamos una base de datos para el metastore de Hive. Se elige una base de datos PostgreSQL.

El fichero con las variables de entorno se incluye en el entregable de este proyecto y se puede consultar en detalle. Lo importante es detectar la naturaleza de esas variables, siendo estas:

* + Conector JDBC al metastore PostgreSQL
  + Drivers
  + Contraseñas de acceso
  + Diseño de esquemas
  + Configuración del metastore de Hive
  + Sistema de ficheros (file system del HDFS)
  + Todas las variables de configuración del propio HDFS que está por debajo de Hive (proxies, usuarios, grupos, permisos…)
  + YARN (proceso clave en la mejora del rendimiento en Hive, por descongestionar la gestión de los recursos del cluster)

## 7.3. Lanzando HDFS y Hive

En la sección de Anexos se encuentra el detalle de cómo levantar HDFS y Hive en Docker.

Comprobamos que tenemos conectividad con los servicios lanzados.

Se muestran los resultados en las ilustraciones 82 y 83.



Ilustración 81: namenode - URL

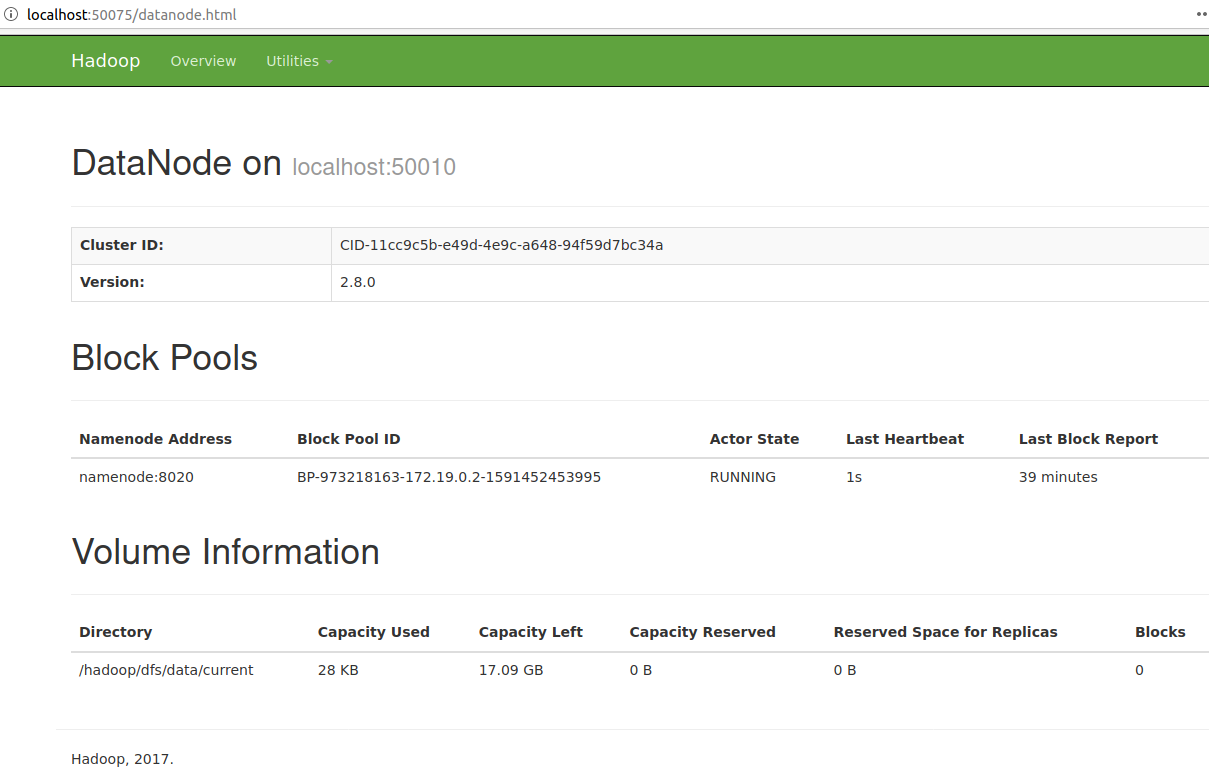


Ilustración 82: datanode - URL

Miramos todos los procesos Docker abiertos:

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$ docker ps

Stdout:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

0ad25d6bd012 **bde2020/hive:2.1.0-postgresql-metastore** "entrypoint.sh /bin/…" 46 minutes ago Up 46 minutes 0.0.0.0:10000->10000/tcp, 10002/tcp hive-server

9bfee770fbda **bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8** "/entrypoint.sh /run…" 47 minutes ago Up 47 minutes (healthy) 0.0.0.0:50075->50075/tcp docker-hadoop-spark-workbench\_datanode\_1

0525c35106f6 **bde2020/hive:2.1.0-postgresql-metastore** "entrypoint.sh /opt/…" 47 minutes ago Up 47 minutes 10000/tcp, 10002/tcp hive-metastore

79c6bb46ab74 **bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8** "/entrypoint.sh /run…" 49 minutes ago Up 49 minutes (healthy) 0.0.0.0:50070->50070/tcp namenode

a35c0edc7222 **bde2020/hive-metastore-postgresql:2.1.0** "/docker-entrypoint.…" 49 minutes ago Up 49 minutes 5432/tcp docker-hadoop-spark-workbench\_hive-metastore-postgresql\_1

6452105eeb82 **bde2020/hdfs-filebrowser:3.11** "/entrypoint.sh buil…" 3 hours ago Up 3 hours 0.0.0.0:8088->8088/tcp docker-hadoop-spark-workbench\_hue\_1

6b6abf3bed85 **jupyter/datascience-notebook:latest** "tini -g -- start-no…" 23 hours ago Up 23 hours 0.0.0.0:8888->8888/tcp jupyter

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$

Vemos que tenemos levantados todos los contenedores Docker con los servicios que estamos utilizando.

## 

## 7.4. Almacenando ficheros en Hive sobre HDFS

Entramos en el HiveServer (detalles en la sección de Anexos).

Entramos en el Docker de Hive y ejecutamos Hive para entrar en su consola.

Allí creamos el esquema de las tablas que usamos en el modelo (X\_train.csv, X\_test.csv, y\_train.csv, y\_test.csv). Después cargamos los propios ficheros csv y comprobamos que podemos hacer consultas a esas tablas en Hive.

Una vez que tenemos almacenadas las tablas en Hive, queremos almacenar también el modelo serializado. Sin embargo, el modelo serializado no tiene una estructura tabular apropiada para Hive. Esto lo resolvemos guardando el modelo directamente en el HDFS trabajando ahora con el Namenode y el Datanode.

Como se ha comentado previamente, todos los detalles técnicos se encuentran en la sección de Anexos.

## 7.5. PySpark

En la parte final queremos profundizar en Spark con Python, también llamado PySpark. El modelo ya lo hemos generado al inicio del proyecto en un notebook de Jupyter. Lo que se busca en este apartado es trabajar introduciendo Spark para ver la dirección en la que habría que ir para evolucionar la memoria hacia un procesado y modelado en Spark.

Por cuestiones de alcance del proyecto, no se va a volver a calcular el modelo dentro de Spark, pero como el objetivo de la memoria es educativo, vamos a sentar la base para tener Spark levantado en un contenedor Docker, donde haremos el cruce de las tablas de datos de sesiones y de datos de canciones de Spotify. Es una manera de partir de un tablón general ya cruzado donde posteriormente se puede aplicar el procesado.

Esta aproximación también es muy útil, porque cuando se trabaja con cantidades muy grandes de datos los cruces pueden tardar mucho tiempo, y Spark va a ser más eficiente por su tratamiento de datos en memoria que correr el cruce en un Jupyter local.

Además de por las propias prestaciones, tenemos que observar que en local también habrá cantidades de datos que no podremos manejar. Spark es distribuido (en contraposición al notebook de Jupyter inicial), lo que es otro buen motivo añadido para evolucionar este trabajo hacia Spark.

Para lanzar PySpark en Docker, lo ejecutamos primero sin el volumen mapeado, paramos el contenedor y mapeamos el volumen. Esto es así por un tema de permisos y grupos dentro de Docker con la imagen que nos vamos a traer.

docker run -p 8888:8888 jupyter/all-spark-notebook

docker run -p 8888:8888 -v ~:/home/jovyan/workspace jupyter/all-spark-notebook



Ilustración : PySpark en Jupyter

Comprobamos que podemos abrir una SparkSession.

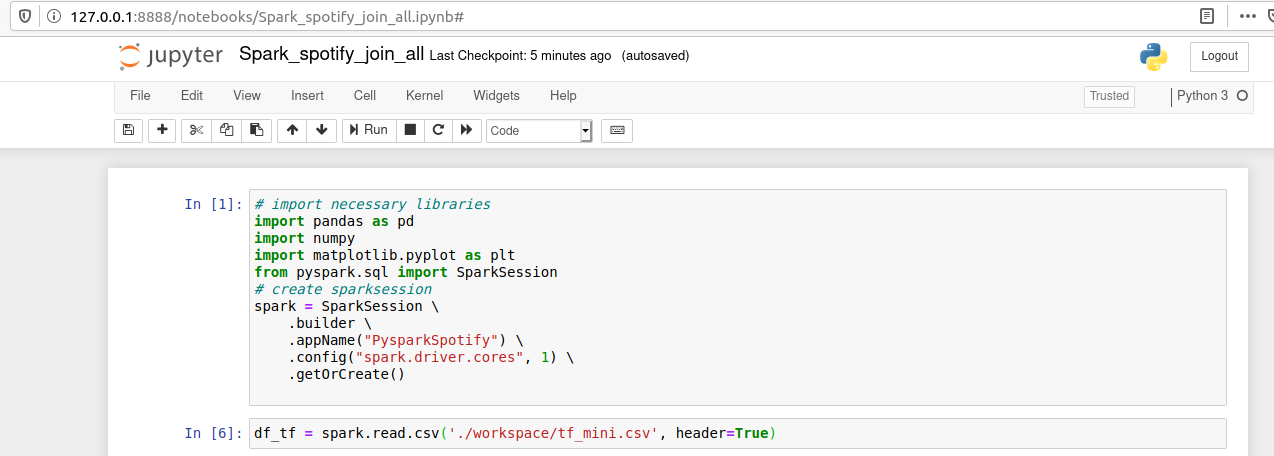


Ilustración : SparkSession

El notebook *Spark\_spotify\_join\_all* que realiza el cruce se adjunta a esta memoria.

Si se desea más detalle se puede acudir a dicho documento donde cada celda está explicada.

De todas formas, aquí también mostramos que el cruce final se ejecuta correctamente.

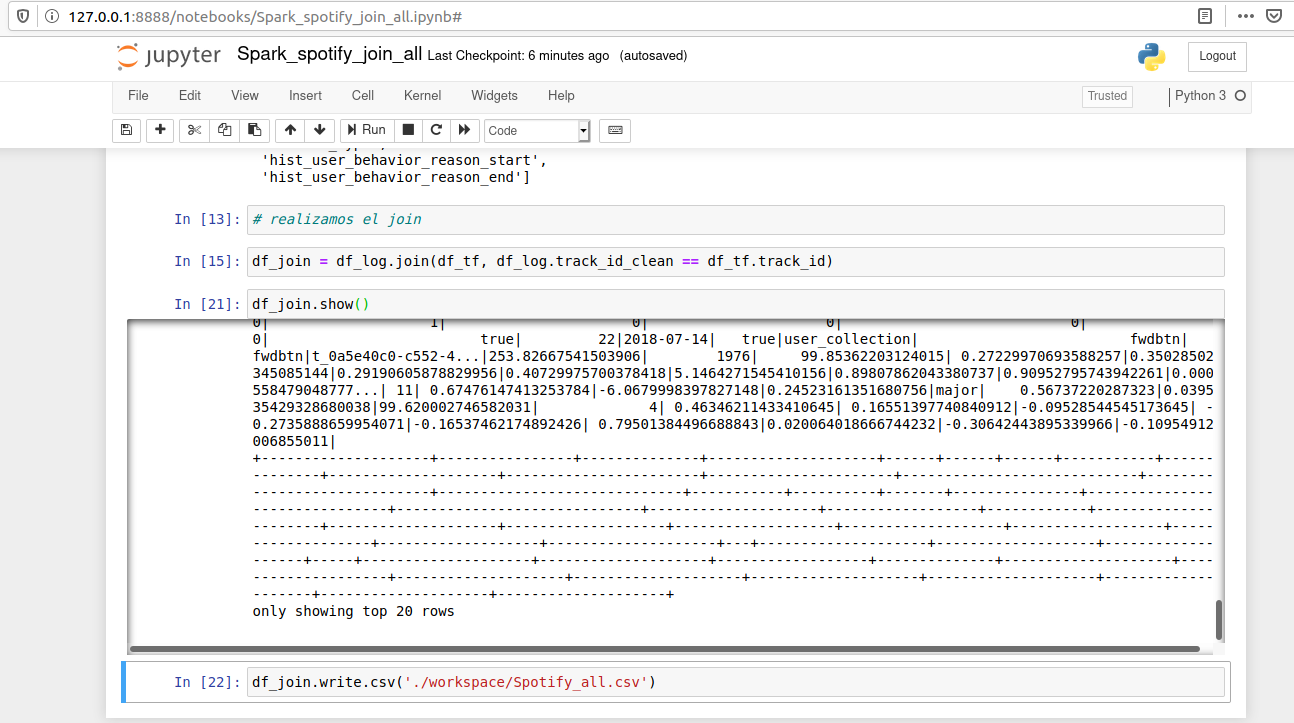


Ilustración : PySpark join

En la siguiente sección estudiaremos la pila ELK para las visualizaciones.

## 7.6. Visualizaciones con ELK

Utilizando también Docker vamos a levantar una pila ELK (Elasticsearch – Logstash – Kibana) para poder realizar un dashboard basado en el tablón csv que hemos calculado con Spark en el punto anterior.

Todos los ficheros que se necesitan están adjuntos en esta entrega.

Primero nos traemos una imagen Docker que contenga la pila ELK.

docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -p 9300:9300 -p 9600:9600 -it --name elk sebp/elk

Copiamos el csv dentro del contenedor.

docker cp ~/Spotify\_all.csv e4330fd2bb4b:/media/Spotify\_all.csv

Copiamos el conf dentro del contenedor.

docker cp ~/Spotify.conf e4330fd2bb4b:/etc/logstash/conf.d/Spotify.conf

Entramos en el contenedor.

docker exec -it e4330fd2bb4b /bin/bash

En en /etc/logstash/conf.d/ está nuestro fichero Spotify.conf

Copiamos el fichero pipelines.yml

docker cp ~/pipelines.yml e4330fd2bb4b:/opt/logstash/config/pipelines.yml

Ahora desde Kibana vamos a hacer el mapeo de los tipos en Elasticsearch, **antes de ingestar con Logstash los campos del csv**. Si no se hace en este orden, los campos serán de tipo string y tendremos problemas para cambiarlos posteriormente. El código del mapeo se puede encontrar adjunto en los documentos que acompañan a esta memoria.

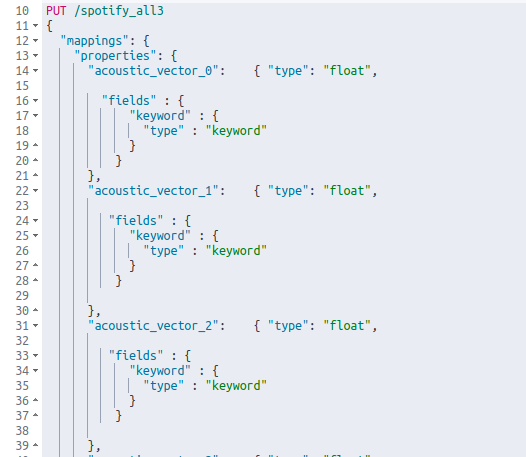


Ilustración : Elasticsearch - mapeo

Ejecutamos el fichero de configuración para cargar los datos del csv.

/opt/logstash/bin# ./logstash --path.data /tmp/logstash/data -f /etc/logstash/conf.d/Spotify.conf

En la siguiente figura vemos cómo Logstash se encarga de leer correctamente los campos del csv. Notamos que la lectura es correcta porque los colores indican que no está tomando todos los campos como si fuesen string (también hay numéricos y booleanos, por eso los colores distintos en el texto).



Ilustración : Logstash, tipos correctos

En la siguiente figura se puede ver cómo se han guardado correctamente los campos y los tipos en Elasticsearch.

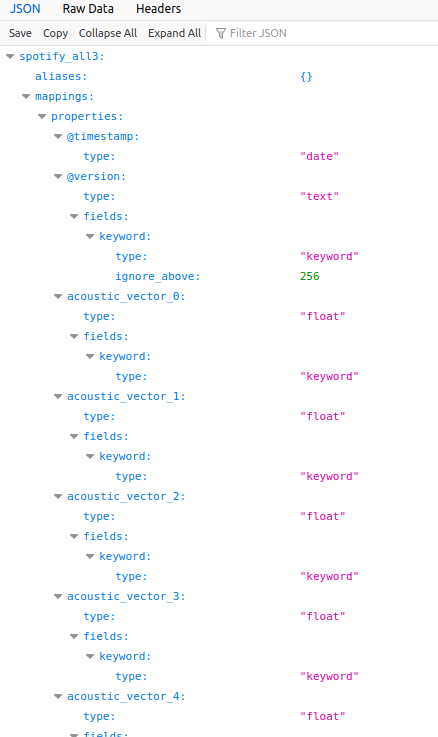


Ilustración :Elasticsearch, tipos correctos

El siguiente paso es la creación del índice desde Kibana. Este paso se realiza desde la sección Discover de Kibana.

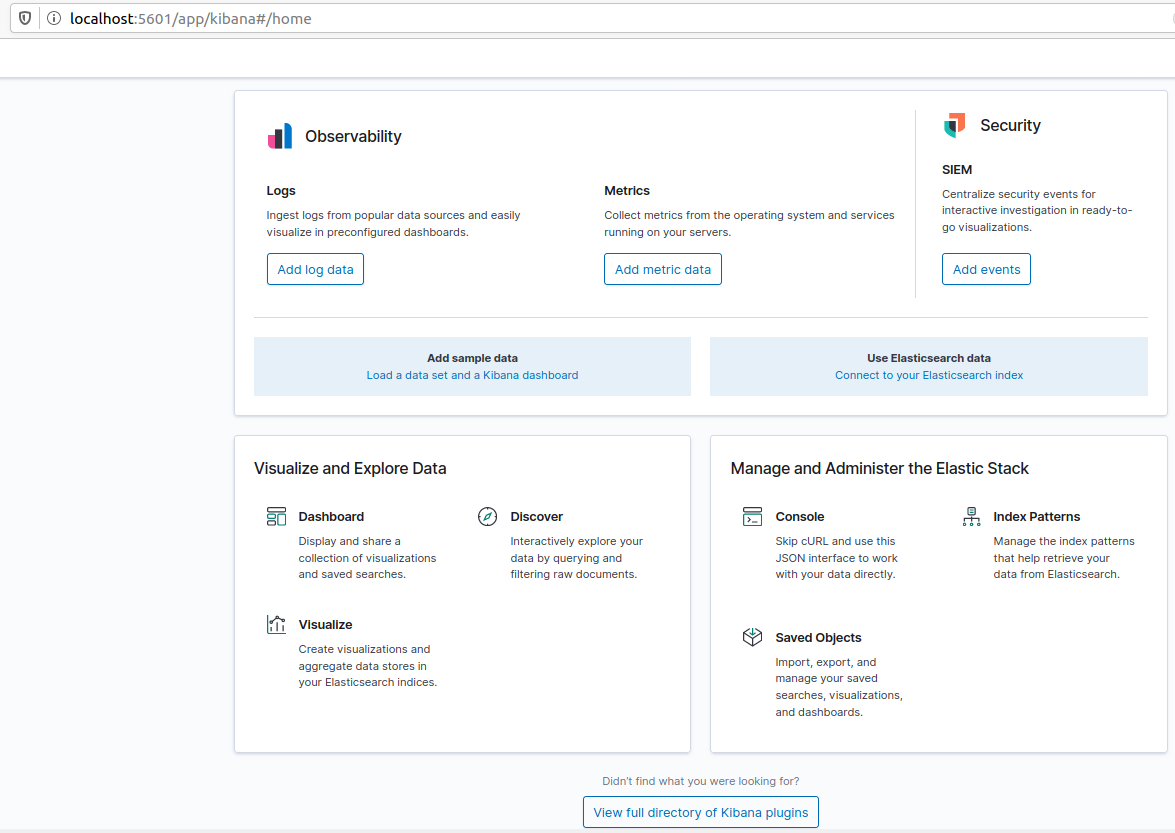


Ilustración : Kibana, discover

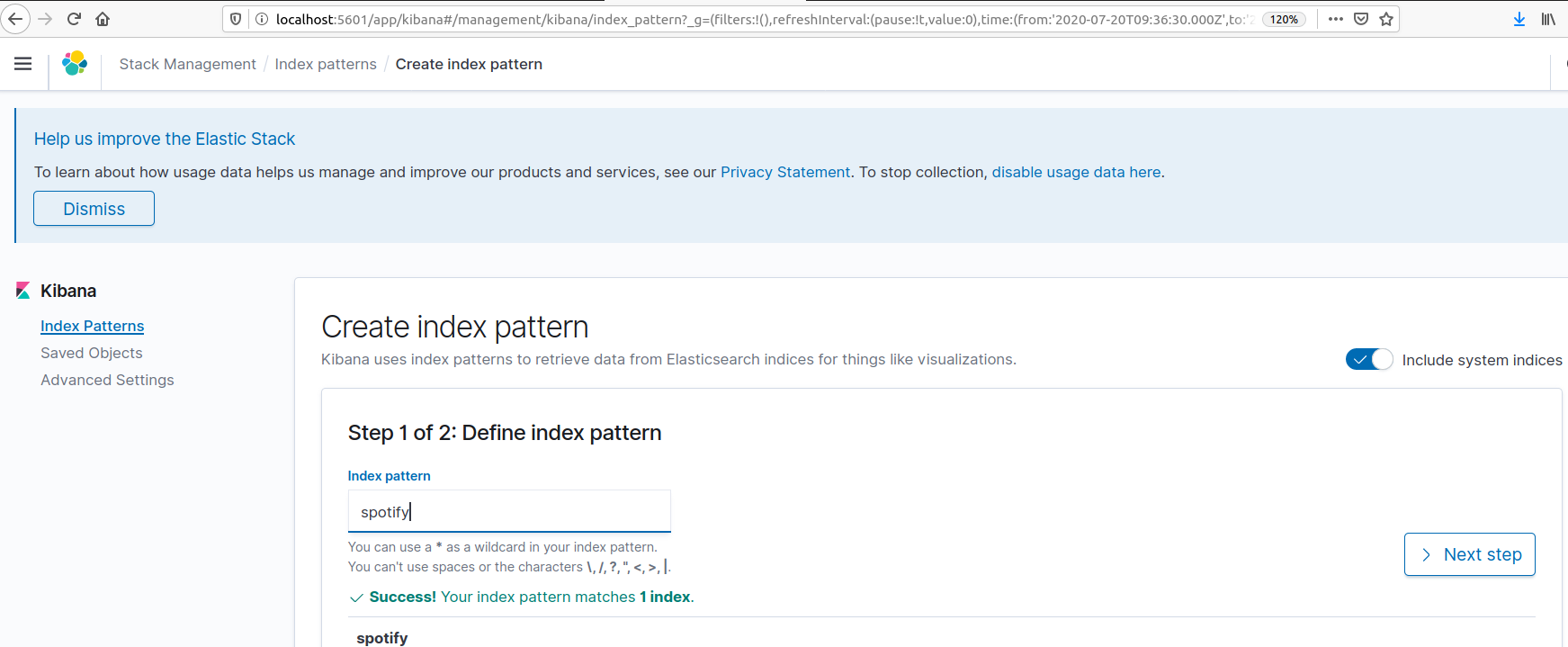


Ilustración :Kibana. índice paso 1

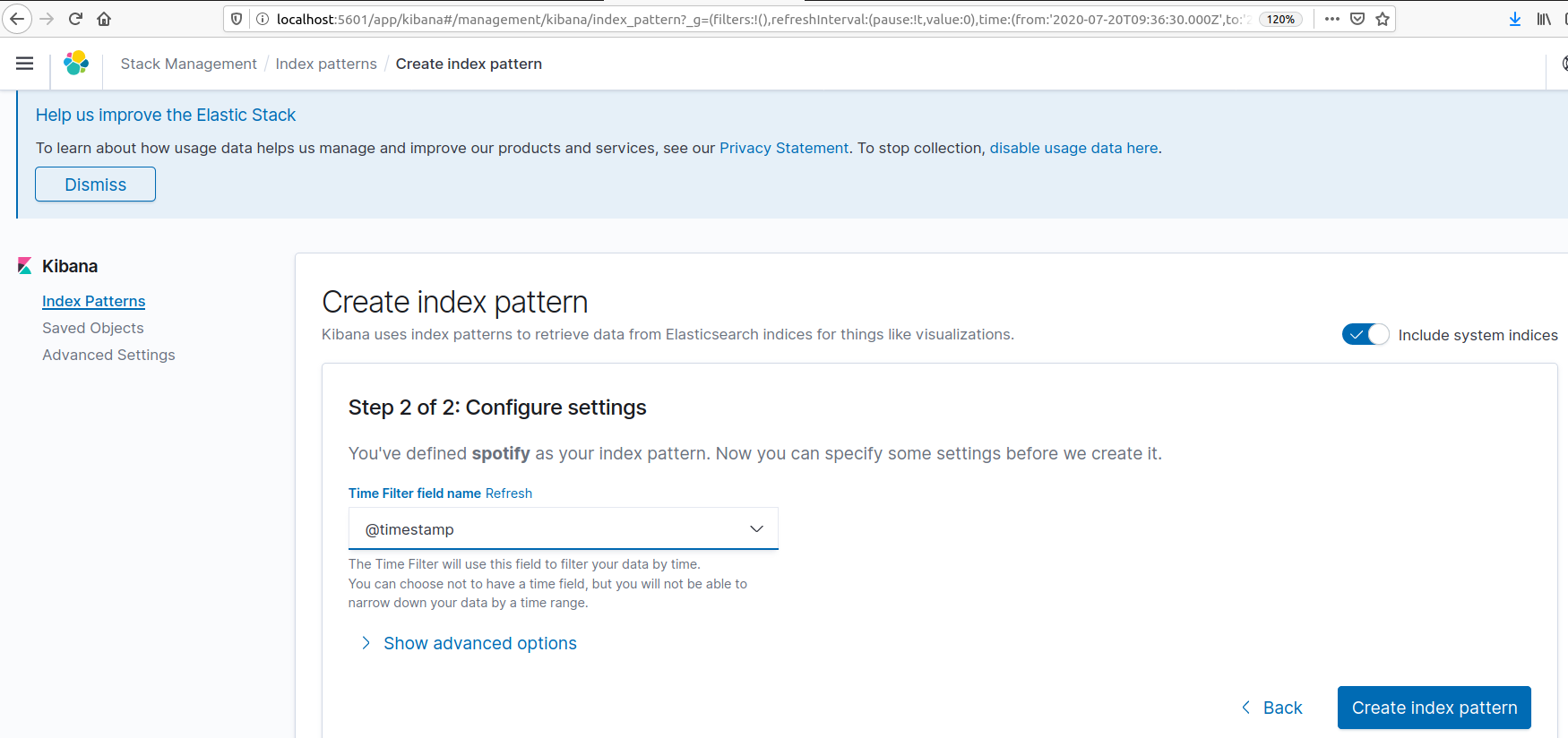
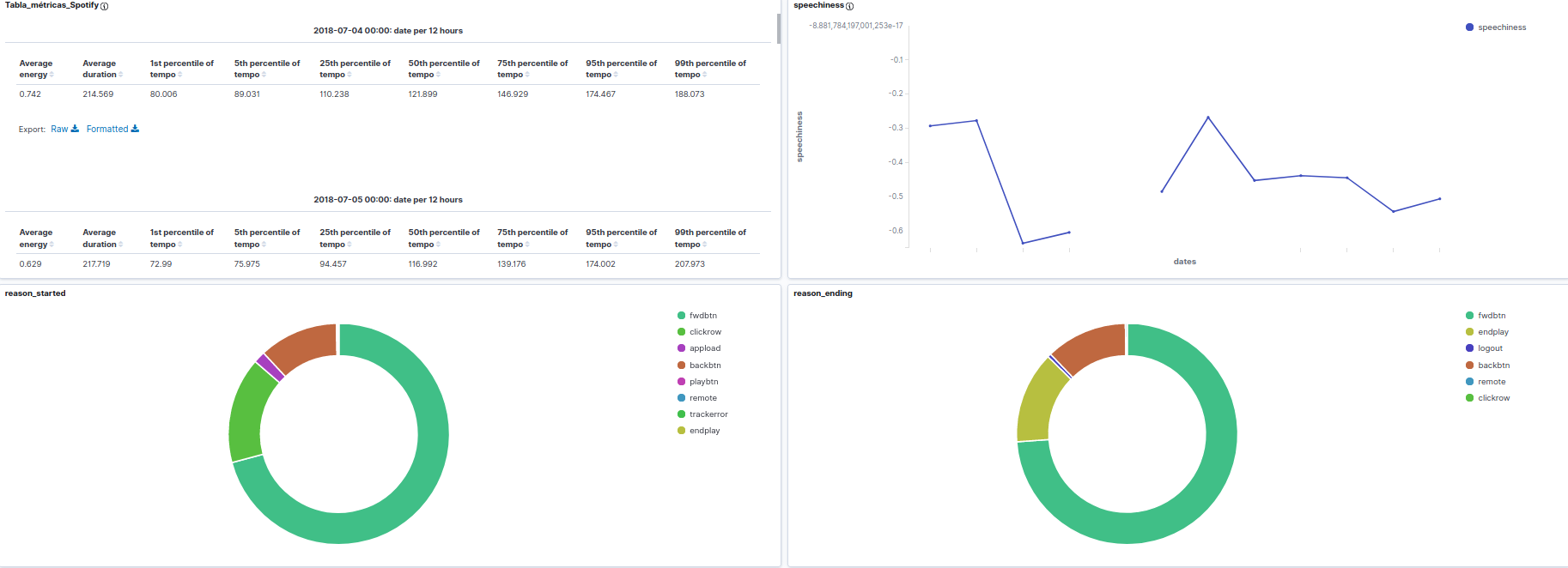


Ilustración : Kibana, índice paso 2

A continuación, debemos parar los contenedores Docker ELK y volver a activarlos (stop y start) para que Kibana sepa cuál es el índice por defecto (que se llama spotify). También es muy importante declarar el índice en minúsculas.

Una vez que hemos vuelto a activar los contenedores ELK, vamos a la sección de visualizaciones y creamos 6 visualizaciones distintas: una tabla con datos sobre la energía de las canciones, un gráfico de línea sobre lo habladas que son las canciones, un gráfico de corona con los motivos por los que ha comenzado una canción, un gráfico de corona con los motivos por los que ha terminado una canción, un mapa de palabras con los contextos más comunes y un gráfico de barras con el valor medio del vector acústico 0 según los días en los que se escucharon las canciones.

El dashboard de Kibana se entrega en los ficheros adjuntos con el nombre de export.ndjson.



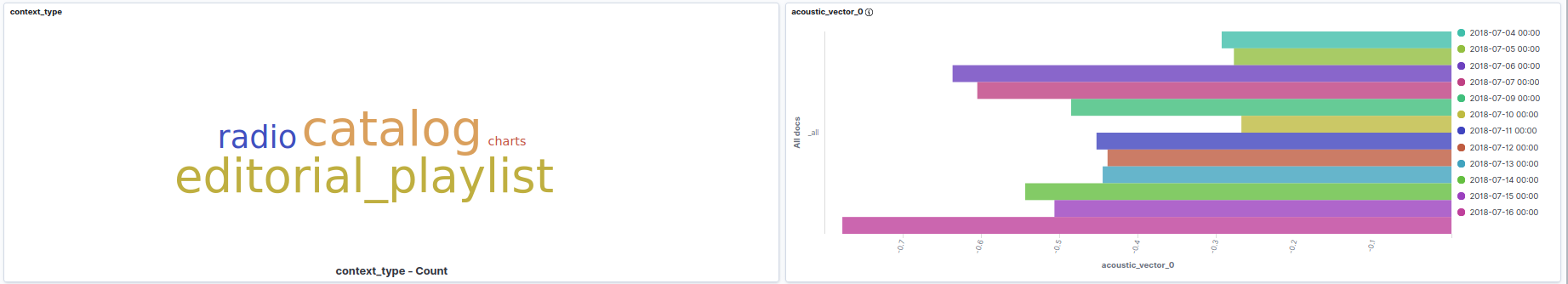


Ilustración :Dashboard en Kibana

Además de estas representaciones gráficas, se puede acudir al índice de figuras para ver todas las representaciones que se han obtenido con el resto de las librerías para visualización.

En los documentos adjuntos existe tanto un fichero exportado desde Kibana como un vídeo mp4 donde se muestran las visualizaciones dentro del dashboard de Kibana.

# 8. Evolución

Hemos cumplido con el alcance planificado en el Gantt, pero es bueno plantearse qué sucesivos pasos se pueden dar para seguir avanzando con este trabajo.

Hay distintas opciones. Se puede pensar en mejorar las prestaciones del modelo, para lo cual seguramente deberíamos aplicar modelos más complejos, como redes neuronales. Sin embargo, las prestaciones del modelo actual son suficientemente buenas (no para competir, pero sí para tener un modelo en producción), por lo que esta opción se dejaría para más adelante.

Por otro lado, se puede plantear que todos los procesos llevados a cabo en este proyecto sean realmente BigData. Si atendemos a las tres uves (volumen, variedad, velocidad), el cuello de botella lo encontramos en el volumen.

Para plantear el procesado y obtención del modelo hemos trabajado con un Jupyter local. Si solventamos este punto trasladando el procesado y modelado a Spark, ya tendríamos un ecosistema completamente BigData (si acaso se puede pensar en el metastore de Hive, pero es que en esta evolución planteada podríamos trabajar con Spark directamente, omitiendo Hive, de forma que Spark almacenase en el HDFS).

Estas serían las ideas propuestas para utilizar esta memoria de base para futuras ampliaciones y evoluciones.

# 9. Conclusiones

Vamos a juntar en este apartado las conclusiones que hemos obtenido tras la realización de esta memoria. Es fundamental que los stacks tecnológicos deben elegirse con base en las necesidades del problema y no a priori. Los datos de entrada son ficheros csv, por lo que son tabulares. Parece razonable pensar en Hive para poder almacenar los ficheros csv que vayamos a generar nosotros.

Enfocando el problema a un entorno real, buscamos que los datos de entrada sean datos reales y no ficticios. Esto nos afecta porque tendremos prestaciones menos eficientes que con datos sintéticos, pero nos acerca al mundo real. Teniendo en cuenta que hemos trabajado con datos reales, las prestaciones del modelo son razonablemente buenas (estamos un 22% por encima de un predictor aleatorio).

En un entorno productivo podríamos tener un HDFS con Hive donde almacenar nuestras grandes tablas y guardar el modelo serializado en algún servidor. En nuestro caso, decidimos guardar el modelo serializado también en el HDFS puesto que el servidor conectado al HDFS está simulado en nuestra máquina local. Así, en lugar de simular un servidor al que nos conectamos, guardamos directamente el modelo en el HDFS y nuestro host sería a su vez ese servidor conectado al HDFS.

Enfrentamos los datos a distintos modelos (regresión lineal, árbol de decisión, random forest, SVM, discriminante Gaussiano y K-NN) para elegir el que mejores métricas ofrece. Aquí estudiamos modelos lineales (regresión logística) y no lineales (los demás). Al explorar modelos no lineales, damos más libertad al modelo, lo que es un punto a nuestro favor.

Otra característica resaltable es que todos los modelos son basados en aprendizaje supervisado, puesto que conocemos el target. Estudiamos entonces distintas métricas en los modelos (exactitud, pérdida logarítmica negativa, ROC AUC).

Balanceamos las clases, pero por generalizar para futuros datos de entrada, utilizamos ROC AUC (que no necesita balanceo de clases). Utilizamos ajuste de hiperparámetros, así como validación cruzada. Las transformaciones y procesados de entrada que hemos realizado son repetibles en nuevos datos y han sido razonadas en cada apartado, para resaltar el objetivo educativo de la memoria.

Hemos eliminado una gran cantidad de outliers al aplicar un filtrado en el rango intercuartílico, pero no afecta a las prestaciones porque disponemos de suficientes datos (como se vio posteriormente en la curva de aprendizaje).

El objetivo del proyecto es trabajar en un proyecto Big Data global, por lo que además de estudiar modelos matemáticos nos preocupamos por desplegar una arquitectura basada en servicios en contenedores Docker donde disponemos de HDFS, Hive (con PostgreSQL de metastore), PySpark y el stack ELK para las visualizaciones.

Nos aseguramos también de que la base de datos relacional que se use para el metastore de Hive sea del tipo de las que hemos estudiado en el curso (en este caso, PostgreSQL, que es de código abierto).

Comprobamos cómo el procesado y exploración de los datos cubre una gran parte del proyecto, siendo estos pasos clave para que el modelo funcione con precisión (son pasos que, aunque parezca que se pueden automatizar, realmente no se puede, porque la calidad del modelo final depende del ajuste que se haya realizado a los datos de partida y a las necesidades de negocio).

También debemos recordar que al utilizar Docker montamos los volúmenes para que el apagado de los contenedores no borre el trabajo previo.

Los ficheros Docker-compose.yml nos facilitan el trabajo al poder levantar varios contenedores relacionados todos a la vez en el mismo script.

Con el namenode y datanode activos, además del YARN (procesos java corriendo), se puede ejecutar Hive. Entrando en el namenode se puede crear a través del sistema de ficheros un directorio y después a través del dfs mover el fichero deseado (el modelo serializado). Aquí vemos claramente la diferencia entre el sistema de ficheros (file system) y el dfs: el sistema de ficheros puede mover, copiar, obtener, etc., ficheros entre un sistema de archivos local y HDFS o cualquier otro sistema de ficheros y viceversa pero cuando se especifica dfs el sistema de ficheros debe estar en dicho formato obligatoriamente (FormacionHadoop.com, 2020).

Respecto a Spark, debemos usar PySpark para ser coherentes con el contenido estudiado en el curso, donde el lenguaje de referencia es Python. Spark nos ayuda a ser más rápidos por el tratamiento en memoria, y especialmente a poder distribuir el procesado. Esto abre la puerta a evolucionar este proyecto hacia un procesado en Spark completo (SparkSQL y SparkML).

Por último, aunque no se haya realizado al procesamiento completo en Spark, sí hemos querido utilizar la salida de Spark como entrada para las visualizaciones finales para mantener un discurso lógico con la arquitectura planteada en la solución del proyecto. Las visualizaciones se han desarrollado utilizando Logstash, Elasticsearch y Kibana, y de nuevo con todos los servicios en Docker.

Aquí hemos visto que, aunque a priori utilizar Docker simplifica la instalación, después los ajustes necesitan más cuidado porque tenemos que entender dos puntos:

1. Las configuraciones que ya vienen definidas en la imagen Docker
2. Los procesos de Docker que afectan al funcionamiento

Como ejemplo del primer punto, hemos tenido que cambiar el pipeline.yml y definir el Spotify.conf.

Como ejemplo del segundo punto, hemos tenido que parar y reactivar los contenedores para que Kibana utilizara el índice por defecto y no lo perdiera entre la creación del índice (leído desde Elasticsearch) y la creación de las visualizaciones.

Por último, en caso de evolucionar el proyecto, no es necesario modificar la parte ELK ni las previas a Spark. El campo de trabajo que se abre ante nosotros está en llevar todo el procesado y modelado que hemos realizado en el Jupyter local a un Jupyter con Spark distribuido, donde además podríamos prescindir de Hive al utilizar Spark en la comunicación con el HDFS.

# Documentos entregados

Los documentos que se entregan en el adjunto de este proyecto son:

* README.md: documento que explica cómo montar el entorno para el proyecto
* Participante.txt: documento que explica los roles que he tomado en este proyecto individual
* Presentación\_TFC.pptx: documento de apoyo para la presentación en la entrega del proyecto
* Ficheros: directorio con todos los documentos necesarios para ejecutar el proyecto
* Planificación Gantt Chart.png: imagen del Gantt con la planificación
* TFC\_Alberto\_Romero.docx: memoria en formato docx
* TFC\_Alberto\_Romero.pdf: memoria en formato pdf
* Spotify\_kibana.mp4: vídeo con el dashboard de Kibana

# Anexo

## Hola-Mundo en Docker

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~**$ sudo docker run hello-world

Stdout:

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

## Notebook de Jupyter en Docker

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~**$ docker run -d -p 8888:8888 --name jupyter --env JUPYTER\_TOKEN=jupyter\_notebook\_token --volume $(pwd)/notebooks:/home/jovyan/work jupyter/datascience-notebook:latest

Stdout:

Unable to find image &apos;jupyter/datascience-notebook:latest&apos; locally

latest: Pulling from jupyter/datascience-notebook

23884877105a: Pulling fs layer

bc38caa0f5b9: Pulling fs layer

2910811b6c42: Pulling fs layer

36505266dcc6: Pulling fs layer

c5f758dc05d3: Pull complete

2f0b81506fdb: Pull complete

441f40fe64ef: Pull complete

fb99bd169219: Pull complete

f48613876bdd: Pull complete

e4f5ba882b7c: Pull complete

c21434c77188: Pull complete

19f19202440b: Pull complete

50c6e016f00b: Pull complete

c9fe8d804c33: Pull complete

7dff026c3fcf: Pull complete

c942425f2c13: Pull complete

a721353ec2be: Pull complete

03458b5c81fe: Pull complete

60c0ac304739: Pull complete

64b7295a171f: Pull complete

1e0768634081: Pull complete

47e8ea323749: Pull complete

1c876a96a3a9: Pull complete

0c1f3d0d7412: Pull complete

79a9adb26259: Pull complete

89e620a4d238: Pull complete

69f1284fdbd2: Pull complete

Digest: sha256:4267ff0245a632f533ed747b98330211246874c28ee49323ad54345026b4029a

Status: Downloaded newer image for jupyter/datascience-notebook:latest

29a8a2f5e95e38a01c07f785191bfc5172a29de0e9d176e1989a3f821e83745b

Y ya tendríamos un notebook de Jupyter dockerizado corriendo en:

http://localhost:8888/?token=jupyter\_notebook\_token[[2]](#footnote-2)

Observamos también que al ser la primera vez que nos descargamos esa imagen, Docker recurre a los pull. Si volvemos a utilizar esa imagen posteriormente, Docker ya tendrá bajada esa imagen en la caché.

Explicación de los parámetros Docker:

-d: detached, así el contenedor queda aparte

-p 8888:8888: asignación de puertos entre host y contenedor

--name jupyter: nombre que le damos para referirnos al servicio

--env JUPYTER\_TOKEN=jupyter\_notebook\_token: contraseña de acceso a la URL del Jupyter

--volume $(pwd)/notebooks:/home/jovyan/work: muy importante, creamos un volumen en el host para que el notebook persista después de cerrar el contenedor Docker

jupyter/datascience-notebook:latest: la imagen que nos queremos traer, es un notebook de Jupyter

## Entorno del notebook de Jupyter del modelo

Antes de continuar de nuevo hacia HDFS y Hive, nos interesa ver qué herramientas hemos estado utilizando. Para eso nos apoyamos en el **comando mágico %pip** que ejecuta el gestor de paquetes pip en el kernel actual.

%pip list

Package Version

---------------------- -------------------

alembic 1.4.2

async-generator 1.10

attrs 19.3.0

backcall 0.1.0

beautifulsoup4 4.9.1

bleach 3.1.5

blinker 1.4

bokeh 2.0.1

Bottleneck 1.3.2

brotlipy 0.7.0

certifi 2020.4.5.1

certipy 0.1.3

cffi 1.14.0

chardet 3.0.4

click 7.1.2

cloudpickle 1.4.1

conda 4.8.2

conda-package-handling 1.6.0

cryptography 2.9.2

cycler 0.10.0

Cython 0.29.19

cytoolz 0.10.1

dask 2.15.0

decorator 4.4.2

defusedxml 0.6.0

dill 0.3.1.1

distributed 2.17.0

entrypoints 0.3

fastcache 1.1.0

fsspec 0.7.4

gmpy2 2.1.0b1

h5py 2.10.0

HeapDict 1.0.1

idna 2.9

imageio 2.8.0

importlib-metadata 1.6.0

ipykernel 5.3.0

ipympl 0.5.6

ipython 7.14.0

ipython-genutils 0.2.0

ipywidgets 7.5.1

jedi 0.17.0

Jinja2 2.11.2

joblib 0.15.1

json5 0.9.0

jsonschema 3.2.0

jupyter-client 6.1.3

jupyter-core 4.6.3

jupyter-telemetry 0.0.5

jupyterhub 1.1.0

jupyterlab 2.1.3

jupyterlab-server 1.1.5

kiwisolver 1.2.0

llvmlite 0.31.0

locket 0.2.0

Mako 1.1.0

MarkupSafe 1.1.1

matplotlib 3.2.1

mistune 0.8.4

mock 4.0.2

mpmath 1.1.0

msgpack 1.0.0

nbconvert 5.6.1

nbformat 5.0.6

networkx 2.4

notebook 6.0.3

numba 0.48.0

numexpr 2.7.1

numpy 1.18.4

oauthlib 3.0.1

olefile 0.46

packaging 20.4

pamela 1.0.0

pandas 1.0.3

pandocfilters 1.4.2

parso 0.7.0

partd 1.1.0

patsy 0.5.1

pexpect 4.8.0

pickleshare 0.7.5

Pillow 7.1.2

pip 20.1

prometheus-client 0.8.0

prompt-toolkit 3.0.5

protobuf 3.11.4

psutil 5.7.0

ptyprocess 0.6.0

pycosat 0.6.3

pycparser 2.20

pycurl 7.43.0.5

Pygments 2.6.1

PyJWT 1.7.1

pyOpenSSL 19.1.0

pyparsing 2.4.7

pyrsistent 0.16.0

PySocks 1.7.1

python-dateutil 2.8.1

python-editor 1.0.4

python-json-logger 0.1.11

pytz 2020.1

PyWavelets 1.1.1

PyYAML 5.3.1

pyzmq 19.0.1

requests 2.23.0

rpy2 3.1.0

ruamel-yaml 0.15.80

ruamel.yaml 0.16.6

ruamel.yaml.clib 0.2.0

scikit-image 0.16.2

scikit-learn 0.22.2.post1

scipy 1.4.1

seaborn 0.10.1

Send2Trash 1.5.0

setuptools 46.1.3.post20200325

simplegeneric 0.8.1

six 1.14.0

sortedcontainers 2.1.0

soupsieve 2.0.1

SQLAlchemy 1.3.17

statsmodels 0.11.1

sympy 1.5.1

tables 3.6.1

tblib 1.6.0

terminado 0.8.3

testpath 0.4.4

toolz 0.10.0

tornado 6.0.4

tqdm 4.45.0

traitlets 4.3.3

typing-extensions 3.7.4.2

tzlocal 2.1

urllib3 1.25.9

vincent 0.4.4

wcwidth 0.1.9

webencodings 0.5.1

wheel 0.34.2

widgetsnbextension 3.5.1

xlrd 1.2.0

zict 2.0.0

zipp 3.1.0

Note: you may need to restart the kernel to use updated packages.

Tabla 27: Entorno

Vemos que el entorno se compone de las herramientas que ya estaban en la imagen Docker del Jupyter.

## Docker-compose para HDFS y Hive

Como ya tenemos construido el Docker-compose para el HDFS y el Docker-compose para Hive, procedemos a ejecutar ambos ficheros en Docker.

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$ docker-compose up -d

Stdout:

Creating network "docker-hadoop-spark-workbench\_default" with the default driver

Pulling namenode (bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8)...

1.1.0-hadoop2.8-java8: Pulling from bde2020/hadoop-namenode

6d827a3ef358: Pull complete

2726297beaf1: Pull complete

7d27bd3d7fec: Pull complete

e61641c845ed: Pull complete

cce4cca5b76b: Pull complete

6826227500b0: Pull complete

c03b117ffd91: Pull complete

821a1547b435: Pull complete

62c556046302: Pull complete

7356cfd2097b: Pull complete

1278a4edba62: Pull complete

90e77cddaa38: Pull complete

83f61e5733ff: Pull complete

ed6bab5146cb: Pull complete

dc277f34c12c: Pull complete

0baa02979410: Pull complete

78e8b8819e1b: Pull complete

08d1331fdbf7: Pull complete

97ad76e49975: Pull complete

7ab15f3d404d: Pull complete

281c3d016950: Pull complete

Digest: sha256:c0714fd74f589e44c1b9d9c800cb9a1379186fcbb65d31aedf3b80cd8a4db285

Status: Downloaded newer image for bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8

Pulling datanode (bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8)...

1.1.0-hadoop2.8-java8: Pulling from bde2020/hadoop-datanode

6d827a3ef358: Already exists

2726297beaf1: Already exists

7d27bd3d7fec: Already exists

e61641c845ed: Already exists

cce4cca5b76b: Already exists

6826227500b0: Already exists

c03b117ffd91: Already exists

821a1547b435: Already exists

62c556046302: Already exists

7356cfd2097b: Already exists

1278a4edba62: Already exists

90e77cddaa38: Already exists

83f61e5733ff: Already exists

ed6bab5146cb: Already exists

dc277f34c12c: Already exists

0baa02979410: Already exists

78e8b8819e1b: Already exists

08d1331fdbf7: Already exists

893e5bb7d312: Pull complete

c36bf523611b: Pull complete

fc7a2a648b07: Pull complete

Digest: sha256:e6d0827f4e3c32d5cb10584963f3aafb5fd8ba4588a7c01e9bd75c834449d850

Status: Downloaded newer image for bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8

Pulling hue (bde2020/hdfs-filebrowser:3.11)...

3.11: Pulling from bde2020/hdfs-filebrowser

f25e451100bc: Pulling fs layer

2dbe4abf311d: Pulling fs layer

7ae2bc99836a: Pulling fs layer

a3ed95caeb02: Pulling fs layer

f1a0a565f855: Pulling fs layer

3f53433f478f: Pulling fs layer

dcb8f963b603: Pulling fs layer

194696c1fe80: Pulling fs layer

8ca254b7aa94: Pulling fs layer

92fe201468f8: Pulling fs layer

081c368134c7: Pulling fs layer

4654ee5f6659: Pulling fs layer

88a820555dbf: Pull complete

14ead1715153: Pull complete

df32fa293aee: Pull complete

f0c9fa3e4fa9: Pull complete

ce0284e53103: Pull complete

cbdb8c564fcc: Pull complete

becd891f4d83: Pull complete

4305be6507a2: Pull complete

cd02abe11589: Pull complete

55b03a36febf: Pull complete

503ca7cfdec1: Pull complete

077f5d65a2f0: Pull complete

16b679fd89e6: Pull complete

787b2d43064e: Pull complete

7decc2d090fb: Pull complete

914d86a00eba: Pull complete

a8ec9c576cb2: Pull complete

4714408faa44: Pull complete

ec27fa202f55: Pull complete

5687b41783e7: Pull complete

538fcca1f5af: Pull complete

8b1a2bff446c: Pull complete

cf63e94ebc3b: Pull complete

Digest: sha256:95b2b8b2e622b474851cb362934592fc22e520f608e891f9aa4be45032b23ccd

Status: Downloaded newer image for bde2020/hdfs-filebrowser:3.11

Creating namenode ... done

Creating docker-hadoop-spark-workbench\_hue\_1 ... done

Creating docker-hadoop-spark-workbench\_datanode\_1 ... done

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$

Ya tenemos creada la infraestructura con los servicios definidos en el Docker-compose.yml. Ahora lanzamos los servicios de Hive con Docker-compose-hive.yml.

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$ docker-compose -f docker-compose-hive.yml up -d namenode hive-metastore-postgresql

Stdout:

Pulling hive-metastore-postgresql (bde2020/hive-metastore-postgresql:2.1.0)...

2.1.0: Pulling from bde2020/hive-metastore-postgresql

5c90d4a2d1a8: Pull complete

22337bfd13a9: Pull complete

c3961b297acc: Pull complete

5a17453338b4: Pull complete

6364e0d7a283: Pull complete

58c25f5c0dad: Pull complete

f0e675ce88d9: Pull complete

10f26c680a34: Pull complete

873d2c220bff: Pull complete

fd10fb78ded6: Pull complete

ff1356ba118b: Pull complete

e030b3f381c4: Pull complete

bd9a5d52d94c: Pull complete

87306d015953: Pull complete

Digest: sha256:2f09ad2f6bc71956668eb0a99769cb43afa2306abea55a65c00c48810416d63d

Status: Downloaded newer image for bde2020/hive-metastore-postgresql:2.1.0

Recreating namenode ... done

Creating docker-hadoop-spark-workbench\_hive-metastore-postgresql\_1 ... done

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$ docker-compose -f docker-compose-hive.yml up -d datanode hive-metastore

Stdout:

Pulling hive-metastore (bde2020/hive:2.1.0-postgresql-metastore)...

2.1.0-postgresql-metastore: Pulling from bde2020/hive

5c68a10e9f3f: Pull complete

ed436531af4d: Pull complete

36ba3ee9b6db: Pull complete

1ab942744935: Pull complete

eaa8cf4c91c8: Pull complete

8c76dcbe31ce: Pull complete

ba11b99d01ff: Pull complete

d106bb94d627: Pull complete

927766f570d1: Pull complete

9df2854d6a8b: Pull complete

c417c679b601: Pull complete

66bf221910ae: Pull complete

9745646856af: Pull complete

80177b7696e0: Pull complete

dcb21c05760c: Pull complete

3a1e3bba5b7f: Pull complete

74c1096139b1: Pull complete

4d80ef79a451: Pull complete

9f3f06303d06: Pull complete

1084509cea85: Pull complete

9957cbbe76cf: Pull complete

dd6291f6ce71: Pull complete

d67219b50ec0: Pull complete

05d5b9c0831a: Pull complete

Digest: sha256:c08e4c07c5d670ccfed2fc5123b2fe536d3678347f65f46629b8d2d98564c1d5

Status: Downloaded newer image for bde2020/hive:2.1.0-postgresql-metastore

namenode is up-to-date

Creating hive-metastore ... done

Recreating docker-hadoop-spark-workbench\_datanode\_1 ... done

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$ docker-compose -f docker-compose-hive.yml up -d hive-server

Stdout:

Creating hive-server ... done

Stdin:

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$ docker-compose -f docker-compose-hive.yml up -d hue

Stdout:

Recreating docker-hadoop-spark-workbench\_spark-worker\_1 ... done

**ubuntu@ubuntu-VirtualBox**:**~/bigdata/docker-hadoop-spark-workbench**$

Ya hemos lanzado los servicios de Hive.

## Trabajando con Hive y el modelo

Entramos en el Docker de Hive.

Stdin:

**buntu@ubuntu-VirtualBox**:**~**$ **docker exec -it hive-server /bin/bash**

root@0ad25d6bd012:/opt# ls

hadoop-2.7.1 hive

root@0ad25d6bd012:/opt# cd hive

root@0ad25d6bd012:/opt/hive# ls

LICENSE NOTICE README.txt RELEASE\_NOTES.txt bin conf examples hcatalog jdbc lib scripts

root@0ad25d6bd012:/opt/hive# cd bin

root@0ad25d6bd012:/opt/hive/bin# ls

beeline beeline.cmd ext hive hive-config.cmd hive-config.sh hive.cmd hiveserver2 hplsql hplsql.cmd metatool schematool

**root@0ad25d6bd012:/opt/hive/bin# hive**

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.1/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in file:/opt/hive/conf/hive-log4j2.properties Async: true

Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.

hive>

**Creamos la tabla**

CREATE TABLE IF NOT EXISTS x\_train\_arv

(session\_position double,

session\_length double,

hist\_user\_behavior\_is\_shuffle double,

hour\_of\_day double,

duration double,

release\_year double,

us\_popularity\_estimate double,

acousticness double,

beat\_strength double,

bounciness double,

danceability double,

dyn\_range\_mean double,

energy double,

flatness double,

instrumentalness double,

key double,

liveness double,

loudness double,

mechanism double,

organism double,

speechiness double,

tempo double,

valence double,

acoustic\_vector\_0 double,

acoustic\_vector\_1 double,

acoustic\_vector\_2 double,

acoustic\_vector\_3 double,

acoustic\_vector\_4 double,

acoustic\_vector\_5 double,

acoustic\_vector\_6 double,

acoustic\_vector\_7 double,

context\_type\_user\_collection double,

hist\_user\_behavior\_reason\_start\_fwdbtn double,

hist\_user\_behavior\_reason\_start\_trackdone double,

mode\_minor double)

COMMENT 'Detalles de x\_train'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

**La creamos en Hive (escribimos esa sentencia)**

hive> CREATE TABLE IF NOT EXISTS x\_train\_arv

> (session\_position double,

> session\_length double,

> hist\_user\_behavior\_is\_shuffle double,

> hour\_of\_day double,

> duration double,

> release\_year double,

> us\_popularity\_estimate double,

> acousticness double,

> beat\_strength double,

> bounciness double,

> danceability double,

> dyn\_range\_mean double,

> energy double,

> flatness double,

> instrumentalness double,

> key double,

> liveness double,

> loudness double,

> mechanism double,

> organism double,

> speechiness double,

> tempo double,

> valence double,

> acoustic\_vector\_0 double,

> acoustic\_vector\_1 double,

> acoustic\_vector\_2 double,

> acoustic\_vector\_3 double,

> acoustic\_vector\_4 double,

> acoustic\_vector\_5 double,

> acoustic\_vector\_6 double,

> acoustic\_vector\_7 double,

> context\_type\_user\_collection double,

> hist\_user\_behavior\_reason\_start\_fwdbtn double,

> hist\_user\_behavior\_reason\_start\_trackdone double,

> mode\_minor double)

> COMMENT &apos;Detalles de x\_train&apos;

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY &apos; |&apos;

> LINES TERMINATED BY &apos;\n&apos;

> STORED AS TEXTFILE;

OK

Time taken: 0.138 seconds

hive>

**Creamos dentro del Docker Hive-server el directorio para los datos**

**ubuntu@ubuntu-VirtualBox**:**~**$ docker exec -it hive-server /bin/bash

root@0ad25d6bd012:/# mkdir datos -p

root@0ad25d6bd012:/# exit

exit

**Copiamos desde fuera del contenedor Docker el csv**

**ubuntu@ubuntu-VirtualBox**:**~**$ docker cp notebooks/X\_train.csv hive-server:datos/X\_train.csv

**ubuntu@ubuntu-VirtualBox**:**~**$ docker cp notebooks/X\_train.csv hive-server:datos/X\_test.csv

**ubuntu@ubuntu-VirtualBox**:**~**$ docker cp notebooks/X\_train.csv hive-server:datos/y\_train.csv

**ubuntu@ubuntu-VirtualBox**:**~**$ docker cp notebooks/X\_train.csv hive-server:datos/y\_test.csv

**ubuntu@ubuntu-VirtualBox**:**~**$

**Entramos en el contenedor**

**ubuntu@ubuntu-VirtualBox**:**~**$ docker exec -it hive-server /bin/bash

root@0ad25d6bd012:/opt# ls /datos

X\_test.csv X\_train.csv y\_test.csv y\_train.csv

**Vamos a Hive y cargamos el csv**

root@0ad25d6bd012:/# /opt/hive/bin/hive

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.1/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in file:/opt/hive/conf/hive-log4j2.properties Async: true

Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.

**hive> LOAD DATA LOCAL INPATH &apos;/datos/X\_train.csv&apos; OVERWRITE INTO TABLE x\_train\_arv;**

Loading data to table default.x\_train\_arv

OK

Time taken: 1.417 seconds

hive>

**Lo comprobamos.**

hive> alter table x\_train\_arv set tblproperties("skip.header.line.count"="1");

OK

Time taken: 0.221 seconds

hive> select \* from x\_train\_arv limit 3;

OK

x\_train\_arv.session\_position x\_train\_arv.session\_length x\_train\_arv.hist\_user\_behavior\_is\_shuffle x\_train\_arv.hour\_of\_day x\_train\_arv.duration x\_train\_arv.release\_year x\_train\_arv.us\_popularity\_estimate x\_train\_arv.acousticness x\_train\_arv.beat\_strength x\_train\_arv.bounciness x\_train\_arv.danceability x\_train\_arv.dyn\_range\_mean x\_train\_arv.energy x\_train\_arv.flatness x\_train\_arv.instrumentalness x\_train\_arv.key x\_train\_arv.liveness x\_train\_arv.loudness x\_train\_arv.mechanism x\_train\_arv.organism x\_train\_arv.speechiness x\_train\_arv.tempo x\_train\_arv.valence x\_train\_arv.acoustic\_vector\_0 x\_train\_arv.acoustic\_vector\_1 x\_train\_arv.acoustic\_vector\_2 x\_train\_arv.acoustic\_vector\_3 x\_train\_arv.acoustic\_vector\_4 x\_train\_arv.acoustic\_vector\_5 x\_train\_arv.acoustic\_vector\_6 x\_train\_arv.acoustic\_vector\_7 x\_train\_arv.context\_type\_user\_collection x\_train\_arv.hist\_user\_behavior\_reason\_start\_fwdbtn x\_train\_arv.hist\_user\_behavior\_reason\_start\_trackdone x\_train\_arv.mode\_minor

0.3888888888888889 0.5 0.0 0.8260869565217391 0.6147336166487314 0.8000000000000114 0.32778037608676414 0.0034181266285669533 0.16518980314154857 0.21448738925018230.35871985114819815 0.16500284765926287 0.8691195798034682 0.4078137163479916 0.0029768771516864873 0.8181818181818182 0.716809071725535 0.7964389128787785 0.6112209385727659 0.2958617885615159 0.08951773480321118 0.6836376154906041 0.1138100147305213 0.49254654950539406 0.8236571571373363 0.45074821644906893 0.33817395283150964 0.28581813410883494 0.836558178367437 0.7446815134555956 0.8984530613407007 1.0 1.0 0.0 1.0

0.6666666666666665 0.40000000000000013 0.0 0.7391304347826086 0.7443828291218335 0.8000000000000114 0.8853032226453479 0.3016911143999167 0.4983725671784943 0.5727070748373723 0.6051126772750499 0.41416921364873305 0.8496699021035937 0.5840870392834834 0.008401802375759701 0.18181818181818182 0.1369269056436372 0.78169771388365920.6721717004661389 0.3191239892849513 0.3527395728132729 0.6709489594880554 0.3363528423853343 0.1063413253916431 0.5328431135560115 0.46869835453744474 0.6669568428000521 0.351254096704463 0.3660765784260654 0.1294392830731268 0.6130851231331751 1.0 1.0 0.0 0.0

0.27777777777777773 1.0 0.0 0.6521739130434783 0.45358705598059673 1.0 0.8527575480871974 0.1929636476590463 0.38517150515078635 0.46988069592006754 0.5468460426896316 0.3440439218955228 0.8454078144109684 0.07765990806987855 4.4756492494210677E-7 0.4545454545454546 0.5846365397495645 0.697970990707411 0.4574239635661831 0.4336733277350877 0.5703977521463734 0.5856496479730318 0.6453659385743429 0.12570633921843455 0.3356187427979069 0.5889199974424225 0.7221514019347721 0.369093505272919950.3421078533653956 0.09863191045053199 0.5299896369928825 1.0 0.0 1.0 0.0

Time taken: 0.166 seconds, Fetched: 3 row(s)

hive>

**Lo comprobamos con la primera fila.**

hive> select \* from x\_train\_arv limit 1;

OK

x\_train\_arv.session\_position x\_train\_arv.session\_length x\_train\_arv.hist\_user\_behavior\_is\_shuffle x\_train\_arv.hour\_of\_day x\_train\_arv.duration x\_train\_arv.release\_year x\_train\_arv.us\_popularity\_estimate x\_train\_arv.acousticness x\_train\_arv.beat\_strength x\_train\_arv.bounciness x\_train\_arv.danceability x\_train\_arv.dyn\_range\_mean x\_train\_arv.energy x\_train\_arv.flatness x\_train\_arv.instrumentalness x\_train\_arv.key x\_train\_arv.liveness x\_train\_arv.loudness x\_train\_arv.mechanism x\_train\_arv.organism x\_train\_arv.speechiness x\_train\_arv.tempo x\_train\_arv.valence x\_train\_arv.acoustic\_vector\_0 x\_train\_arv.acoustic\_vector\_1 x\_train\_arv.acoustic\_vector\_2 x\_train\_arv.acoustic\_vector\_3 x\_train\_arv.acoustic\_vector\_4 x\_train\_arv.acoustic\_vector\_5 x\_train\_arv.acoustic\_vector\_6 x\_train\_arv.acoustic\_vector\_7 x\_train\_arv.context\_type\_user\_collection x\_train\_arv.hist\_user\_behavior\_reason\_start\_fwdbtn x\_train\_arv.hist\_user\_behavior\_reason\_start\_trackdone x\_train\_arv.mode\_minor

0.3888888888888889 0.5 0.0 0.8260869565217391 0.6147336166487314 0.8000000000000114 0.32778037608676414 0.0034181266285669533 0.16518980314154857 0.21448738925018230.35871985114819815 0.16500284765926287 0.8691195798034682 0.4078137163479916 0.0029768771516864873 0.8181818181818182 0.716809071725535 0.7964389128787785 0.6112209385727659 0.2958617885615159 0.08951773480321118 0.6836376154906041 0.1138100147305213 0.49254654950539406 0.8236571571373363 0.45074821644906893 0.33817395283150964 0.28581813410883494 0.836558178367437 0.7446815134555956 0.8984530613407007 1.0 1.0 0.0 1.0

Time taken: 0.214 seconds, Fetched: 1 row(s)

hive>

**Creamos en Hive la tabla x\_test\_arv.**

CREATE TABLE IF NOT EXISTS x\_test\_arv

(session\_position double,

session\_length double,

hist\_user\_behavior\_is\_shuffle double,

hour\_of\_day double,

duration double,

release\_year double,

us\_popularity\_estimate double,

acousticness double,

beat\_strength double,

bounciness double,

danceability double,

dyn\_range\_mean double,

energy double,

flatness double,

instrumentalness double,

key double,

liveness double,

loudness double,

mechanism double,

organism double,

speechiness double,

tempo double,

valence double,

acoustic\_vector\_0 double,

acoustic\_vector\_1 double,

acoustic\_vector\_2 double,

acoustic\_vector\_3 double,

acoustic\_vector\_4 double,

acoustic\_vector\_5 double,

acoustic\_vector\_6 double,

acoustic\_vector\_7 double,

context\_type\_user\_collection double,

hist\_user\_behavior\_reason\_start\_fwdbtn double,

hist\_user\_behavior\_reason\_start\_trackdone double,

mode\_minor double)

COMMENT 'Detalles de x\_test'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/datos/X\_test.csv'

OVERWRITE INTO TABLE x\_test\_arv;

ALTER TABLE x\_test\_arv SET TBLPROPERTIES("skip.header.line.count"="1");

**Lo aplicamos en Hive.**

hive> CREATE TABLE IF NOT EXISTS x\_test\_arv

> (session\_position double,

> session\_length double,

> hist\_user\_behavior\_is\_shuffle double,

> hour\_of\_day double,

> duration double,

> release\_year double,

> us\_popularity\_estimate double,

> acousticness double,

> beat\_strength double,

> bounciness double,

> danceability double,

> dyn\_range\_mean double,

> energy double,

> flatness double,

> instrumentalness double,

> key double,

> liveness double,

> loudness double,

> mechanism double,

> organism double,

> speechiness double,

> tempo double,

> valence double,

> acoustic\_vector\_0 double,

> acoustic\_vector\_1 double,

> acoustic\_vector\_2 double,

> acoustic\_vector\_3 double,

> acoustic\_vector\_4 double,

> acoustic\_vector\_5 double,

> acoustic\_vector\_6 double,

> acoustic\_vector\_7 double,

> context\_type\_user\_collection double,

> hist\_user\_behavior\_reason\_start\_fwdbtn double,

> hist\_user\_behavior\_reason\_start\_trackdone double,

> mode\_minor double)

> COMMENT &apos;Detalles de x\_test&apos;

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY &apos;|&apos;

> LINES TERMINATED BY &apos;\n&apos;

> STORED AS TEXTFILE;

OK

Time taken: 0.429 seconds

hive> LOAD DATA LOCAL INPATH &apos;/datos/X\_test.csv&apos;

> OVERWRITE INTO TABLE x\_test\_arv;

Loading data to table default.x\_test\_arv

OK

Time taken: 0.525 seconds

hive>

hive> alter table x\_test\_arv set tblproperties("skip.header.line.count"="1");

OK

Time taken: 0.236 seconds

hive>

**Seleccionamos las primeras filas almacenadas.**

hive> select \* from x\_test\_arv limit 3;

OK

x\_test\_arv.session\_position x\_test\_arv.session\_length x\_test\_arv.hist\_user\_behavior\_is\_shuffle x\_test\_arv.hour\_of\_day x\_test\_arv.duration x\_test\_arv.release\_year x\_test\_arv.us\_popularity\_estimate x\_test\_arv.acousticness x\_test\_arv.beat\_strength x\_test\_arv.bounciness x\_test\_arv.danceability x\_test\_arv.dyn\_range\_mean x\_test\_arv.energy x\_test\_arv.flatness x\_test\_arv.instrumentalness x\_test\_arv.key x\_test\_arv.liveness x\_test\_arv.loudness x\_test\_arv.mechanism x\_test\_arv.organism x\_test\_arv.speechiness x\_test\_arv.tempo x\_test\_arv.valence x\_test\_arv.acoustic\_vector\_0 x\_test\_arv.acoustic\_vector\_1 x\_test\_arv.acoustic\_vector\_2 x\_test\_arv.acoustic\_vector\_3 x\_test\_arv.acoustic\_vector\_4 x\_test\_arv.acoustic\_vector\_5 x\_test\_arv.acoustic\_vector\_6 x\_test\_arv.acoustic\_vector\_7 x\_test\_arv.context\_type\_user\_collection x\_test\_arv.hist\_user\_behavior\_reason\_start\_fwdbtn x\_test\_arv.hist\_user\_behavior\_reason\_start\_trackdone x\_test\_arv.mode\_minor

0.3888888888888889 0.5 0.0 0.8260869565217391 0.6147336166487314 0.8000000000000114 0.32778037608676414 0.0034181266285669533 0.16518980314154857 0.21448738925018230.35871985114819815 0.16500284765926287 0.8691195798034682 0.4078137163479916 0.0029768771516864873 0.8181818181818182 0.716809071725535 0.7964389128787785 0.6112209385727659 0.2958617885615159 0.08951773480321118 0.6836376154906041 0.1138100147305213 0.49254654950539406 0.8236571571373363 0.45074821644906893 0.33817395283150964 0.28581813410883494 0.836558178367437 0.7446815134555956 0.8984530613407007 1.0 1.0 0.0 1.0

0.6666666666666665 0.40000000000000013 0.0 0.7391304347826086 0.7443828291218335 0.8000000000000114 0.8853032226453479 0.3016911143999167 0.4983725671784943 0.5727070748373723 0.6051126772750499 0.41416921364873305 0.8496699021035937 0.5840870392834834 0.008401802375759701 0.18181818181818182 0.1369269056436372 0.78169771388365920.6721717004661389 0.3191239892849513 0.3527395728132729 0.6709489594880554 0.3363528423853343 0.1063413253916431 0.5328431135560115 0.46869835453744474 0.6669568428000521 0.351254096704463 0.3660765784260654 0.1294392830731268 0.6130851231331751 1.0 1.0 0.0 0.0

0.27777777777777773 1.0 0.0 0.6521739130434783 0.45358705598059673 1.0 0.8527575480871974 0.1929636476590463 0.38517150515078635 0.46988069592006754 0.5468460426896316 0.3440439218955228 0.8454078144109684 0.07765990806987855 4.4756492494210677E-7 0.4545454545454546 0.5846365397495645 0.697970990707411 0.4574239635661831 0.4336733277350877 0.5703977521463734 0.5856496479730318 0.6453659385743429 0.12570633921843455 0.3356187427979069 0.5889199974424225 0.7221514019347721 0.369093505272919950.3421078533653956 0.09863191045053199 0.5299896369928825 1.0 0.0 1.0 0.0

Time taken: 0.27 seconds, Fetched: 3 row(s)

hive>

**Creamos y almacenamos y\_train.csv en Hive.**

CREATE TABLE IF NOT EXISTS y\_train\_arv

(not\_skipped tinyint)

COMMENT 'Detalles de y\_train'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/datos/y\_train.csv'

OVERWRITE INTO TABLE y\_train\_arv;

alter table y\_train\_arv set tblproperties("skip.header.line.count"="1");

hive> CREATE TABLE IF NOT EXISTS y\_train\_arv

> (not\_skipped tinyint)

> COMMENT &apos;Detalles de y\_train&apos;

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY &apos;|&apos;

> LINES TERMINATED BY &apos;\n&apos;

> STORED AS TEXTFILE;

OK

Time taken: 2.197 seconds

hive> LOAD DATA LOCAL INPATH &apos;/datos/y\_train.csv&apos;

> OVERWRITE INTO TABLE y\_train\_arv;

Loading data to table default.y\_train\_arv

OK

Time taken: 0.438 seconds

hive> alter table y\_train\_arv set tblproperties("skip.header.line.count"="1");

OK

Time taken: 0.213 seconds

hive> select \* from y\_train\_arv limit 3;

OK

y\_train\_arv.not\_skipped

0

0

0

Time taken: 0.152 seconds, Fetched: 3 row(s)

hive> select \* from y\_train\_arv where not\_skipped = 1 limit 3;

OK

y\_train\_arv.not\_skipped

1

1

1

Time taken: 0.494 seconds, Fetched: 3 row(s)

hive>

**Creamos y almacenamos y\_train.csv.**

CREATE TABLE IF NOT EXISTS y\_test\_arv

(not\_skipped tinyint)

COMMENT 'Detalles de y\_test'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/datos/y\_test.csv'

OVERWRITE INTO TABLE y\_test\_arv;

alter table y\_test\_arv settblproperties("skip.header.line.count"="1");

Lo ejecutamos en Hive.

hive> CREATE TABLE IF NOT EXISTS y\_test\_arv

> (not\_skipped tinyint)

> COMMENT &apos;Detalles de y\_test&apos;

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY &apos;|&apos;

> LINES TERMINATED BY &apos;\n&apos;

> STORED AS TEXTFILE;

OK

Time taken: 0.103 seconds

hive> LOAD DATA LOCAL INPATH &apos;/datos/y\_test.csv&apos;

> OVERWRITE INTO TABLE y\_test\_arv;

Loading data to table default.y\_test\_arv

OK

Time taken: 0.298 seconds

hive> alter table y\_test\_arv set tblproperties("skip.header.line.count"="1");

OK

Time taken: 0.122 seconds

hive> select \* from y\_test\_arv limit 3;

OK

y\_test\_arv.not\_skipped

0

0

0

Time taken: 0.117 seconds, Fetched: 3 row(s)

hive> select \* from y\_test\_arv where not\_skipped = 1 limit 3;

OK

y\_test\_arv.not\_skipped

1

1

1

Time taken: 0.123 seconds, Fetched: 3 row(s)

hive>

**Almacenamos el modelo en el HDFS.**

**ubuntu@ubuntu-VirtualBox:~$ docker exec -it namenode /bin/bash**

root@79c6bb46ab74:/# mkdir input

root@79c6bb46ab74:/# exit

exit

**ubuntu@ubuntu-VirtualBox**:**~**$ docker cp notebooks/Alberto\_Romero\_modelo\_serializado\_joblib.sav namenode:input

**ubuntu@ubuntu-VirtualBox**:**~**$ docker cp notebooks/Alberto\_Romero\_Proyecto.ipynb namenode:input

**ubuntu@ubuntu-VirtualBox**:**~**$ docker exec -it namenode /bin/bash

root@79c6bb46ab74:/# ls input

Alberto\_Romero\_Proyecto.ipynb Alberto\_Romero\_modelo\_serializado\_joblib.sav

root@79c6bb46ab74:/# hadoop fs -mkdir -p input

root@79c6bb46ab74:/# hdfs dfs -put ./input/\* input

root@79c6bb46ab74:/#

**Comprobamos que tenemos el Jupyter y el modelo serializado en el HDFS.**

root@79c6bb46ab74:/# hdfs dfs -ls input

Found 2 items

-rw-r--r-- 3 root supergroup 1606432 2020-06-09 09:41 input/Alberto\_Romero\_Proyecto.ipynb

-rw-r--r-- 3 root supergroup 24616 2020-06-09 09:41 input/Alberto\_Romero\_modelo\_serializado\_joblib.sav

root@79c6bb46ab74:/#

Es correcto, ya tenemos el modelo serializado en el HDFS, así como los csv obtenidos en el procesado del modelo en Hive.

# Referencias

aicrowd. (8 de enero de 2019). *aicrowd Spotify*. Obtenido de https://www.aicrowd.com/challenges/spotify-sequential-skip-prediction-challenge-old/dataset\_files

Albon, C. (20 de diciembre de 2017). *chrisalbon.com*. Obtenido de https://chrisalbon.com/machine\_learning/model\_evaluation/plot\_the\_learning\_curve/

Anita, O. (20 de agosto de 2019). *Heartbeat.fritz.ai*. Obtenido de https://heartbeat.fritz.ai/seaborn-heatmaps-13-ways-to-customize-correlation-matrix-visualizations-f1c49c816f07

Apache.org. (junio de 2020). *Hadoop*. Obtenido de https://hadoop.apache.org/

Apache.org. (junio de 2020). *Hive*. Obtenido de https://hive.apache.org/

Badr, W. (22 de febrero de 2019). *TowardsDataScience.com*. Obtenido de https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb

*Docker*. (junio de 2020). Obtenido de URL corporativa: https://www.docker.com/

*FormacionHadoop.com*. (junio de 2020). Obtenido de https://formacionhadoop.com/aulavirtual/mod/forum/discuss.php?d=144

Gajawad, S. K. (4 de octubre de 2019). *TowardsDataScience.com*. Obtenido de https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223

Jupyter.org. (junio de 2020). *Jupyter.org*. Obtenido de https://jupyter.org/documentation

Narkhede, S. (26 de junio de 2018). *TowardsDataScience.com*. Obtenido de https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Python3. (junio de 2020). *Python3*. Obtenido de https://www.python.org/download/releases/3.0/

S., E. (20 de mayo de 2020). *How to install Docker in Ubuntu*. Obtenido de https://www.hostinger.com/tutorials/how-to-install-docker-on-ubuntu

Sharma, N. (22 de mayo de 2018). *towardsdatascience.com*. Obtenido de https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba

Thibaut, B. (22 de marzo de 2017). *GitLab*. Obtenido de https://gitlab.enseeiht.fr/tboissin/projet\_cloud/tree/a4457659d7f9c0ca0ce23412ffdc5c0bc39df6e9/docker-hadoop-spark-workbench

Ubuntu.com. (junio de 2020). *Ubuntu*. Obtenido de https://ubuntu.com/desktop/developers

*wikipedia Spotify*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Spotify

1. Las sentencias con fuente Garamond representan ejecuciones en la máquina virtual con sistema operativo Ubuntu. También se identifican fácilmente por incluir el prompt. [↑](#footnote-ref-1)
2. Si no se incluye el token en la URL, se puede introducir directamente al solicitarlo Jupyter al abrirse. [↑](#footnote-ref-2)