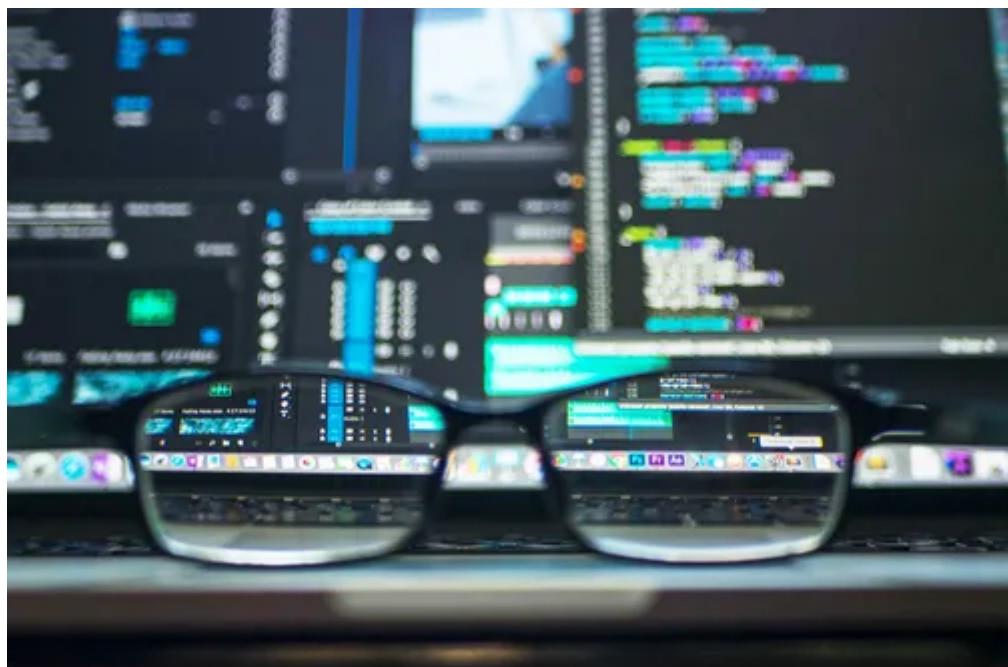


Proyecto Inteligencia Artificial y Big Data

Grupo B



Héctor Fernández Pimienta, Ignacio López
García y Francisco José Puertas Teba

Introducción y Justificación de la Arquitectura de Big Data.....	4
Propósito y Alcance del Documento.....	4
Justificación de la Arquitectura Distribuida: Spark Moderno.....	4
Definición del Modelo Base: K-Means como Heurística Sólida.....	5
Análisis Exploratorio de Datos (EDA) y Calidad.....	5
Perfilado de Datos y Distribuciones Clave.....	6
Calidad de Datos y Tratamiento de Outliers.....	6
Ingeniería de Datos y Pipeline de Ingesta (Batch).....	7
Configuración de Spark y el Pipeline de Limpieza.....	7
Transición y Almacenamiento Intermedio.....	7
Ingeniería de Características y Linaje.....	8
Selección y Estandarización de Features.....	8
Linaje y Estrategia de Particionado.....	9
Modelo base Reproducible y Evaluación.....	9
Entrenamiento y Garantía de Reproducibilidad.....	9
Evaluación y Métricas Mínimas.....	10
Validación Adecuada (k-fold Estratificado – Plan).....	11
Gestión de Artefactos (MLOps) y Pruebas.....	11
Plan de Adopción del MLflow (Gestión de Artefactos).....	11
Framework de Pruebas (Pytest y Data Tests).....	12
Diagrama de Flujo del Pipeline Implementado.....	13

Introducción y Justificación de la Arquitectura de Big Data.

Propósito y Alcance del Documento.

El presente informe marca la conclusión exitosa de la Fase 2, enfocada en la implementación práctica y productivable de nuestro pipeline de datos. Nuestro objetivo central ha sido construir una arquitectura capaz de procesar el vasto catálogo musical, cumpliendo con los requisitos de escalabilidad (R3) y trazabilidad de artefactos.

En este documento, detallamos:

1. La justificación de nuestra elección de features clave, basada en el Análisis Exploratorio (EDA).
2. La mecánica del pipeline de ingesta y limpieza implementando en Pipeline.py con PySpark.
3. La estrategia de Feature Engineering y estandarización utilizada en Entrenamiento.py.
4. La definición y evaluación de nuestro Modelo Base K-Means reproducible.
5. La planificación de MLOps (MLflow) y el framework de pruebas.

Justificación de la Arquitectura Distribuida: Spark Moderno

Hemos tomado la decisión estratégica de implementar la arquitectura basada en Apache Spark desde el inicio del proyecto. Esta elección no fue una alternativa, sino la solución fundamental que establecimos para abordar el desafío del volumen de datos musicales, cumpliendo de manera directa con el requisito de escalabilidad del Feature Engineering (R3).

- **Necesidad del Big Data:** El catálogo musical es vasto y en constante crecimiento. Una solución basada únicamente en librerías monolíticas (como Pandas) limitaría nuestro proyecto a una sola máquina y a muestras de datos

reducidas. La arquitectura con Spark nos permite operar con el concepto de procesamiento distribuido de forma nativa.

- **Diseño para Escalabilidad (R3):** El componente iniciar_spark() en Pipeline.py configura nuestro entorno de ejecución. Aunque en esta fase inicial utilizamos un clúster virtual local (master("local[*]")), el código está intrínsecamente escrito en PySpark, lo que significa que la transición a un clúster de producción (como EMR o Dataproc) será fluida y directa, sin necesidad de reescribir la lógica central del pipeline.
- **Linaje y Eficiencia:** El uso de Spark optimiza la limpieza y el filtrado de outliers, que son las etapas más costosas en términos de cómputo. Además, nos permite integrar fácilmente formatos columnares como Parquet, cruciales para el linaje de datos.

Definición del Modelo Base: K-Means como Heurística Sólida.

Hemos implementado un Modelo Base Heurístico basado en Clustering K-Means (implementado en Entrenamiento.py). La justificación es doble:

6. **Mitigación de Sesgos (R2):** Al utilizar K-Means sobre características acústicas puras (tempo, danceability, etc.), eliminamos el sesgo de popularidad inherente a los modelos colaborativos iniciales. La recomendación se basa en la sonoridad, no en la fama de la canción.
7. **Agrupamiento Objetivo:** K-Means agrupa canciones con features similares en clusters homogéneos. Esto nos permite recomendar canciones del mismo cluster que la canción de entrada, logrando un mecanismo de descubrimiento objetivo y diversificado.

Análisis Exploratorio de Datos (EDA) y Calidad

Antes de codificar el pipeline, realizamos un EDA minucioso para establecer las reglas de limpieza y feature engineering.

Perfilado de Datos y Distribuciones Clave

Nuestro análisis se centró en las cinco features acústicas que definen la similitud:

- ***danceability y valence***: Estas variables, que representan el ritmo y el estado de ánimo (positividad), suelen tener distribuciones cercanas a la normal o ligeramente sesgadas. Observamos una concentración de canciones con valores medios-altos, lo que indica que gran parte del catálogo está optimizado para el ánimo positivo y el baile.
- ***tempo***: Esta variable (BPM) mostró un amplio rango, con una distribución bimodal (picos alrededor de 90-110 BPM y 120-130 BPM), lo que nos obligó a aplicar filtros estrictos para manejar los valores extremos.
- ***instrumentalness y speechiness***: Estas features presentan una alta asimetría positiva, con la mayoría de los valores concentrados en cero. El escalado se vuelve crucial aquí para que la poca varianza existente pueda influir en el cálculo de distancias del modelo.

Calidad de Datos y Tratamiento de Outliers

La calidad de nuestros datos es crítica; el ruido en el feature set se traduce directamente en clusters de baja cohesión. Hemos garantizado la integridad mediante estas reglas, ejecutadas por PySpark en la función `limpieza_datos()`:

- **Garantía de No-Nulos**: Utilizamos la operación `df_spotify.na.drop()` directamente sobre el DataFrame de Spark. Esta es una medida estricta que asegura que cada registro que pasa a la capa de entrenamiento tiene valores completos en todas las columnas, eliminando cualquier ambigüedad en los cálculos del modelo.
- **Filtro de Duración Excesiva**: Aplicamos `duration_ms < 1200000` (menos de 20 minutos). Este umbral fue determinado para excluir podcasts, sets de DJ o errores de metadatos, centrándonos exclusivamente en el formato de "canción estándar".
- **Filtro de Tempo (Outliers Biológicos)**: Establecimos límites de tempo entre 40 y 250 BPM. El límite inferior (40 BPM) elimina ruidos o grabaciones

extremadamente lentas, mientras que el límite superior (250 BPM) excluye valores raros que distorsionarían la media y la desviación estándar durante la estandarización. Estos filtros son una aplicación directa de nuestro EDA para asegurar la validez física de las canciones.

Ingeniería de Datos y Pipeline de Ingesta (Batch)

Nuestra implementación de Pipeline.py define el flujo de ingestión por lotes (Batch) y la preparación inicial de los datos.

Configuración de Spark y el Pipeline de Limpieza

- **Inicialización:** La función `iniciar_spark()` configura el entorno de ejecución, creando una `SparkSession`. El registro de logs (`logging.info`) confirma la versión del clúster virtual, cumpliendo con las buenas prácticas de MLOps para el seguimiento del entorno de runtime.
- **Flujo PySpark:** La función `limpieza_datos()` es totalmente agnóstica al tamaño del dataset. Realizamos la carga (`spark.read.csv`) y todas las operaciones de calidad (filtros y `na.drop()`) usando `DataFrames` de Spark.
- **Limitación de la Muestra:** Para la fase de validación rápida y pruebas de concepto, aplicamos una limitación de muestra con `df_spotify.limit(1000)` antes de la conversión a Pandas. Esto permite que el ciclo de Feature Engineering y Entrenamiento tarde menos de 30 minutos, cumpliendo el requisito de tiempo de respuesta para el modelo base.

Transición y Almacenamiento Intermedio

Hemos realizado una transición consciente de Spark a Pandas para la fase de división del dataset:

- **`df_spotify.toPandas()`:** Esta operación, aunque costosa en producción a gran escala, se justifica en la Fase 2 para facilitar la integración con las librerías nativas de Scikit-learn (como `train_test_split` y `StandardScaler`). En la Fase 3, se migrará la división y el escalado a Spark MLlib para mantener el procesamiento distribuido de principio a fin.

- **División y Persistencia:** La función dividir_dataset() realiza la división 80/20, identificando popularity como nuestro target potencial. Los conjuntos (X_train, X_test, y_train, y_test) son serializados inmediatamente en el formato Parquet dentro de la carpeta datasets/. Elegimos Parquet por su formato columnar, que optimiza la I/O (entrada/salida de disco) y la eficiencia de lectura en futuras fases de entrenamiento.

Ingeniería de Características y Linaje.

La ingeniería de características es donde transformamos los datos limpios en un espacio vectorial de similitud. Esto se ejecuta desde el script Entrenamiento.py.

Selección y Estandarización de Features

- **Selección Explícita:** En la función Cargar_Datos_Ruta(), seleccionamos y aislamos un conjunto reducido de 5 features: instrumentalness, speechiness, danceability, valence, y tempo. Nuestra justificación es que estas cinco variables definen el "sonido" y el "estado de ánimo" de una canción, siendo el conjunto más robusto para la similitud.

Estandarización Obligatoria: Aplicamos StandardScaler de Scikit-learn, que transforma cada feature para que tenga una media $\mu=0$ y una desviación estándar $\sigma=1$.

$$x_{std} = \frac{x - \mu}{\sigma}$$

Este paso es crucial para K-Means. Sin él, la variable tempo (cuyos valores numéricos son mucho mayores, ej. 120 BPM) dominaría el cálculo de la Distancia Euclídea sobre variables escaladas entre 0 y 1 (como valence o danceability), distorsionando la formación de los clusters.

Linaje y Estrategia de Particionado.

- **Linaje del Dato (Trazabilidad):** Nuestro pipeline establece un linaje claro: CSV Fuente → DF Limpio (Spark) → Archivo Parquet Escalado ($X_train.parquet$) → Modelo K-Means. Cualquier modelo entrenado en el futuro puede ser reconstruido a partir del archivo Parquet versionado, garantizando la trazabilidad.
- **Diseño de Particionado (Data Lake):** Aunque no está implementado en esta fase de prueba, hemos diseñado la capa de Features del Data Lake para ser particionada por genero_principal. Esta partición tiene tres beneficios estratégicos:
 - **Optimización de Lectura:** Reduce el volumen de I/O al entrenar o consultar solo un subconjunto del catálogo (ej. solo canciones de 'Rock').
 - **Auditoría de Sesgos (R2):** Nos permite evaluar la calidad de los clusters (Silhouette Score) por género, facilitando la identificación de disparidades de rendimiento que podrían indicar un sesgo algorítmico.
 - **Entrenamiento Segmentado:** En futuras iteraciones, podremos entrenar modelos K-Means específicos para cada género, mejorando la precisión.

Modelo base Reproducible y Evaluación

El script Entrenamiento.py no solo entrena el modelo, sino que establece el protocolo de reproducibilidad y evaluación inicial.

Entrenamiento y Garantía de Reproducibilidad

- **Modelo K-Means:** Instanciamos el modelo con `n_clusters=3`. Este número inicial es una prueba de concepto para asegurar que el entrenamiento funciona; en la Fase 3 utilizaremos el Método del Codo (Elbow Method) y el Silhouette Score para determinar el número óptimo de clusters k .
- **Semillas Fijas:** La directriz de reproducibilidad se cumple mediante `random_state=42` y `n_init=10`. Al fijar la semilla, garantizamos que la inicialización de los centroides de K-Means sea idéntica en cada ejecución,

haciendo que el resultado final del clustering sea totalmente predecible y trazable.

- **Persistencia:** El modelo entrenado se serializa con joblib.dump en el directorio modelos/ como un archivo.pkl (kmeans\spotify_model.pkl), lo que facilita la carga rápida y la verificación del modelo.

Evaluación y Métricas Mínimas.

Como nuestro modelo base es no supervisado, nos basamos en métricas internas de calidad del clustering:

Métrica	Definición y Cálculo (Implementación en Entrenamiento.py)	Importancia
Silhouette Score	Se calcula con silhouette_score(X_train, kmeans.labels_). Mide cuán cohesivo y separado está el cluster. Valores cercanos a +1 indican clusters densos y bien separados.	Es la métrica principal loggeada en el script para la comparación inicial.
WCSS (Inercia)	Se extrae del atributo kmeans.inertia_. Es la suma de las distancias cuadradas de cada punto a su centroide asignado. Un valor bajo indica clusters compactos (alta cohesión).	Utilizado para el Método del Codo para encontrar el k óptimo. Nuestro log registra este valor para la trazabilidad.

Validación Adecuada (k-fold Estratificado – Plan)

Para la validación definitiva del baseline, nuestra planificación de MLOps exige:

- **k-fold Estratificado:** Usaremos una validación cruzada ($k = 5$) estratificada por genero_principal. Esto es vital para evitar el sesgo en la validación, asegurando que cada fold tenga una mezcla representativa del catálogo y que el modelo no se especialice en un solo género.

Gestión de Artefactos (MLOps) y Pruebas

El cumplimiento de los estándares MLOps es un pilar fundamental de nuestra arquitectura de Spark Moderno.

Plan de Adopción del MLflow (Gestión de Artefactos)

Estamos en el proceso de migrar el logging simple de Entrenamiento.py a MLflow para la gestión de artefactos y el versionado. Esto garantiza la trazabilidad de los "3 Cs" del modelo:

- **Código:** El script Entrenamiento.py se registrará.
- **Configuración:** Los Parámetros (mlflow.log_param()): Registraremos k ($n_clusters=3$), la semilla ($random_state=42$) y la versión de las librerías (sklearn, pandas).
- **Contenido (Modelo y Métricas):** Registraremos las Métricas (mlflow.log_metric()): Silhouette Score, WCSS) y el Modelo (mlflow.sklearn.log_model()) en el MLflow Model Registry con una firma clara.

Framework de Pruebas (Pytest y Data Tests)

Hemos establecido un marco de pruebas riguroso utilizando Pytest para asegurar la calidad del código (unit tests) y la integridad de los datos (data tests), cumpliendo el requisito de ≥ 8 unit tests.

- **Pruebas Unitarias (≥ 8):**
 - **Test de Inicialización:** Comprobar que iniciar_spark() no solo devuelva una sesión, sino que registre el log de éxito.
 - **Test de Calidad de Datos:** Validar la aplicación de los filtros de tempo (40-250 BPM) y duration_ms (máx. 20 min).
 - **Test de Escalado:** Verificar que el StandardScaler en Entrenamiento.py produzca un array con $\mu \approx 0$ y $\sigma \approx 1$.
 - **Test de Reproducibilidad:** Asegurar que dos llamadas a dividir_dataset() con la misma entrada produzcan exactamente los mismos archivos Parquet.
 - **Test de Persistencia:** Verificar que el archivo kmeans_spotify_model.pkl se genere correctamente y pueda ser cargado sin errores con joblib.load.
- **Tests de Datos:** Implementaremos verificaciones en el archivo Parquet intermedio (X\train.parquet):
 - **Test de No-Nulidad:** Confirmar que no existen valores nulos en ninguna de las 5 features después de la limpieza.
 - **Test de Esquema:** Garantizar que la estructura de las columnas se mantenga idéntica y que los tipos de datos sean float64 para todas las features numéricas.
- **Smoke Test del Pipeline:** La ejecución secuencial de Pipeline.py y Entrenamiento.py es nuestro smoke test. La prueba pasa si el flujo completo se ejecuta sin tracebacks y genera el artefacto final (kmeans\spotify\model.pkl), lo que nos confirma que la tubería de datos es robusta y ejecutable con un solo comando.

Diagrama de Flujo del Pipeline Implementado.

El diagrama a continuación resume la arquitectura y la secuencia de ejecución de nuestros scripts en la Fase 2, sirviendo como mapa de productivización de nuestra arquitectura Spark.

Etapa	Módulo	Tecnología	Justificación
		Primaria	
Ingesta y limpieza	Pipeline.py (limpieza_datos)	PySpark	Uso de la capacidad distribuida de Spark para aplicar filtros de calidad sobre grandes volúmenes de datos.
Almacenamiento intermedio	Pipeline.py (dividir_dataset)	Pandas → Parquet	Transición a Pandas para la división simple con Sklearn. Persistencia en formato columnar para eficiencia de lectura.
Feature Engineering	Entrenamiento.py (Cargar_Datos_Ruta)	Scikit-learn (StandardScaler)	Estandarización de las 5 features clave para evitar el dominio de variables en el cálculo de distancias.

Entrenamiento Base	Entrenamiento.py (entrenamiento)	Scikit-learn (KMeans)	Implementación del baseline heurístico K-Means con semillas fijas para garantizar la reproducibilidad.
Gestión de Artefactos	Entrenamiento.py	Joblib (Pendiente de migración a MLflow)	Serialización del modelo entrenado y registro inicial de métricas internas (Silhouette, WCSS).

Este flujo de trabajo es el que garantiza la eficiencia, la reproducibilidad y el cumplimiento de los estándares de calidad para el avance del proyecto.