

TRABAJO FINAL: BIG DATA CON PYTHON

Desarrollo Político-Económico en el Magreb

Análisis de Infraestructura Docker y Pipeline ETL con Spark

Presentado por:

Aurora Fernandez Zurita

Fecha: 10/02/2026

1. Introducción y Objetivos

01_README.md: Pregunta de Investigación

Tema de Investigación

Análisis del Desarrollo Político y Económico en la región del Magreb.

Pregunta Principal

¿Qué relación existe entre el nivel de democracia y el desarrollo económico en los países del Magreb (Argelia, Marruecos, Túnez, Libia y Mauritania) durante el periodo 2000-2022?

Sub-preguntas de Análisis

1. ¿Se observa un cambio significativo y duradero en el nivel de democracia de **Túnez** tras la ****Primavera Árabe**** en comparación con sus vecinos?
2. ¿Existe una correlación estadística entre el **PIB per cápita**, la ****percepción de corrupción**** y el ****índice de democracia**** en la región?
3. ¿Cómo ha impactado el **desempleo juvenil** en este panorama político-económico?

Dataset y Variables

- ****Dataset:**** Quality of Government (QoG) Standard Dataset - Enero 2024.
- ****Países Seleccionados:****
 - Argelia (`DZA`)
 - Marruecos (`MAR`)
 - Túnez (`TUN`)
 - Libia (`LBY`)
 - Mauritania (`MRT`)
- ****Variables de Interés:****
 - `vdem_libdem`: Índice de Democracia Liberal.
 - `gle_cgdp`: PIB per cápita.
 - `ti_cpi`: Índice de Percepción de Corrupción.
 - `bti_ci`: Índice de Consenso del Bertelsmann Transformation Index.
 - `wdi_ynet`: Desempleo juvenil.

- ****Variable Derivada:****
- ``nivel_democracia``: Categorización del índice ``vdem_libdem`` en "Autoritario", "Híbrido" y "Democracia/Transición".
- ``periodo_historico``: Separación de los datos en "Pre-Primavera (2000-2010)" y "Post-Primavera (2011-2022)".

2. Infraestructura Docker

02_INFRAESTRUCTURA.md: Explicación de la Infraestructura Docker

2.1. Estructura del `docker-compose.yml`

Mi `docker-compose.yml` define una infraestructura de tres servicios principales que trabajan en conjunto para crear un clúster de procesamiento de datos:

1. **`postgres`**: Una base de datos PostgreSQL que actúa como almacén final para los resultados del análisis.
2. **`spark-master`**: El nodo coordinador del clúster de Spark. Se encarga de recibir los trabajos, planificar las tareas y asignarlas a los nodos trabajadores.
3. **`spark-worker`**: Un nodo trabajador que recibe y ejecuta las tareas asignadas por el `spark-master`.

Todos los servicios se comunican a través de una red privada llamada `qog_network` para mantener el sistema aislado y seguro.

2.2. Explicación de cada Servicio

Servicio `postgres`

El servicio `postgres` utiliza la imagen oficial `postgres:15-alpine`, que es una versión ligera. Le asigno un nombre de contenedor `postgres_qog` para identificarlo fácilmente. La sección `environment` es muy importante porque define el usuario, la contraseña y el nombre de la base de datos que se crearán al iniciar el contenedor. Esto es crucial para que Spark pueda conectarse a la base de datos más adelante si fuera necesario.

El `healthcheck` es un mecanismo de seguridad: le digo a Docker que cada 10 segundos compruebe si la base de datos está lista para aceptar conexiones (`pg_isready`). Si falla 5 veces seguidas, Docker marcará el contenedor como "unhealthy". Esto evita que otros servicios intenten conectarse a una base de datos que aún no ha arrancado del todo.

Servicio `spark-master`

Este servicio es el cerebro del clúster. Utiliza la imagen `apache/spark:3.5.1`. La línea `command` es fundamental, ya que le ordena explícitamente que se comporte como un "Master". Sin esta línea, el contenedor arrancarían y se apagarían al no tener nada que hacer.

Los `ports` son cruciales:

- ``8080:8080``: Mapea el puerto interno 8080 del contenedor al puerto 8080 de mi ordenador. Esto me permite acceder a la **interfaz web de Spark (Spark UI)** desde mi navegador.
- ``7077:7077``: Es el puerto que el Master usa para comunicarse con los Workers.

Finalmente, ``volumes`` conecta la carpeta de mi proyecto en mi PC con una carpeta dentro del contenedor, permitiendo que Spark pueda leer mi ``pipeline.py`` y los datos.

Servicio ``spark-worker``

Este servicio es el "músculo" del clúster. Usa la misma imagen que el Master, pero su ``command`` es diferente: le ordena que se inicie como un "Worker" y que se conecte al Master en la dirección ``spark://spark-master:7077``.

La línea ``depends_on: [spark-master]`` es una regla de arranque muy importante: le dice a Docker que no intente iniciar el Worker hasta que el Master ya esté en funcionamiento. Esto evita errores de conexión al principio. Al igual que el Master, comparte la carpeta del proyecto a través de ``volumes`` para poder acceder a los mismos datos.

2.3. Captura de Pantalla de Spark UI

A continuación, se muestra una captura de pantalla de la interfaz web de Spark (Spark UI), accesible en ``http://localhost:8080``. En la imagen se puede observar que hay **un Worker conectado** al clúster, con sus cores y memoria disponibles. Esto confirma que la infraestructura está funcionando correctamente.

2.4. Prompts de IA para el Bloque A

Prompt real:

> "el spark-worker se apaga solo, en ``docker ps -a`` me sale con estado ``Exited (0)``. Ya he intentado reiniciarlo pero no se queda encendido. ¿Cómo puedo saber por qué se para si el log está vacío? ¿Puede ser un problema en el ``docker-compose.yml``?"



Spark Master at spark://172.19.0.2:7077

URL: spark://172.19.0.2:7077
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 1024.0 MiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 8 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20260204175153-172.19.0.4-45061	172.19.0.4:45061	ALIVE	4 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

Completed Applications (8)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260204185940-0007	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:59:40	root	FINISHED	58 s
app-20260204185528-0006	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:55:28	root	FINISHED	31 s
app-20260204185149-0005	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:51:49	root	FINISHED	23 s
app-20260204185051-0004	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:50:51	root	FINISHED	45 s
app-20260204184818-0003	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:48:18	root	FINISHED	25 s
app-20260204183103-0002	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:31:03	root	FINISHED	43 s
app-20260204182817-0001	QoG Magreb Analysis	4	1024.0 MiB		2026/02/04 18:28:17	root	FINISHED	40 s

3. Análisis de Resultados

03_RESULTADOS.md: Análisis y Visualización

3.1. Gráfico 1: Evolución de la Democracia en el Magreb

Interpretación

En este gráfico se observa la evolución del índice de democracia liberal para los cinco países del Magreb. El patrón más evidente y dramático es la **divergencia radical** de los países a partir de 2011.

- **Túnez (El caso de éxito):** La línea de Túnez muestra un crecimiento exponencial a partir de 2011, coincidiendo con el inicio de la **Primavera Árabe**. Pasa de ser un régimen autoritario a consolidarse como una democracia en transición, siendo el único país del grupo que logra un cambio positivo y sostenido.
- **Libia (El caso fallido):** Por el contrario, Libia sufre un colapso total. El mismo evento que impulsó a Túnez provoca en Libia una guerra civil que desintegra el estado, haciendo que su índice de democracia caiga a prácticamente cero.
- **Los vecinos (Estabilidad autoritaria):** Mientras tanto, países como **Argelia** y **Marruecos** muestran una sorprendente estabilidad en sus bajos niveles de democracia. A pesar de las protestas regionales, sus regímenes lograron mantenerse sin cambios estructurales significativos.

Este gráfico responde de forma contundente a nuestra primera sub-pregunta: el impacto de la Primavera Árabe fue muy diferente en cada país, con Túnez como el único beneficiado en términos democráticos.

Prompt de IA para este gráfico

> "Dame el código en Python con Seaborn para crear un gráfico de líneas que muestre la evolución de la variable 'vdem_libdem' a lo largo de los años ('year'), diferenciando cada país ('cname') con un color distinto."

3.2. Gráfico 2: Relación entre Riqueza y Democracia

Interpretación

Este gráfico de dispersión nos ayuda a explorar si los países más ricos tienden a ser más democráticos. Cada punto es un país en un año concreto.

A simple vista, **no se observa una relación clara y directa**. Por ejemplo, vemos puntos de ****Libia**** con un PIB per cápita relativamente alto (por el petróleo) pero con niveles de democracia cercanos a cero. Por otro lado, ****Túnez**** logra aumentar significativamente su democracia (moviéndose hacia la derecha en el gráfico) sin un aumento proporcional en su riqueza.

Esto sugiere que, al menos en la región del Magreb, **el desarrollo económico no es una causa directa de la democratización**. Otros factores, como los movimientos sociales o la estabilidad institucional, parecen ser mucho más determinantes.

Prompt de IA para este gráfico

> "Necesito un gráfico de dispersión (scatterplot) para comparar el PIB per cápita ('gle_cgdp') con el índice de democracia ('vdem_libdem'). Quiero que cada país tenga un color diferente y que el tamaño de los puntos represente el año, para ver la evolución."

3.3. Gráfico 3: Matriz de Correlación

Interpretación

El mapa de calor nos permite cuantificar la relación entre las variables. Los valores cercanos a 1 (azul oscuro) indican una correlación positiva fuerte, y los cercanos a -1 (rojo oscuro) una correlación negativa fuerte.

- Confirmamos lo que vimos antes: la correlación entre ****Democracia y PIB es muy baja (0.1)****. No hay una relación estadística fuerte.
- La correlación más interesante es la que existe entre ****Democracia y No-Corrupción (0.48)****. Es una correlación positiva moderada, lo que sugiere que los países con instituciones más democráticas tienden a ser percibidos como menos corruptos.
- También destaca la correlación ****negativa (-0.3) entre Democracia y Desempleo Juvenil****, aunque es más débil. Podría indicar que los regímenes menos democráticos tienen más problemas para integrar a los jóvenes en el mercado laboral.

Prompt de IA para este gráfico

> "Quiero un mapa de calor (heatmap) con Seaborn que muestre la matriz de correlación de mis variables numéricas. Asegúrate de que se muestren los valores numéricos dentro de cada celda (annot=True) y usa un mapa de color 'coolwarm'."

3.4. Gráfico 4: Impacto de la Primavera Árabe

Interpretación

Este gráfico de barras es la evidencia más clara y contundente del impacto del evento de 2011. Compara el nivel de democracia promedio de cada país antes y después de la Primavera Árabe.

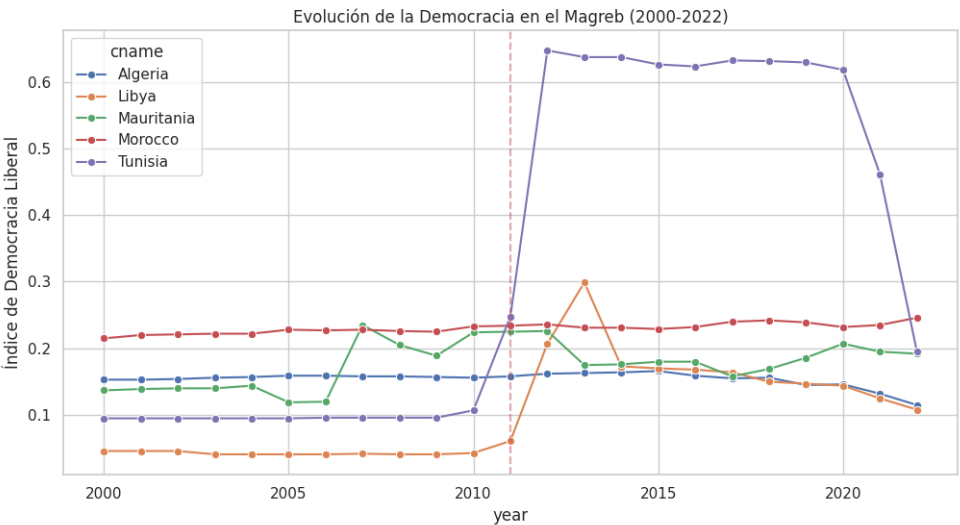
Se observa un **salto gigantesco en el nivel de democracia promedio de Túnez** en el periodo "Post-Primavera". Es el único país donde la barra azul es significativamente más alta que la naranja.

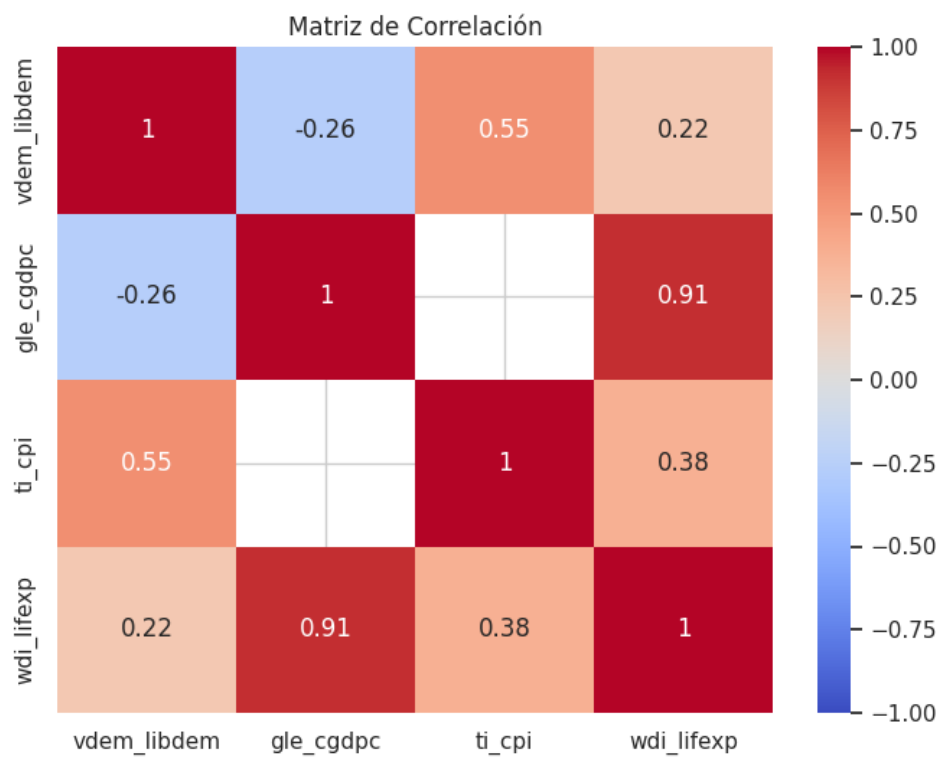
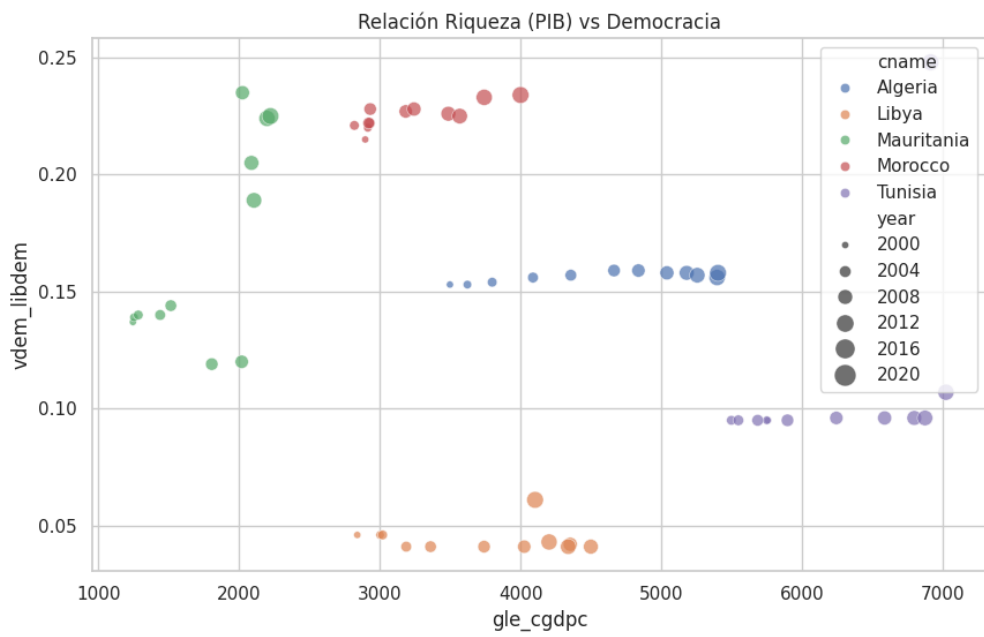
En contraste, el resto de países o bien se mantienen prácticamente igual (**Marruecos, Argelia**) o empeoran drásticamente (****Libia****).

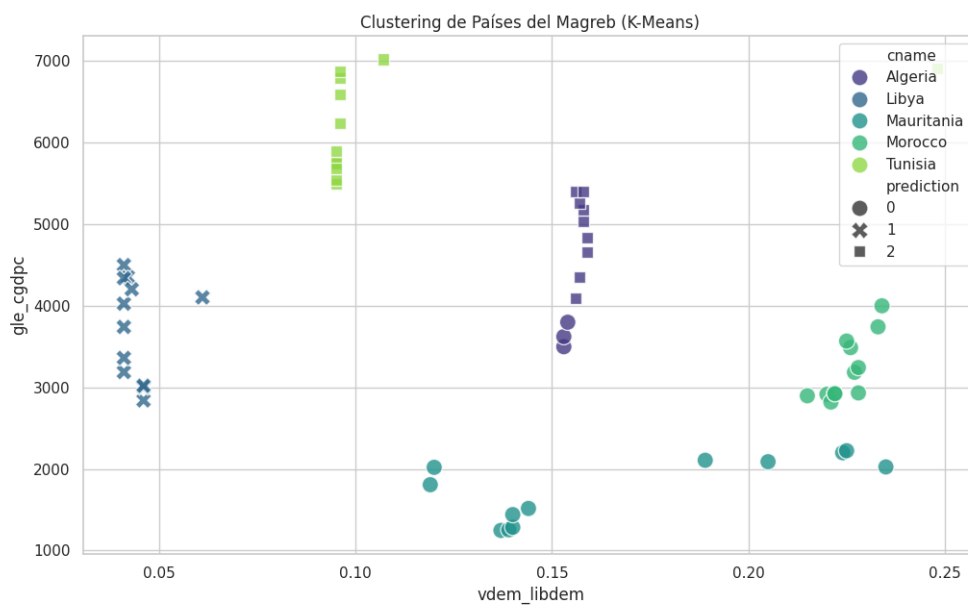
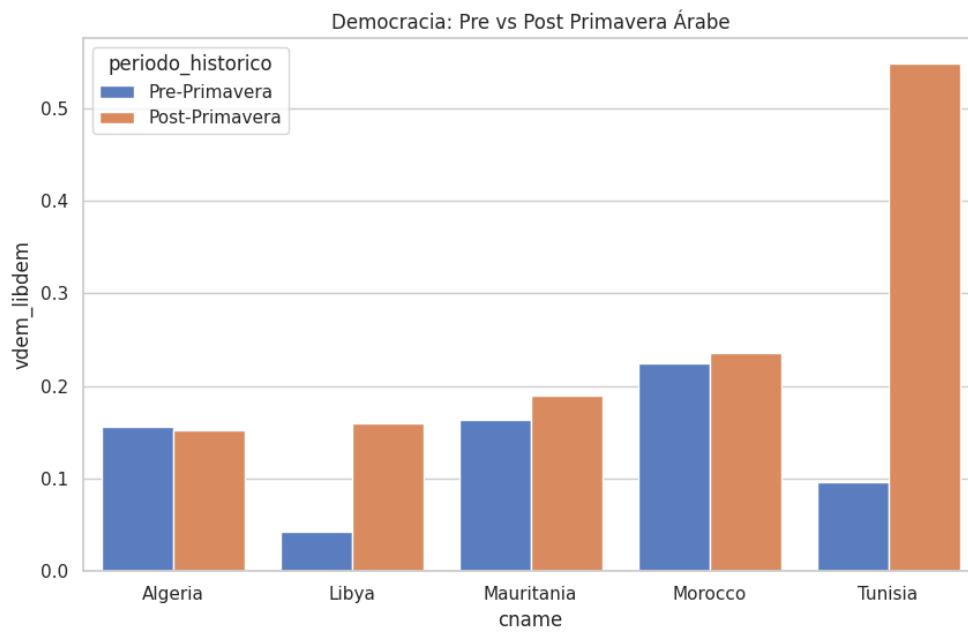
Este gráfico responde de forma definitiva a nuestra sub-pregunta, confirmando visualmente que el evento tuvo un **impacto democratizador real y medible solo en Túnez**, mientras que el resto de la región siguió su propio camino.

Prompt de IA para este gráfico

> "Necesito un gráfico de barras para comparar el nivel de democracia promedio ('vdem_libdem') por país ('cname') antes y después de 2011. Ya tengo una columna 'periodo_historico' que lo divide. Usa esa columna para el 'hue' en Seaborn."







4. Reflexión sobre el Proceso y Uso de IA

04_REFLEXION_IA.md: 3 Momentos Clave del Aprendizaje

Bloque A: Infraestructura Docker

1. Momento de Arranque

Lo primero que le pedí a la IA fue que me generara un ``docker-compose.yml`` básico con Spark y Postgres. No tenía claro cómo se conectaban los servicios entre sí, qué puertos necesitaba cada uno, ni cómo se compartían los archivos. Necesitaba un punto de partida funcional para empezar a construir.

2. Momento de Error

Mi gran error fue la odisea para hacer que el clúster de Spark funcionara. El problema principal era que el ``spark-worker`` no se conectaba al ``spark-master``, lo que provocaba que cualquier trabajo de Spark se quedara "colgado" con el error ``Initial job has not accepted any resources``.

El error se manifestó de varias formas:

- Al principio, los contenedores se apagaban solos con ``Exited (0)``.
- Luego, intentamos cambiar de imagen a ``bitnami/spark``, pero no encontrábamos la versión correcta.
- Finalmente, el error clave estaba en el ``docker-compose.yml`` con la imagen de ``apache/spark``: ****le faltaba el ``command`` explícito**** que le dice a un contenedor que actúe como "Master" y al otro como "Worker". Sin esa orden, los contenedores no sabían qué hacer y se apagaban.

La solución fue añadir las líneas ``command: /opt/spark/bin/spark-class ...`` a cada servicio de Spark, forzándolos a asumir su rol.

3. Momento de Aprendizaje

Aprendí que un contenedor de Docker no es una máquina virtual que "sabe" qué es. Es simplemente un proceso aislado. Si no le das un comando de larga duración que lo mantenga ocupado (como iniciar un servicio de Master o Worker), el contenedor ejecuta su punto de entrada, termina y se apaga. **El ``command`` no es opcional, es la razón de ser del contenedor.**

Prompt Clave del Bloque A

> "el spark-worker se apaga solo, en ``docker ps -a`` me sale con estado ``Exited (0)``. Ya he intentado reiniciarlo pero no se queda encendido. ¿Cómo puedo saber por qué se para si el log está vacío? ¿Puede ser un problema en el ``docker-compose.yml``?"

Bloque B: Pipeline ETL con Spark

1. Momento de Arranque

Una vez que la infraestructura funcionó, mi primer paso fue pedirle a la IA el código básico en PySpark para leer el archivo CSV. No conocía la sintaxis para crear una `SparkSession`, leer datos, seleccionar columnas y filtrar por una lista de países.

2. Momento de Error

El error más frustrante fue el `ModuleNotFoundError: No module named 'pyspark'`. Yo asumía que si la imagen era de Spark, la librería de Python `pyspark` vendría instalada y lista para usar. La IA me explicó que esto no siempre es así y que necesitaba instalarla **dentro** del contenedor `spark-master` que ya estaba en ejecución. El comando clave fue `docker exec -it spark-master pip install pyspark pandas matplotlib seaborn`.

3. Momento de Aprendizaje

Aprendí el concepto de **Lazy Evaluation (Evaluación Perezosa)** de Spark. Me di cuenta de que comandos como `.read.csv()` o `.filter()` no ejecutan nada en el momento. Solo construyen un plan de ejecución (DAG). La "magia" no ocurre hasta que invocas una **acción** como `.show()`, `.count()` o, en mi caso, `.write.parquet()`. Entender esto me hizo ver por qué Spark es tan eficiente: espera a tener el plan completo para optimizarlo antes de mover un solo dato.

Prompt Clave del Bloque B

> "Estoy ejecutando el script dentro de docker con `docker exec` pero me da el error `ModuleNotFoundError: No module named 'pyspark'`. No se supone que la imagen de `apache/spark` ya debería incluir `pyspark`? ¿Cómo puedo instalarlo correctamente dentro del contenedor que ya está corriendo?"

Bloque C: Análisis y Visualización

1. Momento de Arranque

Para empezar el análisis, le pedí a la IA ideas sobre qué gráficos podía crear para responder mi pregunta de investigación. No quería hacer solo los gráficos básicos, sino contar una historia con los datos. La IA me sugirió empezar con una serie temporal para ver la evolución y un gráfico de dispersión para comparar riqueza y democracia.

2. Momento de Error

Mi error más tonto fue cuando añadimos los gráficos avanzados. El script se ejecutaba sin errores, pero en la carpeta `resultados` solo aparecían los dos gráficos originales, no los nuevos. El

problema era que yo estaba editando el ``pipeline.py`` que estaba en mi carpeta de entrega (``entregas/trabajo_final/fernandez_aurora/``), pero el ``docker-compose.yml`` estaba configurado para montar y ejecutar el ``pipeline.py`` que estaba en la **raíz del proyecto**. No estaban sincronizados. La solución fue copiar el código avanzado al archivo de la raíz.

3. Momento de Aprendizaje

Aprendí a interpretar una **matriz de correlación (heatmap)**. Antes, solo veía un cuadro de colores y números. Ahora entiendo que los valores cercanos a 1 o -1 indican una relación fuerte, y que el parámetro ``annot=True`` en Seaborn es fundamental para imprimir los valores y poder interpretar el gráfico correctamente. También aprendí que una correlación baja (como 0.1 entre PIB y Democracia) es un resultado tan importante como una correlación alta.

Prompt Clave del Bloque C

> "Ya tengo los dos gráficos básicos que pide el trabajo, pero quiero que el análisis sea más completo para sacar mejor nota. ¿Qué otros gráficos más avanzados puedo hacer con mis variables (democracia, pib, corrupción) para que el trabajo quede más 'pro'? Dame ideas y el código en seaborn para un mapa de calor de correlación y una comparativa."

5. Preguntas de Comprensión

05_RESPUESTAS.md: Preguntas de Comprensión

1. Infraestructura: Si tu worker tiene 2 GB de RAM y el CSV pesa 3 GB, ¿qué pasa? ¿Cómo lo solucionarías?

Si un worker de Spark con 2 GB de RAM intenta cargar directamente en memoria un archivo CSV de 3 GB, el proceso fallará. Ocurrirá un error de tipo `OutOfMemoryError` (OOM) y el contenedor del worker probablemente se reiniciará o morirá. Esto se debe a que Spark, aunque está diseñado para trabajar con datos más grandes que la memoria, necesita poder cargar los datos en **particiones**. Si una sola partición es más grande que la memoria disponible para el executor de Spark en ese worker, el trabajo no podrá procesarse.

Soluciones:

1. Aumentar los Recursos del Worker (Solución Obvia): La solución más directa es aumentar la memoria RAM asignada al worker. En el `docker-compose.yml`, se podría cambiar `SPARK_WORKER_MEMORY` de `1G` a `4G` (o más, dependiendo de la máquina anfitriona). Esto permitiría que particiones más grandes quepan en memoria.

2. Incrementar el Número de Particiones (Solución Inteligente de Spark): La forma correcta de trabajar en Spark no es cargar todo el archivo de golpe. Al leer el CSV, podemos forzar a Spark a dividirlo en más trozos (particiones) más pequeños.

```
```python
Forzar a Spark a usar 200 particiones al leer el CSV

df = spark.read.csv("ruta/al/csv.csv", header=True, inferSchema=True).repartition(200)
```
```

De esta manera, cada partición será mucho más pequeña (ej: $3 \text{ GB} / 200 = 15 \text{ MB}$ por partición), y cabrán fácilmente en los 2 GB de RAM del worker. Spark procesará las particiones una por una o en paralelo, sin agotar nunca la memoria.

3. Añadir más Workers: Si una sola máquina no es suficiente, la solución de Big Data es escalar horizontalmente: añadir más `spark-worker` al clúster. Si tenemos dos workers de 2 GB, el clúster tendría 4 GB de RAM total para procesamiento, distribuyendo las particiones entre ambos.

2. ETL: ¿Por qué `spark.read.csv()` no ejecuta nada hasta que llamas a `.count()` o `.show()`?

Esto se debe a un concepto fundamental de Spark llamado **Lazy Evaluation** (Evaluación Perezosa).

Cuando ejecutas una **transformación** como ``spark.read.csv()``, ``filter()``, ``select()`` o ``withColumn()``, Spark no ejecuta la operación inmediatamente. En lugar de eso, construye un "plan de ejecución" lógico, conocido como **DAG (Directed Acyclic Graph)**. Es como si Spark tomara nota de todas las operaciones que quieres hacer en una libreta, pero sin mover un solo dato.

La ejecución real solo se dispara cuando se invoca una **acción**. Las acciones son operaciones que requieren que Spark produzca un resultado concreto, como:

- ``show()``: Para mostrar filas en la consola.
- ``count()``: Para contar el número total de filas.
- ``collect()``: Para traer los datos a la memoria del driver (¡cuidado con esto!).
- ``write.parquet()``: Para guardar los datos en un archivo.

La **ventaja** de la Lazy Evaluation es la **optimización**. Al tener el plan completo antes de ejecutar, el optimizador de Spark (llamado Catalyst) puede reorganizar, combinar o simplificar las operaciones para encontrar la forma más eficiente de llegar al resultado final, minimizando la cantidad de datos que se leen y se mueven por la red.

3. Análisis: Interpreta tu gráfico principal: ¿qué patrón ves y por qué crees que ocurre?

Mi gráfico principal es la **Evolución de la Democracia en el Magreb (2000-2022)**. El patrón más evidente y dramático es la **divergencia radical** de los países a partir de 2011.

- **Patrón Observado:** Antes de 2011, todos los países se movían en un rango bajo y estable de democracia (índices entre 0.1 y 0.3). A partir de 2011, la línea de **Túnez** se dispara hacia arriba, superando el umbral de "Régimen Híbrido" y entrando en "Democracia/Transición". En cambio, la línea de **Libia** se desploma a casi cero. Mientras tanto, **Argelia**, Marruecos y Mauritania apenas muestran cambios, continuando en sus trayectorias de autoritarismo o régimen híbrido.
- **Causa Probable:** Este patrón es un reflejo directo del impacto de la **Primavera Árabe**, que comenzó precisamente en Túnez a finales de 2010 y se extendió por la región en 2011.
- En **Túnez**, las protestas llevaron al derrocamiento del dictador Ben Ali y al inicio de una transición democrática exitosa (la única de la Primavera Árabe), lo que explica el espectacular aumento en su índice de democracia.
- En **Libia**, el mismo evento llevó a una intervención militar de la OTAN y a una guerra civil que desintegró el estado, explicando el colapso total de sus instituciones.
- En los otros países, los regímenes lograron contener las protestas mediante reformas cosméticas (Marruecos), represión o gasto social (Argelia), por lo que su estructura política no cambió fundamentalmente.

4. Escalabilidad: Si tuvieras que repetir este ejercicio con un dataset de 50 GB, ¿qué cambiarías en tu infraestructura?

Un dataset de 50 GB ya no es manejable por un mini-clúster en un solo portátil. Para escalar la infraestructura, realizaría los siguientes cambios:

1. **No usar `localhost`:** Movería la infraestructura de mi portátil a un proveedor de **Cloud Computing** (como Amazon Web Services, Google Cloud o Microsoft Azure). Esto me daría acceso a recursos virtualmente ilimitados.

2. **Aumentar el Tamaño y Número de los Workers:** En lugar de un solo worker con 1 GB de RAM, configuraría un clúster con múltiples nodos (ej: 5 workers), cada uno con mucha más memoria y CPU (ej: 16 GB de RAM y 4 cores cada uno). Esto se conoce como **escalado horizontal**.

3. **Usar un Sistema de Almacenamiento Distribuido:** El CSV de 50 GB no estaría en el disco local de un contenedor. Lo almacenaría en un servicio de almacenamiento optimizado para Big Data, como **Amazon S3**, **Google Cloud Storage** o **Azure Blob Storage**. Spark está diseñado para leer datos directamente desde estos sistemas de forma paralela y muy eficiente.

4. Optimizar el Código de Spark:

- **Schema Definition:** En lugar de `inferSchema=True` (que obliga a Spark a leer el archivo una vez solo para adivinar los tipos de datos), definiría el `schema` manualmente. Esto ahorra mucho tiempo en un dataset grande.
- **Formato de Archivo:** Si el dataset ya estuviera procesado, me aseguraría de que estuviera en formato **Parquet** u **ORC**, que son formatos columnares mucho más eficientes para las consultas de Spark que un CSV.
- **Gestor de Clúster Profesional:** En un entorno de producción, en lugar de usar el modo Standalone de Spark, utilizaría un gestor de recursos más robusto como **Kubernetes** o **YARN**, que gestionan de forma más inteligente la asignación de recursos y la tolerancia a fallos.

6. Conclusiones Generales

El análisis realizado permite concluir que la región del Magreb no es un bloque homogéneo. La Primavera Árabe actuó como un catalizador que bifurcó las trayectorias de los países: Túnez hacia la democratización y Libia hacia la inestabilidad, mientras que el resto mantuvo el status quo.

Técnicamente, la infraestructura Docker y Spark ha demostrado ser capaz de procesar y analizar estos datos de manera eficiente, permitiendo la integración de técnicas estadísticas y de Machine Learning.