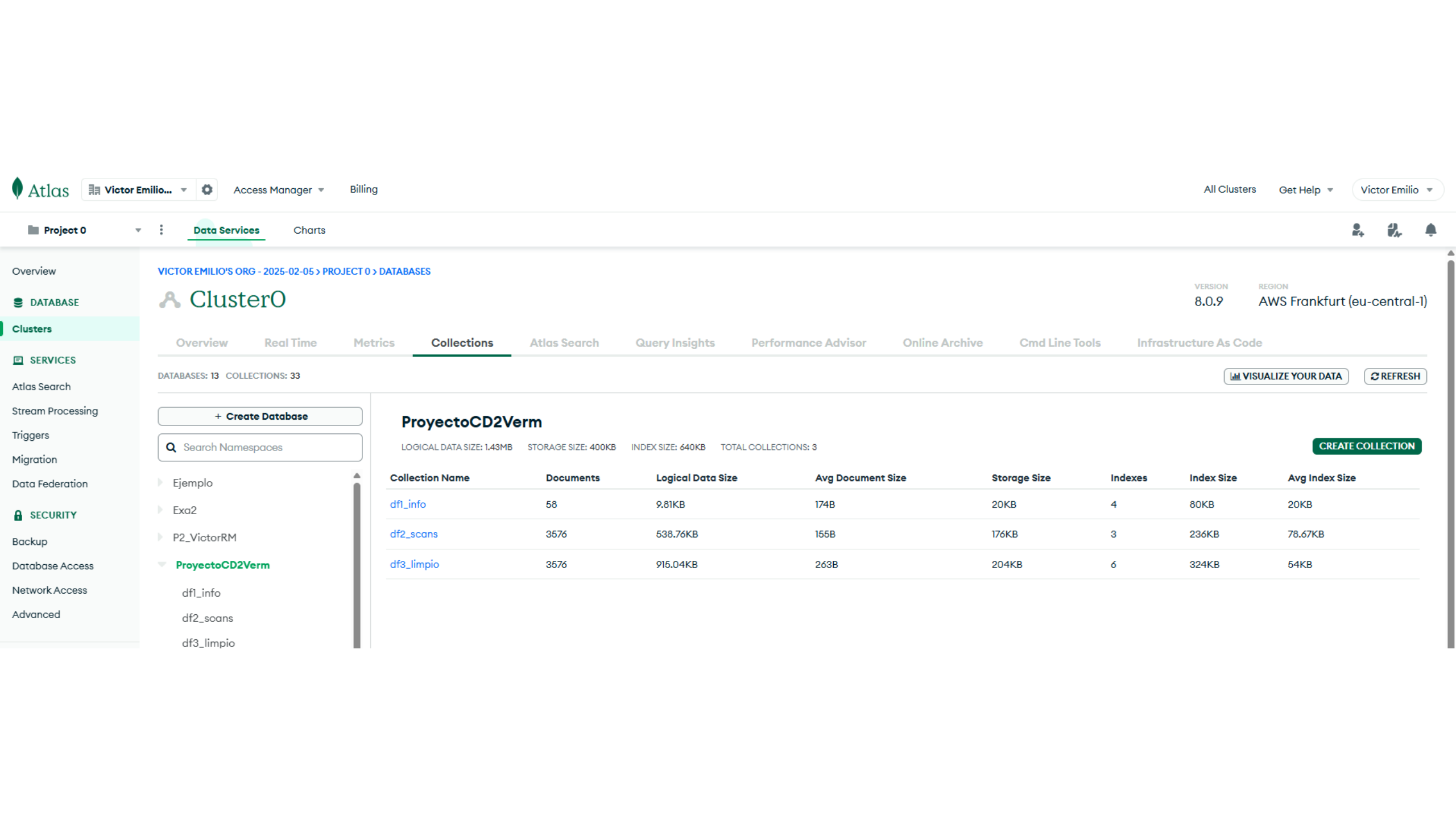

Atlas
Victor Emilio...
Access Manager
Billing
All Clusters
Get Help
Victor Emilio
Project 0
Data Services
Charts
Overview
DATABASE
Clusters
SERVICES
Atlas Search
Stream Processing
Triggers
Migration
Data Federation
SECURITY
Backup
Database Access
Network Access
Advanced
VICTOR EMILIO'S ORG - 2025-02-05 > PROJECT 0 > DATABASES
Cluster0
VERSION
8.0.9
REGION
AWS Frankfurt (eu-central-1)
Overview
Real Time
Metrics
Collections
Atlas Search
Query Insights
Performance Advisor
Online Archive
Cmd Line Tools
Infrastructure As Code
DATABASES: 13 COLLECTIONS: 33
VISUALIZE YOUR DATA
REFRESH
+ Create Database
Search Namespaces
Ejemplo
Exa2
P2_VictorRM
ProyectoCD2Verm
df1_info
df2_scans
df3_limpio
ProyectoCD2Verm
LOGICAL DATA SIZE: 1.43MB
STORAGE SIZE: 400KB
INDEX SIZE: 640KB
TOTAL COLLECTIONS: 3
CREATE COLLECTION
Collection Name
Documents
Logical Data Size
Avg Document Size
Storage Size
Indexes
Index Size
Avg Index Size
df1_info 58 9.81KB 174B 20KB 4 80KB 20KB
df2_scans 3576 538.76KB 155B 176KB 3 236KB 78.67KB
df3_limpio 3576 915.04KB 263B 204KB 6 324KB 54KB

Colab Notebooks - Google Driv

proyecto2cd2.ipynb - Colab

drive.google.com/drive/folders/1zNCigM0QChw3N75Blw3HRvWHUhxXKZCH

Drive

Buscar en Drive

Nuevo

- Página principal
- Mi unidad
- Ordenadores
- Compartido conmigo
- Reciente
- Destacados
- Spam
- Papelera
- Almacenamiento

9,08 GB de 15 GB usado

Obtener más almacenamiento

Mi unidad > Colab Notebooks

Tipo | Personas | Modificado | Fuente

Nombre	Propietario	Última modificació
data	yo	15:10 yo
Copia de Te damos la bienvenida a Colaboratory	yo	15:08 yo
Copia de Te damos la bienvenida a Colaboratory	yo	15:09 yo
proyecto2cd2.ipynb	yo	16:33 yo

tecnico.verm@gmail.com

¡Hola, Victor!

Gestionar tu cuenta de Google

Añadir cuenta

Cerrar sesión

60 % de 15 GB en uso

Política de Privacidad • Términos del Servicio

▶️ Cómo ejecutar este notebook en Google Colab

Para poder ejecutar este notebook en Google Colab correctamente, sigue estos pasos iniciales para preparar el entorno:

🗂️ 1. Montar Google Drive

Esto permite acceder a los archivos JSON desde tu unidad de Google Drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

📦 2. Instalar librerías necesarias

Colab no incluye por defecto algunas de las bibliotecas usadas. Ejecuta esto al inicio:

```
!pip install pymongo dnspython pandas matplotlib seaborn
```

📁 3. Configurar la ruta de acceso a los archivos

Asegúrate de que la carpeta `data` con los archivos `.json` esté ubicada dentro de:

```
/content/drive/MyDrive/Colab Notebooks/data
```

Y define la ruta en una variable para usarla en el notebook:

```
carpeta_data = "/content/drive/MyDrive/Colab Notebooks/data"
```

20 s
```
[3] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

proyecto2cd2.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Compartir Gemini

Comandos + Código + Texto

RAM Disco

20 s [3]
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

6 s [4]
```
!pip install pymongo dnspython pandas matplotlib seaborn
```

```
Collecting pymongo
  Downloading pymongo-4.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
Collecting dnspython
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading pymongo-4.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 39.3 MB/s eta 0:00:00
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 21.5 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.13.0
```

1. Introducción del Proyecto

---

jbjbj Este proyecto se desarrolla como parte de la asignatura **Ingeniería y Ciencia de Datos II** del **Máster Universitario en Ingeniería Informática** de la **Universidad de Málaga**.

Su objetivo principal es aplicar de forma práctica los conocimientos adquiridos durante la asignatura mediante el análisis de un conjunto de

Variables Terminal

16:32 Python 3

1. Introducción del Proyecto

jbjbj Este proyecto se desarrolla como parte de la asignatura **Ingeniería y Ciencia de Datos II** del **Máster Universitario en Ingeniería Informática** de la **Universidad de Málaga**.

Su objetivo principal es aplicar de forma práctica los conocimientos adquiridos durante la asignatura mediante el análisis de un conjunto de datos reales. En concreto, se trabaja con archivos JSON generados por la plataforma **VirusTotal**, que contiene información detallada sobre archivos analizados por diferentes motores antivirus.

El flujo del proyecto incluye las siguientes fases:

1. **Preprocesamiento de los datos**: lectura de los archivos JSON, validación y conversión en una estructura manejable (`DataFrame`).
2. **Limpieza**: detección y tratamiento de valores nulos, duplicados o estructuras inconsistentes.
3. **Almacenamiento en MongoDB**: una base de datos NoSQL adecuada para trabajar con documentos tipo JSON.
4. **Exploración y análisis mediante consultas**: utilizando funciones como `find()` y `aggregate()`.
5. **Visualización de resultados**: a través de gráficas descriptivas y estadísticas.
6. **Mejoras de rendimiento**: aplicación de índices en campos clave de la base de datos.

Este trabajo ha sido realizado por **Víctor Rodríguez Machiquez**, y refleja el desarrollo completo de un flujo de ingeniería de datos moderno sobre datos semiestructurados.

2. Fase de Preprocesamiento de los Datos

En esta fase se procede a la lectura y unificación de los archivos JSON proporcionados por VirusTotal. Cada uno de estos archivos contiene un único reporte generado por varios motores antivirus para un archivo concreto.

El objetivo principal del preprocesamiento es estructurar la información en diferentes niveles de granularidad para facilitar su análisis posterior. Para ello se generan tres estructuras clave:

`df1`: Información general por archivo

Este DataFrame contiene un resumen de cada archivo enviado a VirusTotal, incluyendo:

2. Fase de Preprocesamiento de los Datos

En esta fase se procede a la lectura y unificación de los archivos JSON proporcionados por VirusTotal. Cada uno de estos archivos contiene un único reporte generado por varios motores antivirus para un archivo concreto.

El objetivo principal del preprocesamiento es estructurar la información en diferentes niveles de granularidad para facilitar su análisis posterior. Para ello se generan tres estructuras clave:

`df1`: Información general por archivo

Este DataFrame contiene un resumen de cada archivo enviado a VirusTotal, incluyendo:

- `vhash`
- `community_reputation`
- `first_seen`
- `submission.date`
- `submission.submitter_country`

Es útil para análisis descriptivo y exploratorio a nivel de país, reputación o cronología.

`df2`: Resultados técnicos por motor antivirus

Este DataFrame se genera recorriendo la sección `"scans"` de cada archivo, y contiene una fila por cada motor antivirus que analizó un archivo. Incluye:

- `engine` (nombre del motor)
- `vhash`
- `detected` (si lo marcó como malicioso)
- `version`, `result`, `update`

Es útil para análisis técnico de cobertura y comportamiento de los motores antivirus.

`df3`: Unión enriquecida `df2 + df1`

El tercer DataFrame combina la información de los anteriores utilizando el campo `vhash` como clave de unión. Esto permite realizar análisis cruzados, como:

🔗 `df3`: Unión enriquecida `df2 + df1`

El tercer DataFrame combina la información de los anteriores utilizando el campo `vhash` como clave de unión. Esto permite realizar análisis cruzados, como:

- Qué motores detectan más amenazas por país.
- Comparar reputación comunitaria con efectividad de detección.
- Explorar tendencias por región y tipo de detección.

Esta fase no realiza aún limpieza, pero deja claramente estructurados los datos para su análisis posterior en MongoDB y Python.

[ ] ↳ 7 celdas ocultas

3. Fase de Limpieza de los Datos

Tras el preprocesamiento y la generación de los tres `DataFrames` principales (`df1`, `df2`, `df3`), esta fase se centra en revisar y preparar los datos para su posterior análisis y almacenamiento.

Dado que los archivos JSON originales contienen estructuras complejas y una gran variedad de campos, es necesario:

🧅 Objetivos de limpieza:

- Detectar y gestionar **valores nulos o desconocidos**
- Eliminar **duplicados** si los hubiera
- Asegurar la **consistencia de los tipos de datos**, especialmente fechas
- Preparar el `DataFrame` final (`df3`) para su uso en MongoDB y visualización

✳️ Limpieza aplicada:

- En `df1`, las fechas `first_seen` y `submission.date` fueron convertidas a tipo `datetime`, y los campos faltantes marcados como `"Desconocido"`.
- En `df2`, la columna `update` (de los motores antivirus) se transformó al formato de fecha `%Y%m%d`.
- En `df3`, se revisó que todas las columnas clave (`detected`, `engine`, `country`, etc.) estuvieran completas y sincronizadas.

⚠️ Notas técnicas:

✳️ Limpieza aplicada:

- En `df1`, las fechas `first_seen` y `submission.date` fueron convertidas a tipo `datetime`, y los campos faltantes marcados como `"Desconocido"`.
- En `df2`, la columna `update` (de los motores antivirus) se transformó al formato de fecha `%Y%m%d`.
- En `df3`, se revisó que todas las columnas clave (`detected`, `engine`, `country`, etc.) estuvieran completas y sincronizadas.

⚠️ Notas técnicas:

- Las fechas fueron convertidas a objetos `datetime.datetime` de Python, compatibles con MongoDB.
- Se usó `merge` por `vhash` para asegurar integridad relacional entre los DataFrames.
- No se eliminaron columnas por sparsidad ya que `df3` contiene solo campos relevantes tras la fusión.

Este proceso nos deja con un conjunto de datos limpio, estructurado y listo para ser almacenado en MongoDB Atlas o utilizado en análisis exploratorios posteriores.

[9] 0s

```
# --- ESTADO ANTES DE LA LIMPIEZA ---
print("Tamaño original:")
print(f"df1: {df1.shape}, df2: {df2.shape}, df3: {df3.shape}")

print("\nDuplicados antes (por vhash + engine):", df3.duplicated(subset=["vhash", "engine"]).sum())

print("\nNulos antes (columnas clave en df3):")
print(df3[["vhash", "engine", "detected", "submission.submitter_country"]].isnull().sum())

print("\nTipos de datos antes (fechas en df3):")
print(df3[["first_seen", "submission.date", "update"]].dtypes)


# --- LIMPIEZA ---
# Convertir fechas
df1["first_seen"] = pd.to_datetime(df1["first_seen"], errors="coerce")
df1["submission.date"] = pd.to_datetime(df1["submission.date"], errors="coerce")
df2["update"] = pd.to_datetime(df2["update"], errors="coerce", format="%Y%m%d")

# Eliminar duplicados
df1 = df1.drop_duplicates(subset="vhash")
```

```python
# --- ESTADO ANTES DE LA LIMPIEZA ---
print("Tamaño original:")
print(f"df1: {df1.shape}, df2: {df2.shape}, df3: {df3.shape}")

print("\nDuplicados antes (por vhash + engine):", df3.duplicated(subset=["vhash", "engine"]).sum())

print("\nNulos antes (columnas clave en df3):")
print(df3[["vhash", "engine", "detected", "submission.submitter_country"]].isnull().sum())

print("\nTipos de datos antes (fechas en df3):")
print(df3[["first_seen", "submission.date", "update"]].dtypes)


# --- LIMPIEZA ---
# Convertir fechas
df1["first_seen"] = pd.to_datetime(df1["first_seen"], errors="coerce")
df1["submission.date"] = pd.to_datetime(df1["submission.date"], errors="coerce")
df2["update"] = pd.to_datetime(df2["update"], errors="coerce", format="%Y%m%d")

# Eliminar duplicados
df1 = df1.drop_duplicates(subset="vhash")
df2 = df2.drop_duplicates(subset=["vhash", "engine"])
df3 = df3.drop_duplicates(subset=["vhash", "engine"])


# --- ESTADO DESPUÉS DE LA LIMPIEZA ---
print("\nTamaño después de limpieza:")
print(f"df1: {df1.shape}, df2: {df2.shape}, df3: {df3.shape}")

print("\nNulos después (columnas clave en df3):")
print(df3[["vhash", "engine", "detected", "submission.submitter_country"]].isnull().sum())

print("\nTipos de datos después (fechas en df3):")
print(df3[["first_seen", "submission.date", "update"]].dtypes)

print("\nDuplicados después:", df3.duplicated(subset=["vhash", "engine"]).sum())
```

```
Tamaño original:
df1: (157, 5), df2: (9564, 6), df3: (180882, 10)
```

```
[9] print("\nDuplicados después:", df3.duplicated(subset=["vhash", "engine"]).sum())
```

```
Tamaño original:
df1: (157, 5), df2: (9564, 6), df3: (180882, 10)

Duplicados antes (por vhash + engine): 177306

Nulos antes (columnas clave en df3):
vhash                          222
engine                           0
detected                         0
submission.submitter_country    127
dtype: int64

Tipos de datos antes (fechas en df3):
first_seen         datetime64[ns]
submission.date    datetime64[ns]
update             datetime64[ns]
dtype: object

Tamaño después de limpieza:
df1: (58, 5), df2: (3576, 6), df3: (3576, 10)

Nulos después (columnas clave en df3):
vhash                           59
engine                           0
detected                         0
submission.submitter_country    127
dtype: int64

Tipos de datos después (fechas en df3):
first_seen         datetime64[ns]
submission.date    datetime64[ns]
update             datetime64[ns]
dtype: object

Duplicados después: 0
```

✅ Validación de limpieza aplicada

Tras aplicar la limpieza, se verificaron los siguientes puntos:

✅ Validación de limpieza aplicada

Tras aplicar la limpieza, se verificaron los siguientes puntos:

- No existían duplicados por combinación de `vhash` + `engine`.
- Las fechas fueron convertidas correctamente a tipo `datetime`.
- Se mantuvieron los tamaños de los `DataFrames`, ya que no fue necesario eliminar registros.
- Se detectaron algunos valores nulos en campos como `vhash` y `submission.submitter_country`, lo cual es coherente con las limitaciones de los datos originales (por ejemplo, campos no siempre presentes en los JSON).

Este diagnóstico garantiza que los datos están estructurados, limpios y listos para su inserción en base de datos y análisis posteriores.

4. Almacenamiento en MongoDB

Tras completar la limpieza de los datos, se procede a almacenar los resultados en una base de datos **MongoDB Atlas**. Esta plataforma permite trabajar eficientemente con documentos semiestructurados, como los generados a partir de los archivos JSON de VirusTotal.

Dado que el proyecto ha estructurado la información en tres `DataFrames` independientes, se crea una colección separada para cada uno:

- `df1_info` → Información general por archivo (país, reputación, fecha de envío...)
- `df2_scans` → Resultados de escaneo por motor antivirus
- `df3_limpio` → Unión enriquecida entre los dos anteriores, lista para análisis y visualización

Esta separación permite realizar análisis independientes o combinados, optimizar consultas y aplicar índices específicos según el tipo de información.

> ⚠️ Se utiliza una base de datos remota llamada `ProyectoCD2Verm`, accesible desde MongoDB Atlas.

[10] 11 s
```
from pymongo import MongoClient
import json

# Conectar a MongoDB Atlas
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]
```

4. Almacenamiento en MongoDB

Tras completar la limpieza de los datos, se procede a almacenar los resultados en una base de datos **MongoDB Atlas**. Esta plataforma permite trabajar eficientemente con documentos semiestructurados, como los generados a partir de los archivos JSON de VirusTotal.

Dado que el proyecto ha estructurado la información en tres `DataFrames` independientes, se crea una colección separada para cada uno:

- `df1_info` → Información general por archivo (país, reputación, fecha de envío...)
- `df2_scans` → Resultados de escaneo por motor antivirus
- `df3_limpio` → Unión enriquecida entre los dos anteriores, lista para análisis y visualización

Esta separación permite realizar análisis independientes o combinados, optimizar consultas y aplicar índices específicos según el tipo de información.

> ⚠️ Se utiliza una base de datos remota llamada `ProyectoCD2Verm`, accesible desde MongoDB Atlas.

[10] 11 s

```
from pymongo import MongoClient
import json

# Conectar a MongoDB Atlas
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]

# --- Insertar df1 ---
collection1 = db["df1_info"]
collection1.drop()
collection1.insert_many(df1.to_dict(orient="records"))
print(f"df1_info: {collection1.count_documents({})} documentos insertados")

# --- Insertar df2 ---
collection2 = db["df2_scans"]
collection2.drop()
collection2.insert_many(df2.to_dict(orient="records"))
print(f"df2_scans: {collection2.count_documents({})} documentos insertados")

# --- Insertar df3 (principal) ---
collection3 = db["df3_limpio"]
```

```python
from pymongo import MongoClient
import json

# Conectar a MongoDB Atlas
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]

# --- Insertar df1 ---
collection1 = db["df1_info"]
collection1.drop()
collection1.insert_many(df1.to_dict(orient="records"))
print(f"df1_info: {collection1.count_documents({})} documentos insertados")

# --- Insertar df2 ---
collection2 = db["df2_scans"]
collection2.drop()
collection2.insert_many(df2.to_dict(orient="records"))
print(f"df2_scans: {collection2.count_documents({})} documentos insertados")

# --- Insertar df3 (principal) ---
collection3 = db["df3_limpio"]
collection3.drop()
collection3.insert_many(df3.to_dict(orient="records"))
print(f"df3_limpio: {collection3.count_documents({})} documentos insertados")
```

```
df1_info: 58 documentos insertados
df2_scans: 3576 documentos insertados
df3_limpio: 3576 documentos insertados
```

5. Exploración de Datos usando MongoDB

Una vez que los datos han sido preprocesados, limpiados y almacenados en la base de datos **MongoDB Atlas**, se procede a realizar una fase de exploración para obtener información útil y patrones relevantes directamente desde las colecciones.

En esta nueva estructura, los datos se han distribuido en tres colecciones especializadas:

5. Exploración de Datos usando MongoDB

Una vez que los datos han sido preprocesados, limpiados y almacenados en la base de datos **MongoDB Atlas**, se procede a realizar una fase de exploración para obtener información útil y patrones relevantes directamente desde las colecciones.

En esta nueva estructura, los datos se han distribuido en tres colecciones especializadas:

- `df1_info`: Contiene información general de los archivos, como fechas de detección, país del remitente y reputación comunitaria.
- `df2_scans`: Contiene los resultados de los análisis realizados por diferentes motores antivirus para cada archivo.
- `df3_limpio`: Es una vista enriquecida que une la información de `df1_info` y `df2_scans`, facilitando análisis conjuntos.

🔍 Consultas `find()` por colección

Estas consultas permiten explorar los datos con filtros simples y recuperar documentos individuales o subconjuntos relevantes. Algunos ejemplos:

- Obtener los primeros archivos reportados (`df1_info`) para ver su país de origen o reputación.
- Mostrar los primeros escaneos en `df2_scans` para identificar qué antivirus realizaron detecciones.
- Consultar desde `df3_limpio` para ver combinaciones de archivo + motor antivirus + resultado.

📊 Consultas `aggregate()` por colección

Permiten aplicar transformaciones complejas sobre los documentos, como:

- Contar cuántos archivos ha detectado cada motor antivirus.
- Agrupar por país (`submission.submitter_country`) y contar cuántos archivos se han enviado desde cada uno.
- Calcular estadísticas como el número total de detecciones por tipo de archivo o reputación comunitaria.

Esta fase es clave para responder preguntas como:

- ¿Qué antivirus detectan más amenazas?
- ¿Desde qué países se han enviado más archivos sospechosos?
- ¿Cuál es la distribución general de reputaciones o fechas de detección?

Esta exploración guía directamente a la siguiente fase: la **visualización de resultados**.

proyecto2cd2.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Compartir Gemini

Comandos + Código + Texto

RAM Disco

[11]
```
from pymongo import MongoClient
from pprint import pprint

# Conexión a MongoDB Atlas
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]

# Colecciones
col_info = db["df1_info"]
col_scans = db["df2_scans"]
col_limpio = db["df3_limpio"]

# 1. Consultas básicas (find) ----------------------

print("Primeros archivos reportados (df1_info):")
for doc in col_info.find().limit(3):
    pprint({
        "vhash": doc.get("vhash"),
        "first_seen": doc.get("first_seen"),
        "submitter_country": doc.get("submission.submitter_country")
    })
print("-" * 60)

print("Primeros resultados de escaneo (df2_scans):")
for doc in col_scans.find().limit(3):
    pprint({
        "vhash": doc.get("vhash"),
        "engine": doc.get("engine"),
        "detected": doc.get("detected"),
        "result": doc.get("result")
    })
print("-" * 60)

print("Vista combinada (df3_limpio):")
for doc in col_limpio.find().limit(3):
    pprint({
        "vhash": doc.get("vhash"),
        "engine": doc.get("engine"),
        "detected": doc.get("detected"),
```

Variables Terminal

16:32 Python 3

```
    })
print("-" * 60)

print("Vista combinada (df3_limpio):")
for doc in col_limpio.find().limit(3):
    pprint({
        "vhash": doc.get("vhash"),
        "engine": doc.get("engine"),
        "detected": doc.get("detected"),
        "submitter_country": doc.get("submission.submitter_country")
    })
print("-" * 60)

# 2. Consultas aggregate --------------------------

print("Motores antivirus con más detecciones:")
pipeline_av = [
    {"$match": {"detected": True}},
    {"$group": {"_id": "$engine", "total": {"$sum": 1}}},
    {"$sort": {"total": -1}},
    {"$limit": 5}
]
for r in col_scans.aggregate(pipeline_av):
    print(f"Motor: {r['_id']} - Detecciones: {r['total']}")
print("-" * 60)

print("Archivos por país de origen:")
pipeline_country = [
    {"$match": {"submission.submitter_country": {"$exists": True}}},
    {"$group": {"_id": "$submission.submitter_country", "archivos": {"$sum": 1}}},
    {"$sort": {"archivos": -1}},
    {"$limit": 5}
]
for r in col_info.aggregate(pipeline_country):
    print(f"País: {r['_id']} - Archivos: {r['archivos']}")
```

```
Primeros archivos reportados (df1_info):
{'first_seen': datetime.datetime(2021, 3, 20, 22, 55, 59),
 'submitter_country': 'CZ',
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
{'first_seen': datetime.datetime(2021, 4, 15, 4, 29, 56),
```

```
Primeros archivos reportados (df1_info):
{'first_seen': datetime.datetime(2021, 3, 20, 22, 55, 59),
 'submitter_country': 'CZ',
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
{'first_seen': datetime.datetime(2021, 4, 15, 4, 29, 56),
 'submitter_country': 'US',
 'vhash': '7c5152b8a2e031c3ff3071ac99fb71bc'}
{'first_seen': datetime.datetime(2021, 1, 31, 19, 35, 25),
 'submitter_country': 'DE',
 'vhash': '0e76fbf01fbf96f4db8b543c3c1ed34a'}
------------------------------------------------------------
Primeros resultados de escaneo (df2_scans):
{'detected': False,
 'engine': 'Bkav',
 'result': None,
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
{'detected': False,
 'engine': 'Cynet',
 'result': None,
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
{'detected': False,
 'engine': 'FireEye',
 'result': None,
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
------------------------------------------------------------
Vista combinada (df3_limpio):
{'detected': False,
 'engine': 'Bkav',
 'submitter_country': 'CZ',
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
{'detected': False,
 'engine': 'Cynet',
 'submitter_country': 'CZ',
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
{'detected': False,
 'engine': 'FireEye',
 'submitter_country': 'CZ',
 'vhash': '7596fdd04dba990373ab2f3da0c7dd3f'}
------------------------------------------------------------
Motores antivirus con más detecciones:
Motor: CAT-QuickHeal - Detecciones: 54
Motor: AegisLab - Detecciones: 53
Motor: McAfee - Detecciones: 53
Motor: K7GW - Detecciones: 52
```

```
------------------------------------------------------------
Motores antivirus con más detecciones:
Motor: CAT-QuickHeal - Detecciones: 54
Motor: AegisLab - Detecciones: 53
Motor: McAfee - Detecciones: 53
Motor: K7GW - Detecciones: 52
Motor: Avira - Detecciones: 52
------------------------------------------------------------
Archivos por país de origen:
```

6. Visualización de Resultados

Después de explorar los datos mediante consultas directas a MongoDB, se procede a visualizar algunos patrones clave para facilitar la interpretación de los resultados.

Las visualizaciones se basan en los datos de la colección `df3_limpio`, que combina información técnica (por motor antivirus) con metadatos generales del archivo.

Para ello se recuperan los datos mediante `pymongo`, se transforman en un `DataFrame` con `pandas`, y se representan utilizando `matplotlib`.

📊 Visualizaciones propuestas

1. **Top 10 motores antivirus con más detecciones**
   ¿Qué motores marcaron más archivos como maliciosos?
2. **Distribución por país de envío de archivos**
   ¿Desde qué países se suben más archivos a VirusTotal?
3. **Tendencia temporal de escaneos**
   Evolución del número de archivos escaneados por fecha (`first_seen`).

Estas gráficas ayudan a entender:

- La cobertura y actividad de los motores de detección.
- Posibles focos geográficos de actividad maliciosa.
- Comportamiento temporal del dataset.

```
[12] from pymongo import MongoClient
```

proyecto2cd2.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Compartir Gemini

Comandos + Código + Texto

RAM Disco

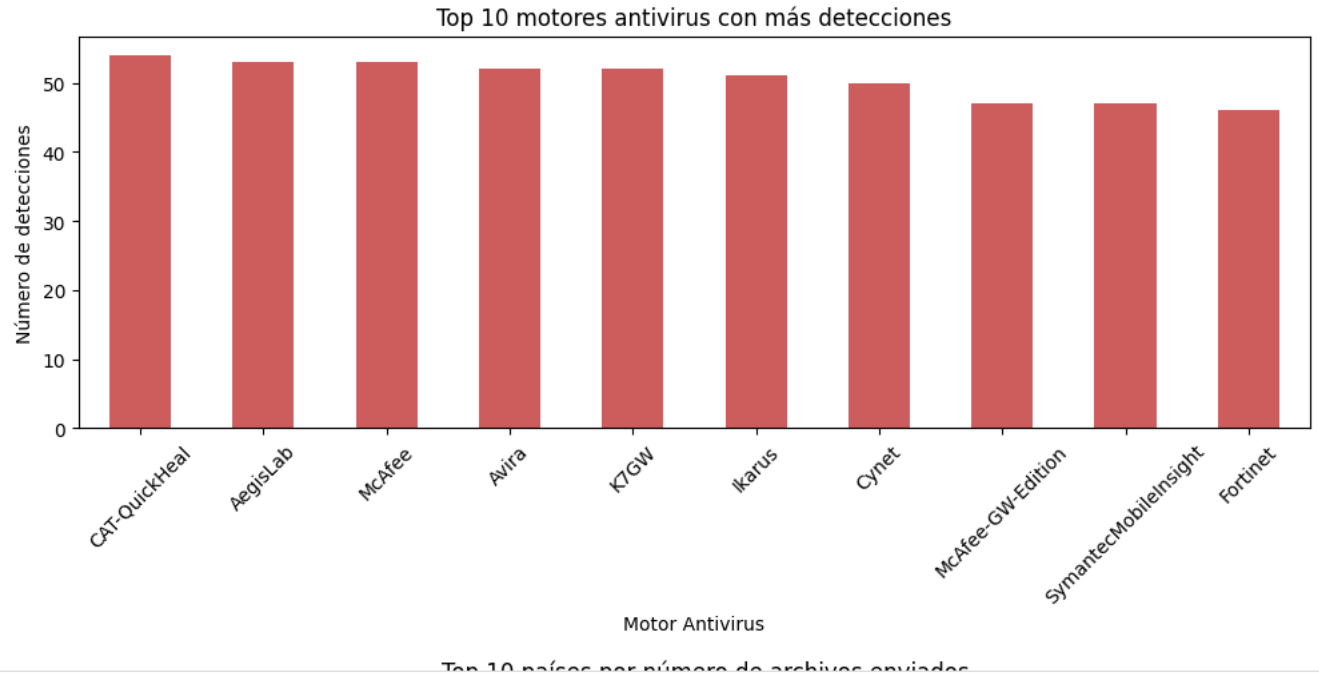
[12] 1s

```
from pymongo import MongoClient
import pandas as pd
import matplotlib.pyplot as plt

# Conexión a MongoDB Atlas
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]
col = db["df3_limpio"]

# Cargar todos los documentos como DataFrame
datos = list(col.find())
df = pd.DataFrame(datos)

# Asegurarse de que fechas estén bien
df["first_seen"] = pd.to_datetime(df["first_seen"], errors="coerce")

# --- Gráfico 1: Top 10 motores con más detecciones ---
top_motores = df[df["detected"] == True]["engine"].value_counts().head(10)

plt.figure(figsize=(10,5))
top_motores.plot(kind="bar", color="indianred")
plt.title("Top 10 motores antivirus con más detecciones")
plt.xlabel("Motor Antivirus")
plt.ylabel("Número de detecciones")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# --- Gráfico 2: Archivos por país de envío ---
top_paises = df["submission.submitter_country"].dropna().value_counts().head(10)

plt.figure(figsize=(10,5))
top_paises.plot(kind="bar", color="skyblue")
plt.title("Top 10 países por número de archivos enviados")
plt.xlabel("País")
plt.ylabel("Cantidad de archivos")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

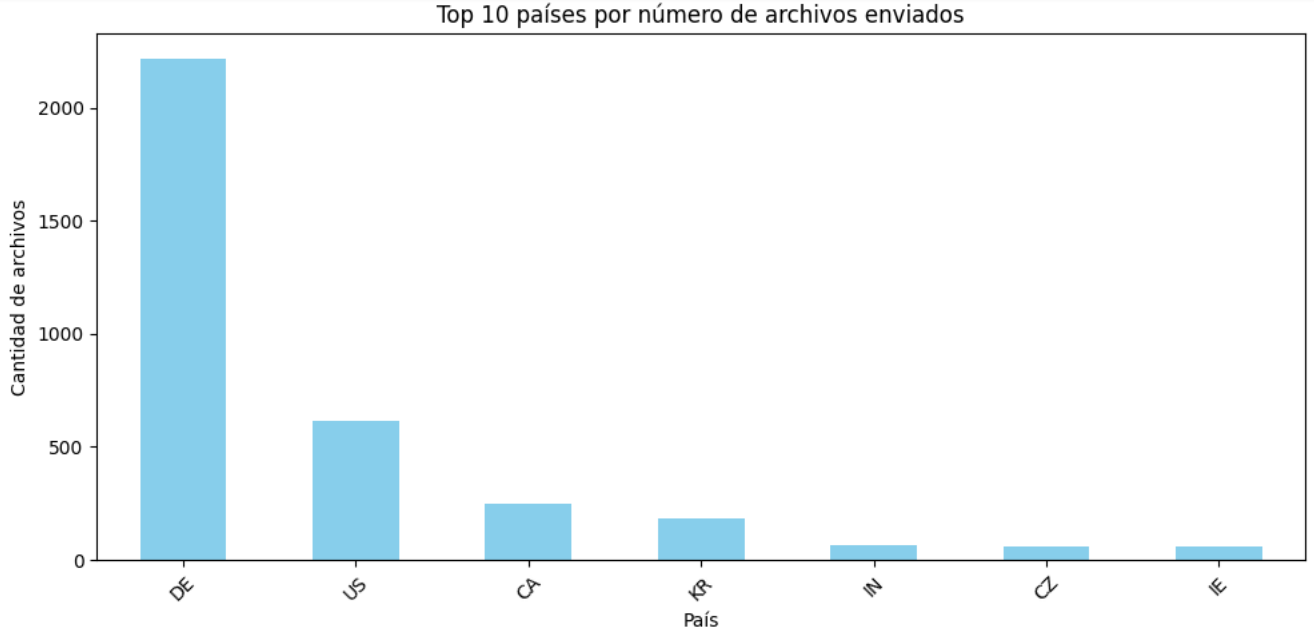
Variables Terminal

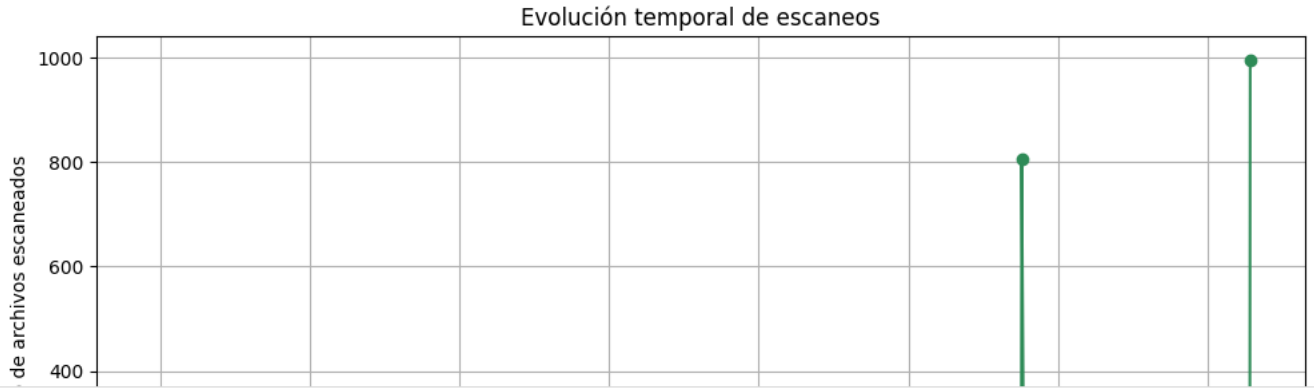
16:32 Python 3

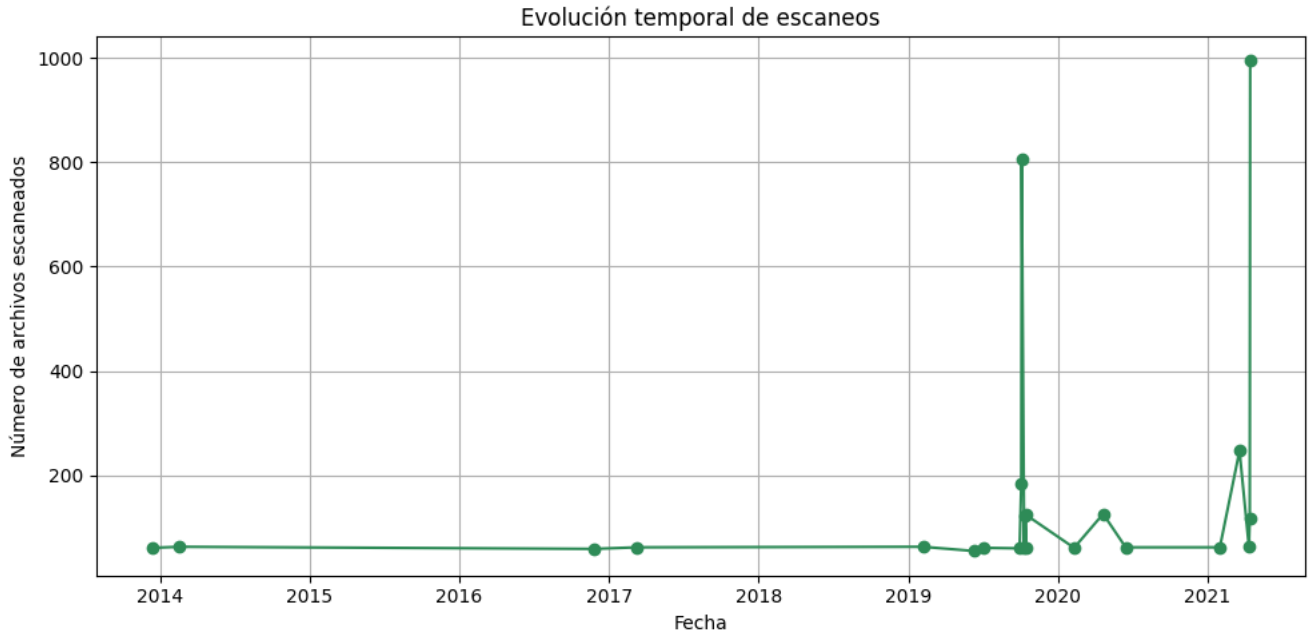
```python
# --- Gráfico 3: Evolución temporal de escaneos ---
df_fecha = df[["first_seen"]].dropna()
scans_por_fecha = df_fecha.groupby(df_fecha["first_seen"].dt.date).size()

plt.figure(figsize=(10,5))
scans_por_fecha.plot(kind="line", marker="o", color="seagreen")
plt.title("Evolución temporal de escaneos")
plt.xlabel("Fecha")
plt.ylabel("Número de archivos escaneados")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Top 10 países por número de archivos enviados


Top 10 países por número de archivos enviados
Cantidad de archivos
2000
1500
1000
500
0
DE
US
CA
KR
IN
CZ
IE
País


Evolución temporal de escaneos
de archivos escaneados
1000
800
600
400



7. Mejora del Rendimiento: Índices en MongoDB

Para optimizar el rendimiento de las consultas realizadas sobre las colecciones en MongoDB Atlas, se crean índices sobre los campos más utilizados en búsquedas y agrupaciones.

Los índices permiten que MongoDB localice documentos de forma más rápida, sin necesidad de recorrer toda la colección. Son especialmente útiles para consultas complejas y operaciones `aggregate`.

Índices aplicados

- `vhash`: clave primaria de los archivos (presente en todas las colecciones)

7. Mejora del Rendimiento: Índices en MongoDB

Para optimizar el rendimiento de las consultas realizadas sobre las colecciones en MongoDB Atlas, se crean índices sobre los campos más utilizados en búsquedas y agrupaciones.

Los índices permiten que MongoDB localice documentos de forma más rápida, sin necesidad de recorrer toda la colección. Son especialmente útiles para consultas complejas y operaciones `aggregate`.

🔧 Índices aplicados

- `vhash`: clave primaria de los archivos (presente en todas las colecciones)
- `engine`: motor antivirus, importante en `df2_scans` y `df3_limpio`
- `detected`: usado en filtros por detecciones positivas
- `submission.submitter_country`: usado en agrupaciones geográficas
- `first_seen`: relevante para ordenamientos cronológicos

Estos índices mejoran la velocidad de ejecución de las consultas y permiten escalar el análisis si el número de documentos crece significativamente.

[13]
```
from pymongo import MongoClient

# Conexión a MongoDB
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]

# --- Ver índices antes ---
print("Índices ANTES de crear (estado inicial):")

print("\ndf1_info:")
for name, index in db["df1_info"].index_information().items():
    print(f"  {name}: {index['key']}")

print("\ndf2_scans:")
for name, index in db["df2_scans"].index_information().items():
    print(f"  {name}: {index['key']}")
```

proyecto2cd2.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Compartir Gemini

Comandos + Código + Texto

RAM Disco

```python
from pymongo import MongoClient

# Conexión a MongoDB
client = MongoClient("mongodb+srv://victorioerm:lX8YbVqY7VPVEjDy@cluster0.7cqbm.mongodb.net/")
db = client["ProyectoCD2Verm"]

# --- Ver índices antes ---
print("Índices ANTES de crear (estado inicial):")

print("\ndf1_info:")
for name, index in db["df1_info"].index_information().items():
    print(f"  {name}: {index['key']}")

print("\ndf2_scans:")
for name, index in db["df2_scans"].index_information().items():
    print(f"  {name}: {index['key']}")

print("\ndf3_limpio:")
for name, index in db["df3_limpio"].index_information().items():
    print(f"  {name}: {index['key']}")

# --- Crear índices ---
db["df1_info"].create_index("vhash")
db["df1_info"].create_index("submission.submitter_country")
db["df1_info"].create_index("first_seen")

db["df2_scans"].create_index([("vhash", 1), ("engine", 1)])
db["df2_scans"].create_index("detected")

db["df3_limpio"].create_index("vhash")
db["df3_limpio"].create_index("engine")
db["df3_limpio"].create_index("detected")
db["df3_limpio"].create_index("submission.submitter_country")
db["df3_limpio"].create_index("first_seen")

print("\nÍndices creados correctamente.")

# --- Ver índices después ---
print("\nÍndices DESPUÉS de crear (estado final):")
```

Variables Terminal

16:32 Python 3

```python
# --- Ver índices después ---
print("\nÍndices DESPUÉS de crear (estado final):")

print("\ndf1_info:")
for name, index in db["df1_info"].index_information().items():
    print(f"  {name}: {index['key']}")

print("\ndf2_scans:")
for name, index in db["df2_scans"].index_information().items():
    print(f"  {name}: {index['key']}")

print("\ndf3_limpio:")
for name, index in db["df3_limpio"].index_information().items():
    print(f"  {name}: {index['key']}")
```

```
Índices ANTES de crear (estado inicial):

df1_info:
  _id_: [('_id', 1)]

df2_scans:
  _id_: [('_id', 1)]

df3_limpio:
  _id_: [('_id', 1)]

Índices creados correctamente.

Índices DESPUÉS de crear (estado final):

df1_info:
  _id_: [('_id', 1)]
  vhash_1: [('vhash', 1)]
  submission.submitter_country_1: [('submission.submitter_country', 1)]
  first_seen_1: [('first_seen', 1)]

df2_scans:
  _id_: [('_id', 1)]
  vhash_1_engine_1: [('vhash', 1), ('engine', 1)]
  detected_1: [('detected', 1)]

df3_limpio:
```

```
print("\ndf3_limpio:")
for name, index in db["df3_limpio"].index_information().items():
    print(f"  {name}: {index['key']}")
```

```
Índices ANTES de crear (estado inicial):

df1_info:
  _id_: [('_id', 1)]

df2_scans:
  _id_: [('_id', 1)]

df3_limpio:
  _id_: [('_id', 1)]

Índices creados correctamente.

Índices DESPUÉS de crear (estado final):

df1_info:
  _id_: [('_id', 1)]
  vhash_1: [('vhash', 1)]
  submission.submitter_country_1: [('submission.submitter_country', 1)]
  first_seen_1: [('first_seen', 1)]

df2_scans:
  _id_: [('_id', 1)]
  vhash_1_engine_1: [('vhash', 1), ('engine', 1)]
  detected_1: [('detected', 1)]

df3_limpio:
  _id_: [('_id', 1)]
  vhash_1: [('vhash', 1)]
  engine_1: [('engine', 1)]
  detected_1: [('detected', 1)]
  submission.submitter_country_1: [('submission.submitter_country', 1)]
  first_seen_1: [('first_seen', 1)]
```