



— **ESCUELA POLITÉCNICA NACIONAL**
ESCUELA DE FORMACIÓN DE TECNÓLOGOS

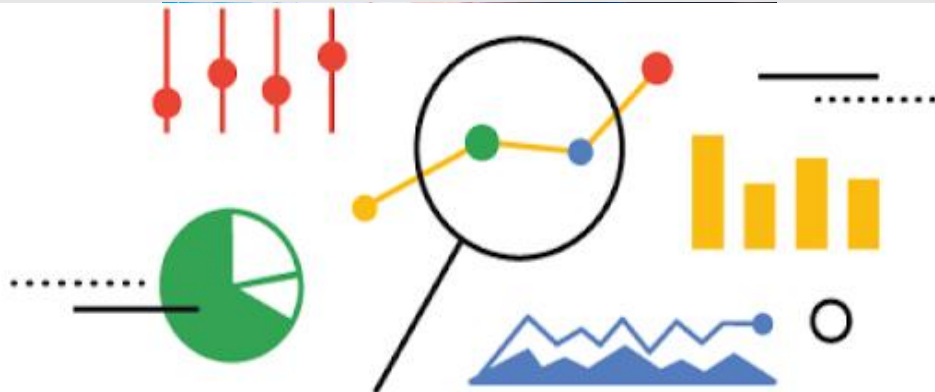


PROYECTO Final

ANÁLISIS DE DATOS

PROFESOR: Ing. Lorena Chulde / Ing. Juan Pablo Zaldumbide
PERÍODO ACADÉMICO: 2025-A
FECHA DE ENTREGA: 05 / 08 / 2025

PROYECTO FINAL



Integrantes:
Mateo Barba
Daniel Diaz
Andrés Panchi
Mercy Perugachi

2025-A

DEFINICIÓN DEL CASO DE ESTUDIO

El proyecto se centra en la integración de sistemas de bases de datos relacionales y NoSQL para consolidar información proveniente de diversas fuentes y facilitar su análisis eficiente. Se utilizan PostgreSQL, MySQL y SQL Server como bases de datos relacionales, junto con MongoDB y CouchDB para la gestión de datos NoSQL. Esta combinación tecnológica permite una infraestructura flexible que soporta la migración, transformación y sincronización de datos entre diferentes formatos y modelos.

La información consolidada se almacena finalmente en SQL Server, desde donde se procesa y se visualiza mediante Power BI. Gracias a dashboards interactivos, es posible explorar patrones, realizar consultas dinámicas y extraer insights que apoyen la toma de decisiones en diversas áreas temáticas abordadas en el proyecto.

OBJETIVOS GENERAL Y ESPECÍFICOS

– Objetivo general

Diseñar e implementar una infraestructura de bases de datos que integre tecnologías SQL y NoSQL, permitiendo la unificación de datos en un repositorio centralizado en SQL Server, optimizado para su análisis y visualización a través de Power BI.

– Objetivos específicos

1. Integrar diferentes fuentes de datos provenientes de al menos cuatro tipos de bases de datos (PostgreSQL, MySQL, MongoDB y SQL Server), logrando una infraestructura unificada que facilite la interoperabilidad entre sistemas relacionales y NoSQL.
2. Establecer procesos eficientes de transferencia y transformación de datos entre bases relacionales y NoSQL por ejemplo, entre PostgreSQL, MySQL, MongoDB y CouchDB, garantizando la conversión de diferentes tipos de archivos (CSV, JSON, etc.) según los requisitos del análisis..
3. Desarrollar dashboards y visualizaciones en Power BI que permitan el análisis actualizado de los datos integrados, apoyando la generación de al menos 15 casos de estudio o escenarios de toma de decisiones basados en los resultados obtenidos.
4. Documentar detalladamente los procesos de extracción, limpieza, transformación, integración y análisis de

datos, facilitando la trazabilidad y replicabilidad del proyecto por parte de cualquier integrante del equipo o parte interesada.

DESCRIPCIÓN DEL EQUIPO DE TRABAJO Y ACTIVIDADES REALIZADAS POR CADA UNO.

Mateo Barba: Responsable de la extracción, limpieza e importación de archivos CSV, JSON y bases de datos NoSQL hacia la base de datos remota MongoDB Atlas. Trabajó principalmente con las temáticas de *Actividades y hobbies* y *Conciertos y eventos públicos* para la obtención de información. Tras la generación de archivos CSV por parte del equipo, integró la data en MongoDB de manera local y posteriormente estableció la conexión con SQL Server mediante Jupyter Notebook.

Daniel Díaz: Encargado de la extracción, limpieza e importación de archivos CSV, JSON y NoSQL hacia la base de datos CouchDB. Su labor se centró en las temáticas de *Eventos y noticias mundiales* para recopilar la información correspondiente.

Andrés Panchi: Responsable de la extracción, limpieza e importación de archivos CSV, JSON y SQL hacia la base de datos PostgreSQL. Trabajó con las temáticas de *Actividades y hobbies* y *Eventos deportivos a nivel mundial* para la obtención de datos.

Mercy Perugachi: A cargo de la extracción, limpieza e importación de archivos CSV y JSON hacia la base de datos MySQL. Su temática asignada fue la definida por el grupo: *Reseñas de vino*. Además, estableció la conexión entre SQL Server y Power BI, realizando el análisis detallado de los datos obtenidos.

RECURSO Y HERRAMIENTAS UTILIZADAS

Para la realización del proyecto se emplearon diversas herramientas y tecnologías que permitieron llevar a cabo la

extracción, transformación, análisis y visualización de datos de manera eficiente. Las principales herramientas utilizadas fueron:

Power BI: Plataforma de visualización y análisis de datos utilizada para la creación de dashboards interactivos y la toma de decisiones basada en datos.

PostgreSQL: Base de datos relacional empleada para el almacenamiento y gestión estructurada de información.

MySQL: Sistema de gestión de bases de datos relacional utilizado para importar y centralizar diferentes fuentes de datos.

MongoDB: Base de datos NoSQL utilizada tanto en entornos locales como en la nube (MongoDB Atlas) para la manipulación flexible y escalable de grandes volúmenes de datos en formato JSON.

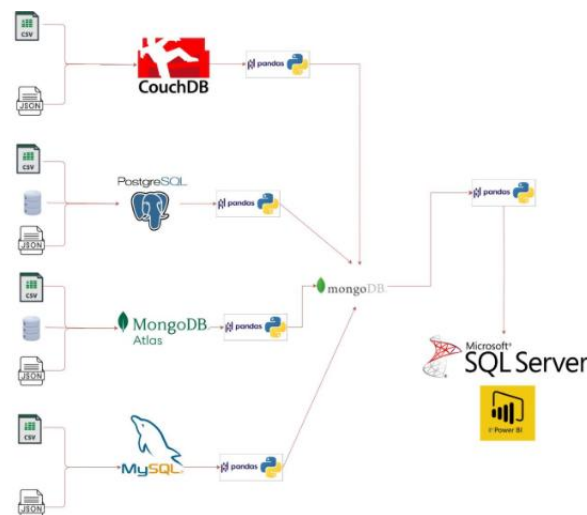
Microsoft Excel: Herramienta empleada en las etapas iniciales para la limpieza, transformación y manipulación de datos en archivos CSV.

SQL Server: Base de datos relacional utilizada como repositorio final y punto central de integración de toda la data procesada.

CouchDB: Base de datos NoSQL seleccionada para almacenar y consultar datos semiestructurados relacionados con eventos y noticias mundiales.

Jupyter Notebook: Entorno interactivo de desarrollo que facilitó la conexión entre las distintas bases de datos, la ejecución de scripts de integración y análisis, y la documentación del proceso de transformación y análisis de datos.

Diseño de la arquitectura



TEMAS GENERALES DE CASOS DE ESTUDIO

Se obtendrán Dashboards de los siguientes casos de estudio:

1. Actividades y Hobbies:

Este estudio permitirá un análisis detallado de actividades únicas. La combinación de análisis de datos, visualización con Power BI y transformación de formatos (CSV, JSON, etc.) de

1.1 Conjunto de datos de actividades diversas

Este conjunto de datos ofrece una rica colección de 32.000 actividades únicas, que abarcan un amplio espectro de intereses y compromisos humanos. Diseñado para inspirar e informar, el conjunto de datos es un recurso invaluable para investigadores, desarrolladores de aplicaciones y cualquier persona interesada en el comportamiento humano y el diseño de actividades.

2. Eventos Públicos

2.1 Eventos especiales en parques

Este conjunto de datos contiene

información sobre los eventos especiales de Parks, incluidos fitness, deportes, baile, películas y conciertos facilitados por la división de Programas Públicos de NYC Parks.

Se trata de eventos puntuales que ocurren una sola vez; estos eventos no forman parte de una serie de programación que ocurre regularmente.

2.1 Eventos públicos en Empire State

Plaza: principios de 1999

Este conjunto de datos contiene una lista de eventos públicos celebrados en Empire State Plaza con información sobre los eventos, como fecha y hora, incluidos muchos próximos.

3. Noticias

3.1 Noticias BBC

Noticias destacadas de la última década, con el fin de obtener cuáles fueron los porcentajes tanto de noticias entrantes, cuantos casos fueron al año, de donde surgió dicha información, etc.

4. Actividades y Hobbies

4.1 Banco de datos de béisbol

Baseball Databank es una compilación de datos históricos del béisbol en un formato conveniente y ordenado, distribuido bajo términos de Datos Abiertos.

Esta versión del banco de datos de béisbol fue descargada de [Sitio web de Sean Lahman](#).

4.2 Recomendación de Spotify

Este conjunto contiene información de la música de Spotify con el fin de obtener información como la variabilidad de canciones favoritas solo con sus identificaciones.

4.3 goodreads_books_json

Este es un archivo de tipo Json, el cual contiene registros de libros leídos.

4.4 Conjunto de datos de ejercicios para miembros del gimnasio

Este conjunto de datos proporciona una descripción detallada de las rutinas de ejercicio, los atributos físicos y las métricas de aptitud física de los miembros del gimnasio. Contiene 973 muestras de datos de gimnasios, incluidos indicadores clave de rendimiento como frecuencia cardíaca, calorías quemadas y duración del entrenamiento. Cada entrada también incluye datos demográficos y niveles de experiencia, lo que permite un análisis exhaustivo de los patrones de aptitud física, la progresión de los atletas y las tendencias de salud.

5. Eventos Deportivos a Nivel Mundial

5.1 126 años de conjunto de datos olímpicos históricos

Este conjunto de datos presenta un archivo completo de la historia olímpica, que abarca 126 años desde los **primeros** Juegos Olímpicos modernos en Atenas en 1896 hasta los Juegos Olímpicos de Invierno de Beijing. Proporciona resultados a nivel de atleta, información biográfica detallada y métricas de rendimiento, lo que lo convierte en un recurso valioso para analistas deportivos, entusiastas de datos, investigadores e historiadores.

6. Actividades y Hobbies

6.1 Reseñas de Vino

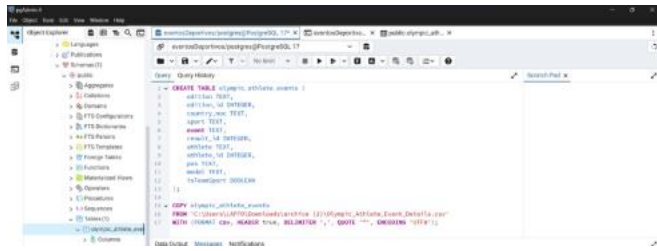
(MERCY coloca información del dataset, búsate en los ejemplos de arriba)

EXTRACCIÓN DE DATOS.

POSTGRES (Panchi)

1.- Creo una base de datos y una tabla con las cabeceras de mi archivo.csv, este paso

para cada archivo CSV, una vez que haya realizado la respectiva limpieza de cada dataset mediante jupyter notebook.

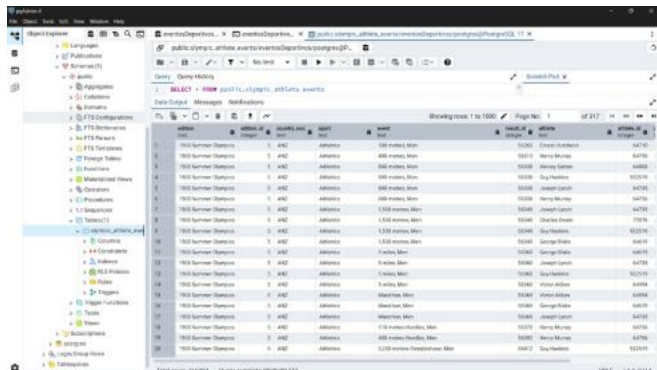


- Como me da error de copiar el archivo, me voy a psql y pego el siguiente comando:
- `\copy olympic_athlete_events FROM 'C:\Users\LAPTO\Downloads\arc s.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', QUOTE '', ENCODING 'UTF8')`

2.- Crear la Base de Datos y la Tabla en PostgreSQL

Del dataset de baseball, creo una tabla fielding.

Nos conectamos a la base de datos y creamos las respectivas tablas.



3. - Importar el .json a PostgreSQL

Primero convierto mi archivo .csv a json

```
import pandas as pd
import json

# Ruta del archivo
# df = pd.read_csv('data.csv')

# Ver los primeros filas
df.head()
```

	deniability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration.ms	signature	liked
0	0.883	0.6243	7	-6.764	0	0.0477	0.451	0.00734	0.1000	0.6280	95.908	304524	4	0
1	0.762	0.7030	1	-7.851	0	0.3060	0.306	0.000000	0.0912	0.5190	151.320	247178	4	1
2	0.261	0.6149	1	-27.528	0	0.0419	0.082	0.897500	0.1020	0.0282	75.296	280607	4	0
3	0.722	0.7360	3	-6.954	0	0.0585	0.431	0.000001	0.1230	0.5620	89.880	299039	4	1
4	0.787	0.5720	1	-7.518	1	0.2220	0.145	0.000500	0.0751	0.8470	155.117	179413	4	1


```
[14]: # Convertir a JSON (formato orientado a registros)
      json_data = df.to_json(orient='records', indent=2)

      # Mostrar una parte del resultado
      print(json_data[:500]) # Muestra Los primeros 500 caracteres

      [
        {
          "danceability":0.803,
          "energy":0.624,
          "key":7,
          "loudness":-6.764,
          "mode":0,
          "speechiness":0.0477,
          "acousticness":0.451,
          "instrumentalness":0.000734,
          "liveness":0.1,
          "valence":0.628,
          "tempo":95.968,
          "duration_ms":304524,
          "time_signature":4,
          "liked":0
        },
        {
          "danceability":0.762,
          "energy":0.703,
          "key":10,
          "loudness":-7.951,
          "mode":0,
          "speechiness":0.306,
          "acousticness":0.206,
          "instrumentalness":0.0,

```

- Después de convertir, procedo a guardar y se procede a conectar con PostgreSQL.

```
[15]: # Guardar el JSON en un archivo
      with open('music_data.json', 'w') as f:
          f.write(json_data)

      print('Archivo guardado como "music_data.json"')
      Archivo guardado como "music_data.json"

[16]: import psycopg2
      from psycopg2 import sql
      import pandas as pd
      import json

[17]: # Configuro los parámetros de conexión
      db_params = {
          'host': 'localhost',
          'database': 'music',
          'user': 'postgres',
          'password': '1234',
          'port': '5432' # Puerto por defecto de PostgreSQL
      }

[18]: def connect_to_db():
      try:
          # Establecer conexión
          conn = psycopg2.connect(**db_params)
          print("Conexión exitosa a PostgreSQL")
          return conn
      except Exception as e:
          print(f"Error al conectar a PostgreSQL: {e}")
          return None

```

- Creo una tabla e inserto la data de mi json

```
[19]: def create_table(conn):
      try:
          cursor = conn.cursor()

          create_table_query = """
          CREATE TABLE IF NOT EXISTS music_features (
              id SERIAL PRIMARY KEY,
              danceability FLOAT,
              energy FLOAT,
              key INT,
              loudness FLOAT,
              mode INT,
              speechiness FLOAT,
              acousticness FLOAT,
              instrumentalness FLOAT,
              liveness FLOAT,
              valence FLOAT,
              tempo FLOAT,
              duration_ms INT,
              time_signature INT,
              liked INT
          )
          """
          cursor.execute(create_table_query)
          conn.commit()
          print("Tabla creada exitosamente")
      except Exception as e:
          print(f"Error al crear tabla: {e}")
      finally:
          if cursor:
              cursor.close()

```

```
[20]: def insert_data_from_json(conn, json_file_path):
      try:
          # Cargar el archivo JSON
          with open(json_file_path, 'r') as f:
              data = json.load(f)

          cursor = conn.cursor()

          # Preparar consulta de inserción
          insert_query = """
          INSERT INTO music_features (
              danceability, energy, key, loudness, mode, speechiness,
              acousticness, instrumentalness, liveness, valence, tempo,
              duration_ms, time_signature, liked
          ) VALUES (
              %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
          )
          """

          # Insertar cada registro
          for record in data:
              cursor.execute(insert_query, (
                  record['danceability'],
                  record['energy'],
                  record['key'],
                  record['loudness'],
                  record['mode'],
                  record['speechiness'],
                  record['acousticness'],
                  record['instrumentalness'],
                  record['liveness'],
                  record['valence'],
                  record['tempo'],
                  record['duration_ms'],
                  record['time_signature'],
                  record['liked']
              ))

          conn.commit()
          print(f"Se insertaron {len(data)} registros exitosamente")
      except Exception as e:
          conn.rollback()
          print(f"Error al insertar datos: {e}")
      finally:
          if cursor:
              cursor.close()

```

```
[21]: # Ejecutar conexión
      conn = connect_to_db()

      if conn:
          # Crear tabla
          create_table(conn)

          # Insertar datos (compruebe de que el archivo JSON existe)
          insert_data_from_json(conn, 'music_data.json')

          # Opcional: Leer datos desde PostgreSQL a un Dataframe
          query = "SELECT * FROM music_features LIMIT 5"
          df_from_db = pd.read_sql(query, conn)
          print(f"Se obtuvieron {len(df_from_db)} registros desde PostgreSQL.")
          print(df_from_db)

          finally:
              # Cerrar conexión
              conn.close()
              print("Conexión cerrada")

      Conexión exitosa a PostgreSQL
      Tabla creada exitosamente
      105 registros insertados exitosamente

      Primeros 5 registros desde PostgreSQL:
      id danceability energy key loudness mode speechiness acousticness \
      0 1 0.803 0.624 7 -6.764 0 0.0477 0.451
      1 2 0.762 0.703 10 -7.951 0 0.306 0.206
      2 3 0.803 0.624 7 -6.764 0 0.0477 0.451
      3 4 0.762 0.703 10 -7.951 0 0.306 0.206
      4 5 0.803 0.624 7 -6.764 0 0.0477 0.451

      Instrumentalness liveness valence tempo duration_ms time_signature \
      0 0.000734 0.1 0.628 95.968 304524 4
      1 0.000000 0.0001 0.504 95.529 205736 4
      2 0.000000 0.0000 0.000 75.296 20087 4
      3 0.000000 0.0000 0.000 75.296 20087 4
      4 0.000000 0.0000 0.000 75.296 20087 4

```

- Y por último verifico la tabla en PostgreSQL.

id	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	0.803	0.624	7	-6.764	0	0.0477	0.451	0.000734	0.1	0.628	95.968	304524	4
1	0.762	0.703	10	-7.951	0	0.306	0.206	0.000000	0.0001	0.504	95.529	205736	4
2	0.803	0.624	7	-6.764	0	0.0477	0.451	0.000000	0.0000	0.000	75.296	20087	4
3	0.762	0.703	10	-7.951	0	0.306	0.206	0.000000	0.0000	0.000	75.296	20087	4
4	0.803	0.624	7	-6.764	0	0.0477	0.451	0.000000	0.0000	0.000	75.296	20087	4

```
[22]: # Carga el archivo JSON
      with open('music_data.json', 'r') as f:
          data = json.load(f)

      # Insertar datos
      insert_data_from_json(conn, 'music_data.json')

      # Verificar la inserción
      query = "SELECT * FROM music_features LIMIT 5"
      df_from_db = pd.read_sql(query, conn)
      print(df_from_db)

```

[illegible]

- Y conectamos a Postgresql.

```
[16] # Create a PostgreSQL
$ip install sqlalchemy psycopg2-binary
$apt-get purge as pd
$rm sqlalchemy $apt-get create_engine

[notice] A new release of pip is available: 20.0.1 -> 20.2
[notice] To update, run: python3 -m pip install --upgrade pip
Collecting sqlalchemy
  Downloading sqlalchemy-1.4.0-py3-none-any.whl (406kB)
Requirement already satisfied: psycopg2-binary in /usr/local/lib/python3.8/dist-packages (1.4.0)
Collecting greenlet<1.1, from sqlalchemy
  Downloading greenlet-0.4.13-py3.py3-none-any.whl (24kB)
Requirement already satisfied: psycopg2-binary==2.8.0 in /usr/local/lib/python3.8/dist-packages (from sqlalchemy) (2.8.0)
Installing sqlalchemy-1.4.0-py3.py3-none-any.whl (406kB)
.....
0.0/2.1 MB 7.0 MB/s eta 0:00:01
.....
1.0/2.1 MB 4.8 MB/s eta 0:00:01
.....
2.0/2.1 MB 7.1 MB/s eta 0:00:00
Installing greenlet 0.4.13 (py3.py3-none-any.whl)
Installing collected packages: greenlet, sqlalchemy
Successfully installed greenlet-0.4.13 sqlalchemy-1.4.0

[16] $cat > |$python3
constraints = "1234"
host = "localhost"
ports = "5432"
base_data = "libro5"

engine = create_engine(f'postgresql+psycopg2://{user}:{constraints}@{host}:{ports}/{base_data}')

[16] $python3
from sqlalchemy import create_engine

# Connect to column as list of strings 2020
cols = ["id", "authors", "popular_places", "similar_books", "series"]
for col in cols:
    if col in df.columns:
        df[col].values = df[col].values.apply(lambda x: json.loads(x) if isinstance(x, list) else [])

# Open connection
connection = postgresql
constraints = "1234"
host = "localhost"
ports = "5432"
base_data = "libro5"
engine = create_engine(f'postgresql+psycopg2://{user}:{constraints}@{host}:{ports}/{base_data}')

# Subit into 1000 files para probar
df['libro5_name'] = df['goodreads'].apply(lambda x: x['goodreads'], index=False)
```

MYSQL WORKBENCH (Perugachi)

Se detallan los pasos seguidos para la extracción, limpieza y carga del datasets de “vinos” en MySQL antes de su análisis en Power BI..

1. Preparación de los datasets

Antes de importar los datos a MySQL, se realizó un proceso de limpieza utilizando **pandas** en Python para asegurar su calidad y compatibilidad.

Las acciones realizadas incluyeron:

- **Carga de archivos CSV con pandas:**
Se usó `pd.read_csv()` para leer los datasets.
 - **Verificación del delimitador:** Se confirmó que los valores estuvieran separados por comas (,).
 - **Conversión de tipos de datos:** Se aseguraron los formatos adecuados para fechas, números y cadenas de texto.
 - **Eliminación de valores nulos y duplicados:**
 - Se reemplazaron valores faltantes con NULL o valores adecuados

según la variable.

- Se eliminaron registros duplicados con `drop_duplicates()`.

```
#limpieza de datos

import pandas as pd

if 'df_unified' not in locals():
    print("Error, no está definido 'df_unified'")
else:
    print("Iniciando limpieza de datos")
    print(f"Número de filas iniciales: {len(df_unified)}")

    # transformar tipos de datos
    # 'points' y 'price' a valores numéricos
    # errors='coerce' convierte cualquier valor no numérico a NaN
    print(" :::::::::::::::::::: Transformar tipos de datos ::::::::::::::::::::")
    df_unified['points'] = pd.to_numeric(df_unified['points'], errors='coerce')
    df_unified['price'] = pd.to_numeric(df_unified['price'], errors='coerce')

    print("\nTipos de datos después de la conversión:")
    print(df_unified[['points', 'price']].info())

    # VALORES NULOS (NaN) EN COLUMNAS DE TEXTO
    # Reemplazar los valores NaN con un string descriptivo
    print(" :::::::::::::::::::: Valores nulos ::::::::::::::::::::")
    for col in ['country', 'description', 'designation', 'province', 'region_1', 'region_2', 'variety', 'winery', 'title', 'taster_name', 'taster_twitter_handle']:
        if col in df_unified.columns:
            df_unified[col] = df_unified[col].fillna(desconocido)

    print("\nValores nulos después de reemplazar (deberían ser 0 para las columnas de texto):")
    print(df_unified.isnull().sum())

    # ELIMINAR DUPLICADOS
    print(" :::::::::::::::::::: Eliminar registros duplicados ::::::::::::::::::::")
    num_duplicates_initial = df_unified.duplicated(subset=['description', 'winery', 'points', 'price']).sum()
    print(f"Número de duplicados encontrados: {num_duplicates_initial}")

    df_unified.drop_duplicates(subset=['description', 'winery', 'points', 'price'], inplace=True)

    print(f"Número de filas finales después de eliminar duplicados: {len(df_unified)}")
```

```
# verificar
print(f"Validado final del DataFrame después de la limpieza:")
df_unified.info()

print("\nEstadísticas del DataFrame después de la limpieza:")
print(df_unified.describe(include='all'))

print("Limpieza completa")

# Limpieza de datos
Número de filas iniciales: 109052
:::::::::::::::::: Transformar tipos de datos ::::::::::::::::::::

Tipos de datos después de la conversión:
Out[10]: pandas.core.frame.DataFrame
Info: 109052 entries, 8 to 469576
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  --
 0   points      109052 non-null  float64
 1   price       356817 non-null  float64
 dtypes: float64(1), int64(1)
memory usage: 3.8 MB

# Valores nulos
:::::::::::::::::: Valores nulos ::::::::::::::::::::

Valores nulos después de reemplazar (deberían ser 0 para las columnas de texto):
id                0
points            0
title             0
description        0
taster_name       0
taster_twitter_handle  0
price            12870
designation        0
variety            0
region_1          0
region_2          0
province          0
country           0
winery            0
winery_score      0
dtypes: int64

# Eliminar registros duplicados
:::::::::::::::::: Eliminar registros duplicados ::::::::::::::::::::
Número de duplicados encontrados: 0
Número de filas finales después de eliminar duplicados: 109052
```

2. Creación de la Base de Datos y Tablas en

MySQL

Se utilizó MySQL Workbench para la creación de la base de datos wine y sus respectivas tablas.

Las tablas creadas almacenaron información clave como:

- Wines_reviews_csv:** Datos de puntuaciones Elo de clubes de fútbol.
- Wines_reviews_json:** Información detallada de

- partidos, incluyendo equipos, fechas, estadísticas y resultados.



3. Importación de los archivos CSV en MySQL opción de encabezados.

Para la carga de los datos, se utilizó la opción de Jupyter Notebbok y psasmos Workbench, siguiendo estos pasos:

- Abrir Jupyter Notebooky** conectar con el servidor MySQL.

```
# Importar CSV a MySQL

import json
import mysql.connector
from mysql.connector import Error
import pandas as pd

DB_CONFIG = {
    'host': '127.0.0.1',
    'database': 'wine',
    'user': 'root',
    'password': '1234'
}

# Ruta
CSV_FILE_PATH = 'C:/Users/USER/Desktop - Escuela Politécnica Nacional/Scriptorio/Análisis de Datos/ProyectoFinal400/winemag_data_first100k.csv'

def import_wine_csv_data():
    conn = None
    try:
        # Conectar a la base de datos MySQL
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor()

        # Cargar el archivo CSV en un DataFrame de Pandas
        df = pd.read_csv(CSV_FILE_PATH, index_col=0)
        df.index.name = 'id'
        df.reset_index(inplace=True)

        print(f"Total de reseñas de vino a importar desde CSV: {len(df)} filas.")

        # SQL para insertar datos
        insert_sql = """
INSERT INTO wine_reviews_csv (
    id, country, description, designation, points, price, province,
    region_1, region_2, variety, winery
) VALUES (
    %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
)
"""

        for i, row in enumerate(df.iterrows(index=False)):
            values = (
                row.id,
                row.country,
                row.description,
                row.designation if pd.notna(row.designation) else None,
                row.points,
                row.price if pd.notna(row.price) else None,
                row.province,
                row.region_1 if pd.notna(row.region_1) else None,
                row.region_2 if pd.notna(row.region_2) else None,
                row.variety,
                row.winery
            )

            try:
                cursor.execute(insert_sql, values)
                if (i + 1) % 1000 == 0:
                    print(f"Insertados {i + 1} registros...")
            except Error as err:
                print(f"Error al insertar la reseña {i+1} (ID: {row.id}): {err}")
                pass

        conn.commit()
        print("Importación de reseñas de vino desde CSV completada exitosamente.")

    except FileNotFoundError:
        print(f"Error: El archivo '{CSV_FILE_PATH}' no se encontró.")
    except pd.errors.EmptyDataError:
        print(f"Error: El archivo CSV '{CSV_FILE_PATH}' está vacío.")
    except pd.errors.ParserError as e:
        print(f"Error al parsear el CSV: {e}")
    except Error as e:
        print(f"Error de MySQL: {e}")
    except Exception as e:
        print(f"Ocurrió un error inesperado: {e}")
    finally:
        if conn and conn.is_connected():
            cursor.close()
            conn.close()
            print("Conexión a MySQL cerrada.")

if __name__ == "__main__":
    import_wine_csv_data()
```

- Abrir Jupyter Notebooky** conectar con el servidor MySQL.

Tecnología Superior en Desarrollo de Software

```
import json
import mysql.connector
from mysql.connector import Error

# configuración de mysql
DB_CONFIG = {
    'host': '127.0.0.1',
    'database': 'wine',
    'user': 'root',
    'password': '1212'
}

# ruta
JSON_FILE_PATH = 'C:/Users/USER/OneDrive - Escuela Politécnica Nacional/escritorio/Análisis de Datos/ProyectoFinal/400/dnema_data_1386_v2.json'

def import_wine_data():
    conn = None
    try:
        # Conectar a la base de datos MySQL
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor()

        with open(JSON_FILE_PATH, "r", encoding="utf-8") as f:
            data = json.load(f)

        if isinstance(data, list):
            print(f"Total de reseñas de vino a importar: {len(data)}")

            insert_sql = """
            INSERT INTO wine_reviews_json (
                points, title, description, taster_name, taster_twitter_handle,
                price, designation, variety, region_1, region_2, province, country, winery
            ) VALUES (
                %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
            )
            """

            for i, review in enumerate(data):
                values = (
                    int(review.get('points')) if review.get('points') is not None else None,
                    review.get('title'),
                    review.get('description'),
                    review.get('taster_name'),
                    review.get('taster_twitter_handle'),
                    float(review.get('price')) if review.get('price') is not None else None,
                    review.get('designation'),
                    review.get('variety'),
                    review.get('region_1'),
                    review.get('region_2'),
                    review.get('province'),
                    review.get('country'),
                    review.get('winery')
                )
                try:
                    cursor.execute(insert_sql, values)
                except Error as e:
                    print(f"Error al insertar la reseña ({i}): {err}")
                    print(f"Datos problemáticos: {values}")
                    pass

            # Confirmar los cambios en la base de datos
            conn.commit()
            print("Importación de reseñas de vino completada exitosamente.")
        else:
            print(f"El archivo JSON no es una lista de objetos, o tiene un formato inesperado.")

    except FileNotFoundError:
        print(f"Error: El archivo '{JSON_FILE_PATH}' no se encontró.")
    except json.JSONDecodeError as e:
        print(f"Error al decodificar el JSON: {e}")
    except Exception as e:
        print(f"Error al decodificar el JSON: {e}")
    finally:
        if conn and conn.is_connected():
            cursor.close()
            conn.close()
            print("Conexión a MySQL cerrada.")

if __name__ == "__main__":
    import_wine_data()

Total de reseñas de vino a importar: 129973
Importación de reseñas de vino completada exitosamente.
Conexión a MySQL cerrada.
```

- Esto nos crea 2 tablas



- Exportar la base de datos wine de mysql a Python para unificar los datos

```
C:\Users\USER>C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqldump.exe -u root -p wine > wine_database_dump.sql
Enter password: ****
C:\Users\USER>
```

```
#IMPORTAR LA DATABASE EN MYSQL A PYTHON

import mysql.connector
import pandas as pd
from mysql.connector import Error

# conexión a MySQL |
DB_CONFIG = {
    'host': '127.0.0.1',
    'database': 'wine',
    'user': 'root',
    'password': '1212'
}

def export_mysql_to_pandas(table_name):
    conn = None
    df = None
    try:
        # Conectar a la base de datos MySQL
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor()

        query = f"SELECT * FROM {table_name}"
        print(f"Ejecutando consulta: {query}")

        df = pd.read_sql_query(query, conn)

        print(f"Datos de la tabla '{table_name}' cargados exitosamente en un DataFrame. Filas: {len(df)}")
        return df

    except Error as e:
        print(f"Error al conectar o ejecutar la consulta en MySQL: {e}")
        return None
    except Exception as e:
        print(f"Ocurrió un error inesperado: {e}")
        return None
    finally:
        if conn and conn.is_connected():
            cursor.close()
            conn.close()
            print("Conexión a MySQL cerrada.")

if __name__ == "__main__":
    print("--- Importando datos de wine_reviews_json a Python ---")
    df_json = export_mysql_to_pandas('wine_reviews_json')
    if df_json is not None:
        print(f"Primeras 5 filas del DataFrame de JSON:")
        print(df_json.head())
        print(f"Información del DataFrame de JSON:")
        df_json.info()

    print("\n" + "=" * 50 + "\n")

    print("--- Importando datos de wine_reviews_csv a Python ---")
    df_csv = export_mysql_to_pandas('wine_reviews_csv')
    if df_csv is not None:
        print(f"Primeras 5 filas del DataFrame de CSV:")
        print(df_csv.head())
        print(f"Información del DataFrame de CSV:")
        df_csv.info()

    --- Importando datos de wine_reviews_json a Python ---
    Ejecutando consulta: SELECT * FROM wine_reviews_json
    C:\Users\USER\AppData\Local\Temp\ipykernel_16208\3047483448.py:36: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or data
    base string URI or valid SQLAlchemy connection object. Other SQLAlchemy objects are not tested. Please consider using SQLAlchemy.
      df = pd.read_sql_query(query, conn) # <- pd.read_sql_query necesita la CADENA SQL 'query'
    Datos de la tabla 'wine_reviews_json' cargados exitosamente en un DataFrame. Filas: 259942
    Conexión a MySQL cerrada.
    Primeras 5 filas del DataFrame de JSON:
    0 1 87 Nicosia 2013 Vulkan Bianco (Etna)
    1 2 87 Quinte dos Avidagos 2011 Arlingas Red (Bours)
    2 3 87 Raintree 2013 Pinot Gris (Williams Valley)
    3 4 87 St. Julien 2013 Reserve Late Harvest Riesling ...
    4 5 87 Sweet Cheeks 2012 Vintner's Reserve Wild Child...
```

- Unificación de Datos

```
# UNIFICACION DE DATOS

import pandas as pd

print("Iniciando la unificación de DataFrames...")

df_unified = pd.concat([df_json, df_csv], ignore_index=True)

print("Unificación completada.")
print(f"Número total de reseñas unificadas: {len(df_unified)} filas.")

# verificar
print("\nPrimeras 5 filas del DataFrame unificado:")
print(df_unified.head())

print("\nÚltimas 5 filas del DataFrame unificado:")
print(df_unified.tail())

print("\nInformación del DataFrame unificado:")
df_unified.info()

print("\nColumnas del DataFrame unificado:")
print(df_unified.columns)
```

Tecnología Superior en Desarrollo de Software

```
Iniciando la unificación de DataFrames...
Unificación completada.
Número total de reseñas unificadas: 410867 filas.
```

Primeras 5 filas del DataFrame unificado:

	id	points			title		
0	1	87		Nicosia	2013 Vulkà Bianco	(Etna)	
1	2	87		Quinta dos Avidagos	2011 Avidagos Red	(Douro)	
2	3	87		Rainstorm	2013 Pinot Gris	(Willamette Valley)	
3	4	87		St. Julian	2013 Reserve Late Harvest Riesling		
4	5	87		Sweet Cheeks	2012 Vintner's Reserve Wild Child		

			description	taster_name	
0			Aromas include tropical fruit, broom, brimston...	Kerin O'Keefe	
1			This is ripe and fruity, a wine that is smooth...	Roger Voss	
2			Tart and snappy, the flavors of lime flesh and...	Paul Gregutt	
3			Pineapple rind, lemon pith and orange blossom ...	Alexander Peartree	
4			Much like the regular bottling from 2012, this...	Paul Gregutt	

	taster_twitter_handle	price		designation	
0	@kerinokeefe	NaN		Vulkà Bianco	
1	@vossroger	15.0		Avidagos	
2	@paulgwine	14.0		None	
3	None	13.0		Reserve Late Harvest	
4	@paulgwine	65.0		Vintner's Reserve Wild Child Block	

	variety	region_1	region_2	province	
0	White Blend	Etna	None	Sicily & Sardinia	
1	Portuguese Red	None	None	Douro	
2	Pinot Gris	Willamette Valley	Willamette Valley	Oregon	
3	Riesling	Lake Michigan Shore	None	Michigan	
4	Pinot Noir	Willamette Valley	Willamette Valley	Oregon	

	country	winery
0	Italy	Nicosia
1	Portugal	Quinta dos Avidagos
2	US	Rainstorm
3	US	St. Julian
4	US	Sweet Cheeks

• Analisis Sentimental

```
##Análisis de sentimiento

pip install vaderSentiment

Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py3-none-any.whl.metadata (572 bytes)
Requirement already satisfied: requests in c:\python313\lib\site-packages (from vaderSentiment) (2.32.4)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\python313\lib\site-packages (from requests->vaderSentiment) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in c:\python313\lib\site-packages (from requests->vaderSentiment) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\python313\lib\site-packages (from requests->vaderSentiment) (2.5.0)
Requirement already satisfied: certifi<=2017.4.17 in c:\python313\lib\site-packages (from requests->vaderSentiment) (2025.6.15)
Downloading vaderSentiment-3.3.2-py3-none-any.whl (125 kB)
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
Note: you may need to restart the kernel to use updated packages.

import pandas as pd
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

print("Empezando análisis de sentimiento")
# Crear la instancia del analizador
analyzer = SentimentIntensityAnalyzer()

# Función para obtener la puntuación de sentimiento
def get_sentiment_score(text):
    if pd.isna(text):
        return 0.0
    vs = analyzer.polarity_scores(text)
    return vs['compound']

# Aplicar la función a la columna 'description' del DataFrame
# columna 'sentiment_score'
df_unified['sentiment_score'] = df_unified['description'].apply(get_sentiment_score)

print("Análisis de sentimiento finalizado")

# resultado
# cerca de 1: Muy positivo ::::: cerca de -1: Muy negativo ::::: cerca de 0: Neutral

# reseñas positivas
print("\nReseñas más positivas (top 12):")
top_positive = df_unified.sort_values(by='sentiment_score', ascending=False).head(12)
print(top_positive[['description', 'sentiment_score']])

# reseñas negativas
print("\nReseñas más negativas (top 12):")
top_negative = df_unified.sort_values(by='sentiment_score', ascending=True).head(12)
print(top_negative[['description', 'sentiment_score']])

# estadística del análisis sentimental
print("\nEstadísticas del sentimiento general:")
print(df_unified['sentiment_score'].describe())

Iniciando el análisis de sentimiento...
Análisis de sentimiento completado.

Reseñas más positivas (top 5):
```

	description	sentiment_score
123946	There is not just a notion of ripeness but alm...	0.9937
325952	This is a splendid wine. Yes, it is more power...	0.9931
375502	This may be the most expensive Malbec in Calif...	0.9926
323147	This brilliant PG has just a bit of older Fren...	0.9917
68665	A touch of flinty reduction clings to the nose...	0.9912

```
Reseñas más negativas (top 5):
```

	description	sentiment_score
82547	The vintage was terribly difficult, with cold ...	-0.9288
368615	Disappointing considering the price. There's n...	-0.9202
101307	This is an underripe stench bomb with pyrazine...	-0.9153
306548	Hard, leathery aromas, with an intense but excc...	-0.9062
29581	This is a wine the winery has lavished some ex...	-0.9011

```
Estadísticas del sentimiento general:
count    169652.000000
mean         0.525969
std         0.371916
min        -0.928800
25%         0.318200
50%         0.642800
75%         0.827100
max         0.993700
Name: sentiment_score, dtype: float64
```

• Guardamos como dataframe

```
import pandas as pd

print("Guardar el dataframe a csv")

output_file = 'unified_wine_reviews_limpio.csv'

df_unified.to_csv(output_file, index=False, encoding='utf-8')

print(f"DataFrame guardado exitosamente en '{output_file}'.")

Guardar el dataframe a csv
DataFrame guardado exitosamente en 'unified_wine_reviews_limpio.csv'.
```


CouchDB (Chico)


1. Instalar e importar dependencias


```
[1]: pip install pandas
Requirement already satisfied: pandas in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (2.2.3)
Requirement already satisfied: numpy<=1.26.0 in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2.2.5)
Requirement already satisfied: python-dateutil<=2.8.2 in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz<=2020.1 in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil<=2.8.2->pandas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.


[2]: pip install couchdb
Requirement already satisfied: couchdb in c:\users\danieles\appdata\local\programs\python\python313\lib\site-packages (1.2)
Note: you may need to restart the kernel to use updated packages.
```

- Usamos la Pagina web de Kaggle para descargar los datasets

 [bbc_news](#)

 [datos_exportados](#)

 [EventosClima](#)

 [fighter_stats](#)

2. Verificamos y limpiamos

- [En figther_stats](#)
- Primero verificamos si los valores nulos que hay en la tabla, aquí hay valores faltantes, tanto numéricos como de text

```
df_csv.isna().sum()
```

name	1
wins	1
losses	1
height	1
weight	1
reach	656
stance	78
age	161
SLpM	1
sig_str_acc	1
SAPM	1
str_def	1
td_avg	1
td_acc	1
td_def	1
sub_avg	1
dtype:	int64

- Corregimos los errores

```
# Rellenar valores de texto con valores estandar
df_csv['name'] = df_csv['name'].fillna('Unknown')
df_csv['stance'] = df_csv['stance'].fillna(df_csv['stance'].mode()[0])

# Rellenar valores numericos con la media de su columna
num_cols = ['wins', 'losses', 'height', 'weight', 'reach', 'age',
            'SLpM', 'sig_str_acc', 'SApM', 'str_def',
            'td_avg', 'td_acc', 'td_def', 'sub_avg']

for col in num_cols:
    df_csv[col] = df_csv[col].fillna(df_csv[col].mean())
```


- Volvemos a verificar, y vemos que ya no hay valores nulos

```
#revisamos por ultima vez
print(df_csv.isnull().sum())
```

```
name          0
wins          0
losses        0
height        0
weight        0
reach         0
stance        0
age           0
SLpM          0
sig_str_acc   0
SApM          0
str_def       0
td_avg        0
td_acc        0
td_def        0
sub_avg       0
dtype: int64
```

- Luego de que estos ya estén limpios los exportamos, como json

```
df_csv.to_json("Datos_UFC_Limpios.json", orient="records", indent=3)
```

 Datos_UFC_Limpios.json

- Realizamos conexión a la base en la cual queremos trabajar.

```
#Pasar una base de datos no relacional
```

```
#nos conectamos con nuestras credenciales ya antes
```

```
import couchdb

# Reemplaza con tus datos
user = 'danny'
password = '12345'
url = f'http://{user}:{password}@127.0.0.1:5984/'

# Conexión
couch = couchdb.Server(url)
```

```
#creamos una base de datos
```

```
db_name = 'datos_limpios_todo'

if db_name in couch:
    db = couch[db_name]
else:
    db = couch.create(db_name)
```

```
#subimos los dicumentos
```

```
# Subir CSV
for _, row in df_csv.iterrows():
    doc = row.to_dict()
    db.save(doc)
```

- Repetimos esto para los demás csv, corregimos errores.
- Convertimos en JSONS

```
df_csv.to_json("Datos_BBC_Limpios.json", orient="records", indent=3)
```

```
##pasamos el archivo de csv a JSON para q mas facilidad en la bae de datos
df_csv.to_json("Datos_Clima_Limpios.json", orient="records", indent=3)
```

- Exportamos a COUCHDb

```
import couchdb

# Reemplaza con tus datos
user = 'danny'
password = '12345'
url = f'http://{user}:{password}@127.0.0.1:5984/'

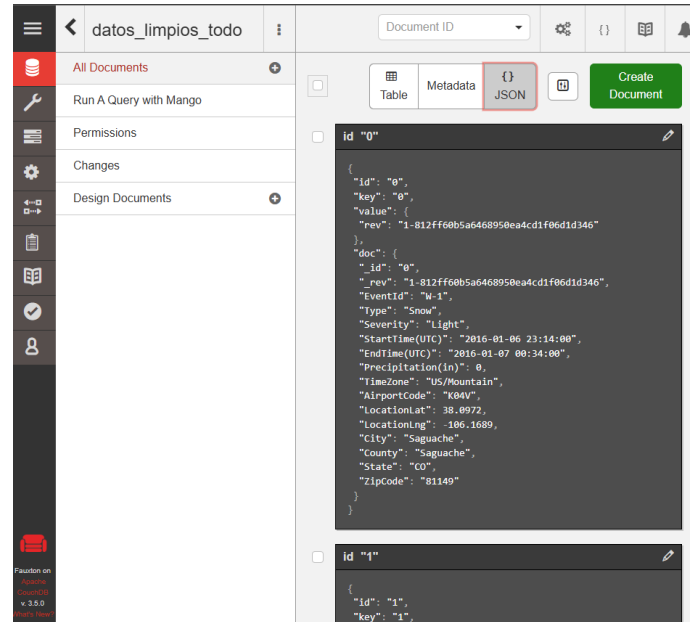
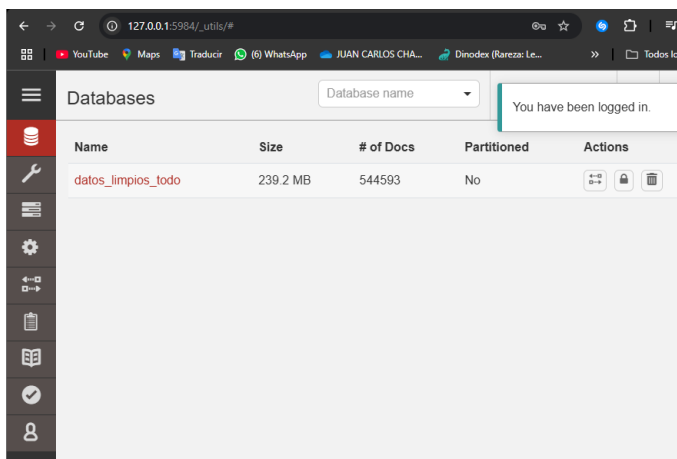
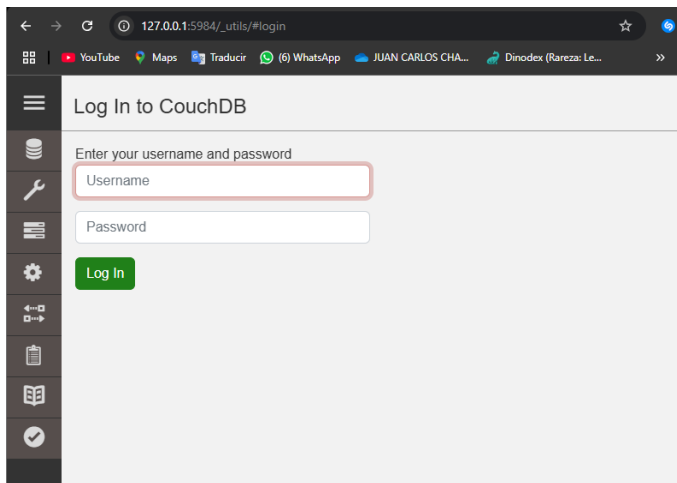
# Conexión
couch = couchdb.Server(url)

db_name = 'datos_limpios_todo'

# Conexión a la base ya existente (sin crearla de nuevo)
db = couch[db_name]

# Subir CSV
for _, row in df_csv.iterrows():
    doc = row.to_dict()
    db.save(doc)
```

- Visualizamos CouchDB
- Entramos con nuestras credenciales que definimos al momento de instalar



MONGO DB (Barba)

1. Cargar datasets

<https://catalog.data.gov/dataset/parks-special-events> (eventos)
<https://catalog.data.gov/dataset/?tags=concerts> (eventos)
<https://www.kaggle.com/datasets/anthonytherrien/diverse-activities-dataset> (hobbies)

2. Se procedio a importar a Python el csv, limpiarlo y guardarlo en un csv aparte

```
# 1. Importar librerías
import pandas as pd

# 2. Leer archivo CSV
df = pd.read_csv('Parks_Special_Events.csv')

# 3. Mostrar las primeras filas
display(df.head())

# 4. Ver información general del DataFrame
display(df.info())

# 5. Verificar valores nulos por columna
display(df.isnull().sum())

# 6. Eliminar filas duplicadas
df = df.drop_duplicates()

# 7. Rellenar valores nulos
df = df.fillna('Sin datos')

# 8. Mostrar resumen estadístico de columnas numéricas
display(df.describe())

# 9. Ver las primeras filas después de limpiar
print("Datos limpios:")
display(df.head())

# 10. Guardar el resultado en un nuevo archivo
df.to_csv('Parks_Special_Events_LIMPIO.csv', index=False)
```

Datos limpios:

	Unit	Group Name/Partner	Date and Time	Borough	LocationType	Location	Event Name	Event Type	Category	Classification	Attendance	
0	Urban Park Rangers	Urban Park Rangers	10/12/2019 11:00:00 AM	Queens	Adventure Course	Alley Road Adventure Course	PEP Family and Friends	Agency Produced Event	Nature	Sin datos	75.0	Chi Adult#Adult
1	Urban Park Rangers	Greenbelt Conservancy	11/20/2019 06:00:00 PM	Staten Island	Nature Center	Greenbelt Nature Center	Family Science Night	Local Event	Nature	Sin datos	30.0	G
2	Urban Park Rangers	Greenbelt Nature Center	12/19/2019 06:00:00 PM	Staten Island	Nature Center	Greenbelt Nature Center	Family science night	Community Based Event	Academic/Out of School time	Sin datos	9.0	Children#
3	Urban Park Rangers	Greenbelt	01/15/2020 06:00:00 PM	Staten Island	Nature Center	Greenbelt Nature Center	Family Science Night	Agency Produced Event	Academic/Out of School time	Sin datos	30.0	Children#
4	Urban Park Rangers	Greenbelt Nature Center	01/19/2020 11:00:00 AM	Staten Island	Nature Center	Greenbelt Nature Center	Winter Wilderness Skills	Local Event	Nature	Sin datos	62.0	Chi Adult#

- Inmediatamente se subió a mongo atlas mediante la conexión en jupyter notebook

```
import pandas as pd
from pymongo import MongoClient

# 1. Leer el archivo CSV
df = pd.read_csv('Parks_Special_Events_LIMPIO.csv')
display(df.head())

# URL de conexión
uri = "mongodb+srv://alexmteo2753:contrasena123@cluster0.dcccbj0.mongodb.net/actividad?retryWrites=true&w=majority&appName=Cluster"

# Conectar con MongoDB
cliente = MongoClient(uri)

# Seleccionar base de datos y colección
db = cliente['Proyecto_Analisis']
coleccion = db['Eventos_Publicos_1']

# Convertir DataFrame a diccionarios
datos = df.to_dict(orient='records')

# Insertar en la colección
coleccion.insert_many(datos)

print(f"Se subieron {len(datos)} documentos a MongoDB Atlas")
```

Proyecto_Analisis.Eventos_Publicos_1

STORAGE SIZE: 3.5MB LOGICAL DATA SIZE: 7.78MB TOTAL DOCUMENTS: 19702 INDEXES TOTAL SIZE: 848KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

[Generate queries from natural language in Compass](#)

[INSERT DOCUMENT](#)

Filter [Type a query: { field: 'value' }](#) [Reset](#) [Apply](#) [Options](#)

QUERY RESULTS: 1-20 OF MANY

```
{
  "_id": ObjectId("688fcb42f7e92a00309114f6"),
  "Unit": "Urban Park Rangers",
  "Group Name/Partner": "Greenbelt Conservancy",
  "Date and Time": "11/20/2019 06:00:00 PM",
  "Borough": "Staten Island",
  "LocationType": "Nature Center",
  "Location": "Greenbelt Nature Center",
  "Event Name": "Family Science Night",
  "Event Type": "Local Event",
  "Category": "Nature",
  "Classification": "Sin datos",
  "Attendance": "30.0",
  "Audience": "General Public",
  "Source": "Sharepoint"
}
```

[PREVIOUS](#) 1-20 of many results [NEXT](#)

- Un segundo csv descargado se lo limpio y se guardo como otro archivo csv

```
# 1. Importar librerías
import pandas as pd

# 2. Leer archivo CSV
df = pd.read_csv('Public_Events_at_the_Empire_State_Plaza.csv')

# 3. Mostrar las primeras filas
display(df.head())

# 4. Ver información general del DataFrame
display(df.info())

# 5. Verificar valores nulos por columna
display(df.isnull().sum())

# Eliminar una columnas vacías
df = df.drop('Event ID', axis=1)

# 6. Eliminar filas duplicadas
df = df.drop_duplicates()

# 7. Rellenar valores nulos
df = df.fillna('Sin datos')

# 8. Mostrar resumen estadístico de columnas numéricas
display(df.describe())

# 9. Ver las primeras filas después de limpiar
print("Datos limpios:")
display(df.head())

# 10. Guardar el resultado en un nuevo archivo
df.to_csv('Public_Events_at_the_Empire_State_Plaza_LIMPIO.csv', index=False)
```

- Se le convirtió el csv a db

```
import pandas as pd
import sqlite3

# 1. Leer el CSV
df = pd.read_csv('Public_Events_at_the_Empire_State_Plaza_LIMPIO.csv')

# 2. Crear conexión SQLite
conn = sqlite3.connect('Public_Events_at_the_Empire_State_Plaza.db')

# 3. Subir el DataFrame como una tabla
df.to_sql('datos', conn, if_exists='replace', index=False)

# 4. Cerrar conexión
conn.close()

print("CSV convertido a archivo SQL con éxito.")

CSV convertido a archivo SQL con éxito.
```

- Se leyó el db

```
import sqlite3
import pandas as pd

# Conectar a tu archivo SQLite
conn = sqlite3.connect('Public_Events_at_the_Empire_State_Plaza.db')

# Leer una tabla (ajusta el nombre si es diferente)
df = pd.read_sql_query("SELECT * FROM datos", conn)

conn.close()

# Ver las primeras filas
print("Datos de la tabla:")
display(df.head())
```

	Event Date	Start Time	End Time	Event Name	Event Description	Event Location	Admission
0	01/03/2023	5:00 P.M.	6:00 P.M.	Fitness at the Plaza	Free and open to everyone regardless of age, f...	Cornerstone at the Plaza	FREE
1	01/04/2023	10:00 A.M.	2:00 P.M.	Farmers Market	Empire State Plaza Farmers Market, indoors on ...	Empire State Plaza, Indoors	FREE
2	01/05/2023	5:00 P.M.	6:00 P.M.	Fitness at the Plaza	Free and open to everyone regardless of age, f...	Cornerstone at the Plaza	FREE
3	01/07/2023	9:00 A.M.	12:00 P.M.	Learn to Skate Clinic	Sin datos	Empire State Plaza Ice Rink	FREE
4	01/10/2023	5:00 P.M.	6:00 P.M.	Fitness at the Plaza	Free and open to everyone regardless of age, f...	Cornerstone at the Plaza	FREE

- Y se subió a mongo atlas

```
from pymongo import MongoClient

url = "mongodbsrv://alemateo2753:contrasena123@cluster0.4dcbbj0.mongodb.net/actividad?retryWrites=true&majorityAvailability=Cluster0"
cliente = MongoClient(url)

# Seleccionar base de datos y colección
db = cliente['Proyecto_Analisis']
coleccion = db['Eventos_Publicos_2']

# Convertir DataFrame en una lista de diccionarios
documentos = df.to_dict(orient='records')

# Insertar todos los registros en la colección
coleccion.insert_many(documentos)

print(f"Se subieron {len(documentos)} documentos desde SQLite a MongoDB Atlas.")

Se subieron 11750 documentos desde SQLite a MongoDB Atlas.
```

Proyecto_Analisis.Eventos_Publicos_2

STORAGE SIZE: 2.23MB LOGICAL DATA SIZE: 4.46MB TOTAL DOCUMENTS: 11750 INDEXES TOTAL SIZE: 512KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' }

Reset

Apply

Options

QUERY RESULTS: 1-20 OF MANY

```
{
  "_id": ObjectId('688fce8ff7e92a00309161ee'),
  "Event Date": "01/05/2023",
  "Start Time": "5:00 P.M.",
  "End Time": "6:00 P.M.",
  "Event Name": "Fitness at the Plaza",
  "Event Description": "Free and open to everyone regardless of age, fitness level, or experience.",
  "Event Location": "Cornerstone at the Plaza",
  "Admission": "FREE"
}
```

```
{
  "_id": ObjectId('688fce8ff7e92a00309161ef'),
  "Event Date": "01/07/2023",
  "Start Time": "9:00 A.M."
}
```

PREVIOUS

1-20 of many results

NEXT

- El archivo json de hobbies estaba mal estructurado, se implemento un código para adecuarlo y guardarlo como un nuevo archivo

```
# Abrimos el archivo mal estructurado
with open('activity.json', 'r', encoding='utf-8') as f:
    lineas = f.readlines()

# Eliminamos saltos de línea y espacios innecesarios
lineas_limpias = [linea.strip() for linea in lineas if linea.strip()]

# Unimos las líneas como elementos de una lista JSON válida
json_valido = '[' + '\n'.join(lineas_limpias) + '\n']

# Guardamos el archivo reparado
with open('activity_reparado.json', 'w', encoding='utf-8') as f:
    f.write(json_valido)

print("Creado 'activity_reparado.json' con formato válido.")

Creado 'activity_reparado.json' con formato válido.
```

- Limpiamos datos y guardamos en otro archivo

```
import pandas as pd

# 1. Cargar el archivo JSON reparado
df = pd.read_json('activity_reparado.json')

# 2. Verificar estructura general
print("Columnas y tipos:")
print(df.dtypes)

# 3. Verificar si hay valores nulos (NaN)
print("\nValores nulos por columna:")
print(df.isnull().sum())

# 4. Eliminar filas con valores nulos
df = df.dropna()

# 6. Verificar valores negativos en 'participants' o 'price'
df = df[(df['participants'] >= 1) & (df['price'] >= 0)]

# 7. Reiniciar índices
df = df.reset_index(drop=True)

# 8. Mostrar muestra final
print("\nDatos limpios:")
display(df.head())

# 9. Guardar el DataFrame limpio si deseas
df.to_json('activity_limpio.json', orient='records', indent=4)
print("\nDatos limpios guardados en 'activity_limpio.json'")
```

- Subimos a mongo atlas

```
from pymongo import MongoClient
import json

# 1. URL de conexión de MongoDB Atlas
url = "mongodbsrv://alemateo2753:contrasena123@cluster0.4dcbbj0.mongodb.net/actividad?retryWrites=true&majorityAvailability=Cluster0"

# 2. Conectar con MongoDB Atlas
cliente = MongoClient(url)

# 3. Seleccionar base de datos y colección
db = cliente['Proyecto_Analisis']
coleccion = db['Actividades_Hobbies']

# 4. Cargar el archivo JSON limpio
with open('activity_limpio.json', 'r', encoding='utf-8') as f:
    datos = json.load(f)

# 5. Subir los datos
if isinstance(datos, list):
    coleccion.insert_many(datos)
    print(f"Se subieron {len(datos)} documentos a MongoDB Atlas")
else:
    print("El archivo JSON no es una lista de documentos.")

Se subieron 32000 documentos a MongoDB Atlas
```

Proyecto_Analisis.Actividades_Hobbies

STORAGE SIZE: 1.98MB LOGICAL DATA SIZE: 3.68MB TOTAL DOCUMENTS: 32000 INDEXES TOTAL SIZE: 1.27MB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' }

Reset

Apply

Options

QUERY RESULTS: 1-20 OF MANY

```
{
  "_id": ObjectId('689117c423967afb991e0d6a'),
  "activity": "Meditate for five minutes",
  "type": "relaxation",
  "participants": 1,
  "price": 0
}
```

```
{
  "_id": ObjectId('689117c423967afb991e0d6b'),
  "activity": "Uninstall unused apps from your devices",
  "type": "busywork",
  "participants": 1,
  "price": 0
}
```

PREVIOUS

1-20 of many results

NEXT

IMPORTACION DE DATOS DE MONGO ATLAS → SQL SERVER → POWER BII

1. Se unio todos los datasets en un mongo local, total 10 datasets (10 colecciones), 1 millon de datos

- Se unió primero la parte del estudiante **MATEO BARBA**
- Se insalo pymongo

```
pip install pymongo
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pymongo in c:\users\dell\appdata\roaming\python\python312\site-packages (4.13.0)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pymongo) (2.7.0)
Note: you may need to restart the kernel to use updated packages.
```

```
from pymongo import MongoClient

# 1. Conexión a Atlas
uri_atlas = "mongodb+srv://alexmato2733:contrasena123@cluster0.mongodb.net/actividad?retryWrites=true&majorityOplogName=Cluster0"
cliente_atlas = MongoClient(uri_atlas)
db_atlas = cliente_atlas['Proyecto_Analisis']

# 2. Conexión a Mongo Local
cliente_local = MongoClient("mongodb://localhost:27017/")
db_local = cliente_local['PROYECTO_ANALISIS_UNIDO']

# 3. Obtener todas las colecciones del origen
colecciones = db_atlas.list_collection_names()
print("Colecciones encontradas:", colecciones)

# 4. Elavar cada colección
for nombre_col in colecciones:
    datos = list(db_atlas[nombre_col].find())

    # Filtrar el _id para evitar conflictos de duplicación
    for doc in datos:
        doc.pop('_id', None)

    # Insertar en Mongo Local
    if datos:
        db_local[nombre_col].insert_many(datos)
        print(f"Copiada colección '{nombre_col}' con {len(datos)} documentos.")
    else:
        print(f"Colección '{nombre_col}' está vacía.")

print("¡Todas las colecciones fueron copiadas de Atlas a tu Mongo local!")

Colecciones encontradas: ['Eventos_Publicos_2', 'Actividades_Hobbies', 'Eventos_Publicos_1']
Copiada colección 'Eventos_Publicos_2' con 11750 documentos.
Copiada colección 'Actividades_Hobbies' con 32000 documentos.
Copiada colección 'Eventos_Publicos_1' con 19702 documentos.
¡Todas las colecciones fueron copiadas de Atlas a tu Mongo local!
```

- **DANIEL DIAZ**

```
import json
from pymongo import MongoClient

# 1. Cargar archivo JSON
with open("Datos_Clima_Limpios.json", "r", encoding="utf-8") as file:
    datos = json.load(file)

# 2. Conectarse a MongoDB Local
cliente = MongoClient("mongodb://localhost:27017/")
db = cliente["PROYECTO_ANALISIS_UNIDO"]
coleccion = db["Eventos_Noticias_Mundiales_1"]

# 3. Insertar Los datos
if isinstance(datos, list):
    coleccion.insert_many(datos)
else:
    coleccion.insert_one(datos)

print("Datos insertados correctamente en MongoDB local desde JSON.")
```

Datos insertados correctamente en MongoDB local desde JSON.

```
import json
from pymongo import MongoClient

# 1. Cargar archivo JSON
with open("Datos_UFC_Limpios.json", "r", encoding="utf-8") as file:
    datos = json.load(file)

# 2. Conectarse a MongoDB Local
cliente = MongoClient("mongodb://localhost:27017/")
db = cliente["PROYECTO_ANALISIS_UNIDO"]
coleccion = db["Eventos_Noticias_Mundiales_2"]

# 3. Insertar Los datos
if isinstance(datos, list):
    coleccion.insert_many(datos)
else:
    coleccion.insert_one(datos)

print("Datos insertados correctamente en MongoDB local desde JSON.")
```

Datos insertados correctamente en MongoDB local desde JSON.

- **MERCY PERUGACHI**

```
import pandas as pd
from pymongo import MongoClient

# 1. Leer el archivo CSV
df = pd.read_csv("unifed_wine_reviews_limpio.csv")

# 2. Conexión a MongoDB Local
client = MongoClient("mongodb://localhost:27017/")
db = client["PROYECTO_ANALISIS_UNIDO"]
coleccion = db["Definido_Grupo_Vino_Reviews"]

# 3. Convertir DataFrame a diccionarios
datos = df.to_dict(orient="records")

# 4. Insertar Los datos en MongoDB
coleccion.insert_many(datos)

print("Datos insertados correctamente en MongoDB local")
```

Datos insertados correctamente en MongoDB local

- **IVAN PANCHI**

```

: #EVENTOS DEPORTIVOS

: import pandas as pd
  from pymongo import MongoClient

  # 1. Leer el archivo CSV
  df = pd.read_csv("data.csv")

  # 2. Conexión a MongoDB Local
  client = MongoClient("mongodb://localhost:27017/")
  db = client["PROYECTO_ANALISIS_UNIDO"]
  coleccion = db["Eventos_Deportivos_1"]

  # 3. Convertir DataFrame a diccionarios
  datos = df.to_dict(orient="records")

  # 4. Insertar Los datos en MongoDB
  coleccion.insert_many(datos)

  print("Datos insertados correctamente en MongoDB local")

Datos insertados correctamente en MongoDB local

: import pandas as pd
  from pymongo import MongoClient

  # 1. Leer el archivo CSV
  df = pd.read_csv("Olympic_Athlete_Event_Details_CLEAN.csv")

  # 2. Conexión a MongoDB Local
  client = MongoClient("mongodb://localhost:27017/")
  db = client["PROYECTO_ANALISIS_UNIDO"]
  coleccion = db["Eventos_Deportivos_2"]

  # 3. Convertir DataFrame a diccionarios
  datos = df.to_dict(orient="records")

  # 4. Insertar Los datos en MongoDB
  coleccion.insert_many(datos)

  print("Datos insertados correctamente en MongoDB local")

Datos insertados correctamente en MongoDB local

```

```

: import pandas as pd
  from pymongo import MongoClient

  # 1. Leer el archivo CSV
  df = pd.read_csv("gym_members_analysis.csv")

  # 2. Conexión a MongoDB Local
  client = MongoClient("mongodb://localhost:27017/")
  db = client["PROYECTO_ANALISIS_UNIDO"]
  coleccion = db["Actividades_Hobbies_2"]

  # 3. Convertir DataFrame a diccionarios
  datos = df.to_dict(orient="records")

  # 4. Insertar Los datos en MongoDB
  coleccion.insert_many(datos)

  print("Datos insertados correctamente en MongoDB local")

Datos insertados correctamente en MongoDB local

: import pandas as pd
  from pymongo import MongoClient

  # 1. Leer el archivo CSV
  df = pd.read_csv("Fatal_accidents_drivers.csv")

  # 2. Conexión a MongoDB Local
  client = MongoClient("mongodb://localhost:27017/")
  db = client["PROYECTO_ANALISIS_UNIDO"]
  coleccion = db["Eventos_Noticias_Mundiales_3"]

  # 3. Convertir DataFrame a diccionarios
  datos = df.to_dict(orient="records")

  # 4. Insertar Los datos en MongoDB
  coleccion.insert_many(datos)

  print("Datos insertados correctamente en MongoDB local")

Datos insertados correctamente en MongoDB local

```

- Se logro exportar con éxito todos los datasets



2. EXPORTACIÓN A SQL SERVER

```

pip install pymongo pyodbc pandas

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pymongo in c:\users\dell\appdata\roaming\python\python312\site-packages (4.13.0)
Requirement already satisfied: pyodbc in c:\users\dell\appdata\roaming\python\python312\site-packages (5.2.0)
Requirement already satisfied: pandas in c:\users\dell\appdata\roaming\python\python312\site-packages (2.2.3)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pymongo) (2.7.0)
Requirement already satisfied: numpy>=1.26.0 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pandas) (2.2.5)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\dell\appdata\roaming\python\python312\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.

```

- El código lee la base de datos en mongo, copia las colecciones que están y las pasa a sql server, se tiene que tener creada una base de datos antes de hacer este proceso

```

import pymongo
import pyodbc
import pandas as pd
import re

# 1. Conectar a MongoDB Local
cliente_mongo = pymongo.MongoClient("mongodb://localhost:27017/")
db_mongo = cliente_mongo['PROYECTO_ANALISIS_UNIDO']

# 2. Conectar a SQL Server
conn_str = (
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=localhost;"
    "DATABASE=PROYECTO_ANALISIS_UNIDO;"
    "Trusted_Connection=yes;"
)

conn = pyodbc.connect(conn_str)
cursor = conn.cursor()

# 3. Procesar todas las colecciones
for nombre_col in db_mongo.list_collection_names():
    print(f"Procesando colección: {nombre_col}")

    docs = list(db_mongo[nombre_col].find())
    for d in docs:
        d.pop('_id', None)

    if not docs:
        print(f"Colección '{nombre_col}' está vacía. Se omite.")
        continue

    # Convertir a DataFrame
    df = pd.DataFrame(docs)

    # 4. Limpiar nombres de columnas para SQL Server
    df.columns = [re.sub(r'\\W+', '_', col) for col in df.columns]

    # 5. Crear tabla SQL con columnas tipo VARCHAR(MAX)
    columnas_sql = ",,\n".join([f"{{col}} VARCHAR(MAX)" for col in df.columns])
    crear_tabla_sql = f"""
    IF OBJECT_ID('{nombre_col}', 'U') IS NULL
    CREATE TABLE [{nombre_col}] (
        {columnas_sql}
    )
    """
    cursor.execute(crear_tabla_sql)
    conn.commit()

    # 6. Insertar datos usando placeholders seguros
    placeholders = ",,\n".join(["?" for _ in df.columns])
    insert_sql = f"INSERT INTO [{nombre_col}] ({', '.join(df.columns)}) VALUES ({placeholders})"

    for _, fila in df.iterrows():
        valores = fila.fillna("").tolist()
        cursor.execute(insert_sql, valores)

conn.commit()
print(f"Insertados {len(df)} registros en tabla '{nombre_col}'")

# Cerrar conexión
cursor.close()
conn.close()
print(" Migración completa de MongoDB local a SQL Server.")

```

```

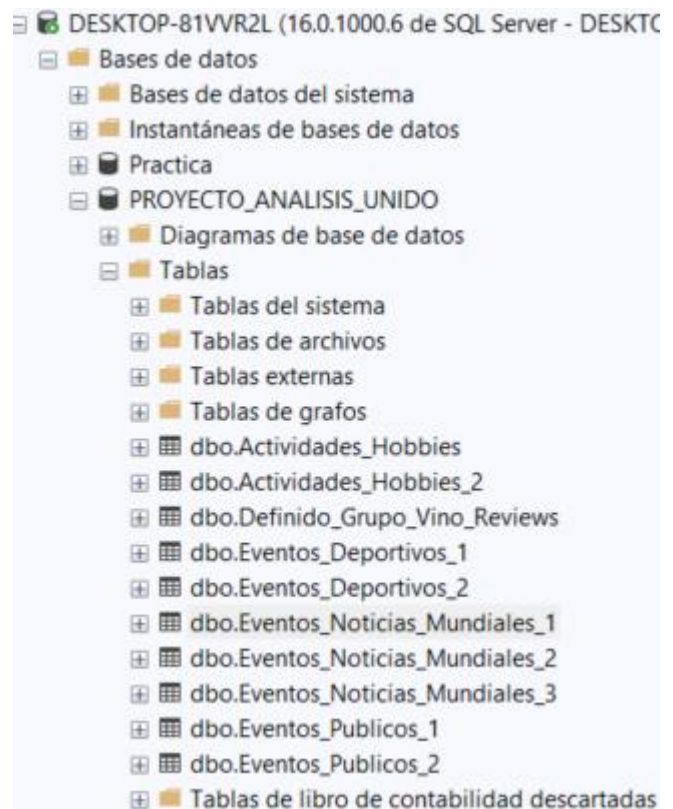
conn.commit()
print(f"Insertados {len(df)} registros en tabla '{nombre_col}'")

# Cerrar conexión
cursor.close()
conn.close()
print(" Migración completa de MongoDB local a SQL Server.")

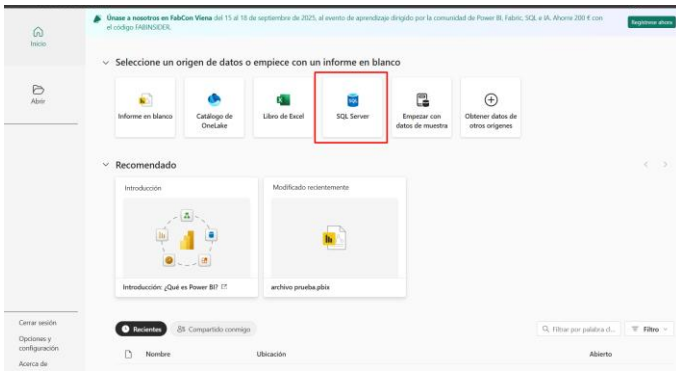
Procesando colección: Eventos_Deportivos_2
Insertados 316834 registros en tabla 'Eventos_Deportivos_2'.
Procesando colección: Eventos_Noticias_Mundiales_2
Insertados 2479 registros en tabla 'Eventos_Noticias_Mundiales_2'.
Procesando colección: Actividades_Hobbies_3
Insertados 9999 registros en tabla 'Actividades_Hobbies_3'.
Procesando colección: Definido_Grupo_Vino_Reviews
Insertados 169652 registros en tabla 'Definido_Grupo_Vino_Reviews'.
Procesando colección: Eventos_Publicos_1
Insertados 19702 registros en tabla 'Eventos_Publicos_1'.
Procesando colección: Eventos_Publicos_2
Insertados 11750 registros en tabla 'Eventos_Publicos_2'.
Procesando colección: Actividades_Hobbies_2
Insertados 973 registros en tabla 'Actividades_Hobbies_2'.
Procesando colección: Actividades_Hobbies
Insertados 32000 registros en tabla 'Actividades_Hobbies'.
Procesando colección: Eventos_Noticias_Mundiales_1
Insertados 499999 registros en tabla 'Eventos_Noticias_Mundiales_1'.
Procesando colección: Eventos_Deportivos_1

```

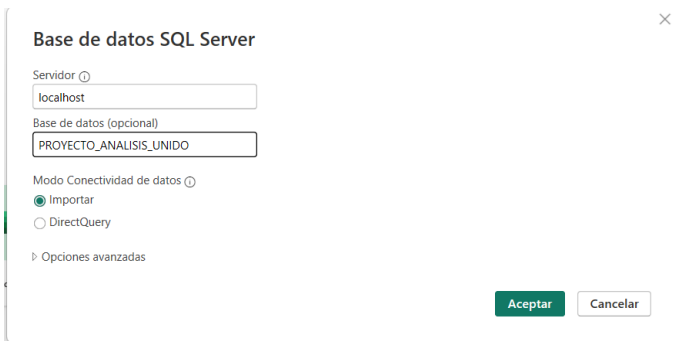
- Exportación con éxito a sql server



- Exportación a power bi
- En el programa de power bi se selecciono lo siguiente



- Localhos poque esta es host local del sql server
- Abajo la base de datos que habíamos creado



- Se exporto los datasets a power bi con exito

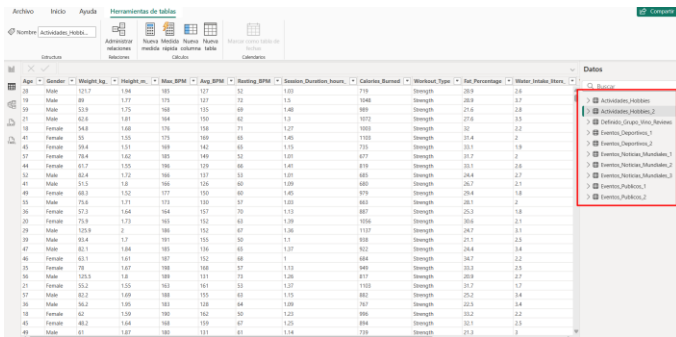


Gráfico: "Recuento de participants por type y type"

- **Recreational (recreativas)** son las más populares, seguidas por:
 - **Social**
 - **Education**
 - **Busywork**
- Las actividades de **music**, **diy** y **cooking** tienen la menor participación.

Interpretación: Las personas prefieren actividades recreativas y sociales, probablemente por ser más entretenidas y colaborativas.

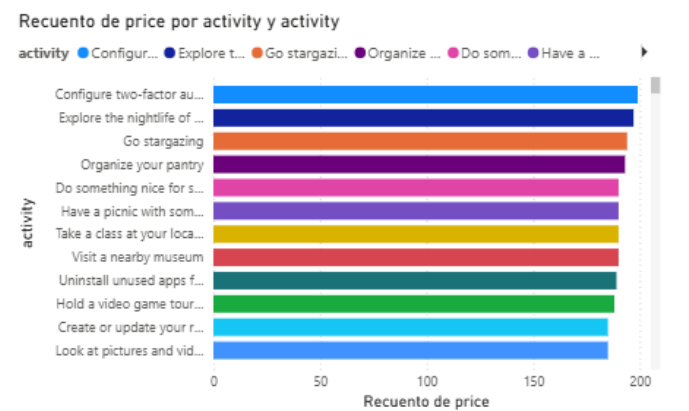


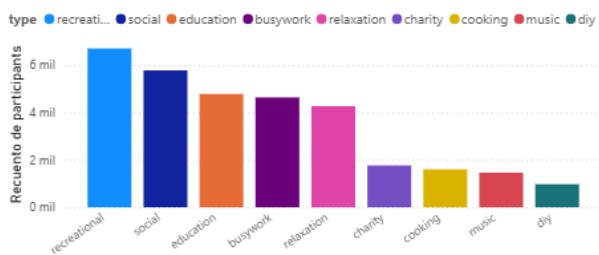
Gráfico: "Recuento de price por activity y activity"

- Las actividades con mayor recuento de "price" (costo) son:
 - "Configure two-factor auth"
 - "Explore the nightlife of..."
 - "Go stargazing"
- Muchas actividades tienen un precio similar alto (cerca de 200 unidades).

Interpretación: Hay actividades tecnológicas y sociales que implican un mayor costo, lo cual puede estar relacionado con requerimientos técnicos o accesibilidad.

ANALISIS DE LA INFORMACION

Recuento de participants por type y type



Recuento de price por type

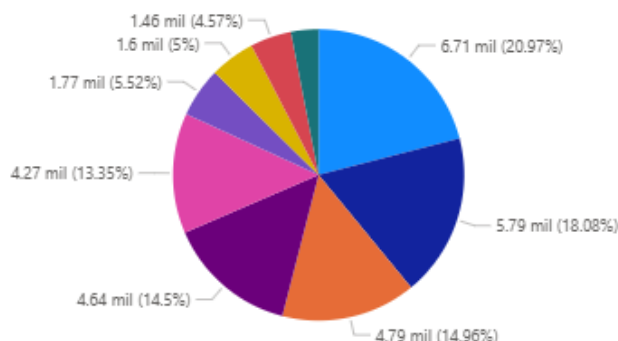


Gráfico: "Recuento de price por type" (Gráfico circular)

- Los tipos con mayor costo total son:
 - 1. Recreational (20.97%)**
 - 2. Social (18.08%)**
 - 3. Education (14.95%)**
 - 4. Busywork (14.55%)**
- Otros tipos como music, cooking y diy tienen muy bajo peso en el total de costos.

Interpretación: Las actividades recreativas y sociales no solo son populares, también son las que más recursos demandan.



Gráfico: Mapa "Recuento de activity por activity y participants"

- Las actividades están distribuidas en:
 - América del Norte
 - Europa
 - Asia
 - Sudamérica (en menor cantidad)
- Se observan más marcadores en América del Norte y Europa.

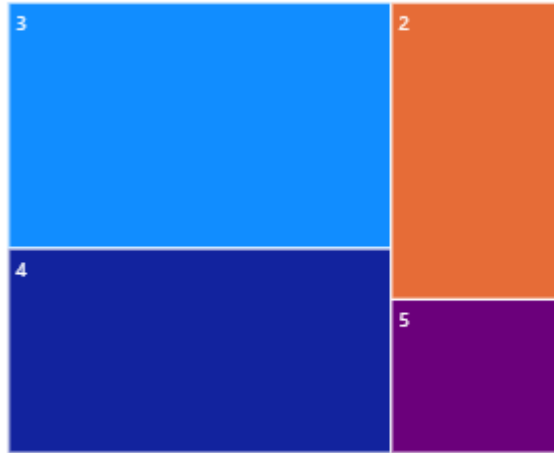
Interpretación: Las actividades analizadas están más concentradas en regiones desarrolladas, posiblemente por mayor disponibilidad de datos o acceso a tecnología.

RESULTADOS OBTENIDOS

- Al analizar el gráfico de recuento de participantes por tipo de actividad, se observa que las actividades recreativas presentan la mayor participación, seguidas por las sociales, educativas y de tipo "busywork". En contraste, las actividades de cocina, música y manualidades (DIY) tienen una participación considerablemente menor.
- En el gráfico de recuento de precio por actividad, se evidencia que varias actividades tienen un costo similar y elevado, especialmente aquellas relacionadas con tecnología y ocio, como "Configure two-factor auth", "Explore the nightlife of..." y "Go stargazing". Esto indica que ciertas actividades requieren mayores recursos económicos para realizarse.
- El gráfico circular de recuento de precio por tipo muestra que los tipos de actividad que concentran el mayor gasto total son las recreativas (20.97%), sociales (18.08%) y educativas (14.95%). En cambio, los tipos como cocina, música y DIY representan un porcentaje menor del total del precio acumulado.
- Finalmente, al observar el mapa de distribución geográfica de actividades por participantes, se aprecia que la mayor concentración de actividades se encuentra en América del Norte, Europa y Asia. En América del Sur y África también hay presencia, aunque en menor proporción.

RESULTADOS OBTENIDOS

Recuento de Gender por Workout_Frequency_days_week_

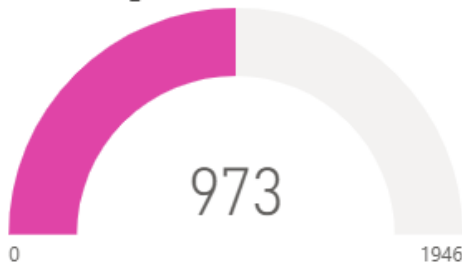


Análisis:

La mayoría de las personas entrenan entre 3 y 4 días a la semana. Hay una menor proporción de quienes entrenan solo 2 o hasta 5 días. La distribución sugiere una rutina estable de entrenamiento en la mayoría de los casos.

Resultado obtenido: La frecuencia más común de entrenamiento semanal es de 3 a 4 días, lo que sugiere un compromiso moderado con la actividad física.

Recuento de Max_BPM

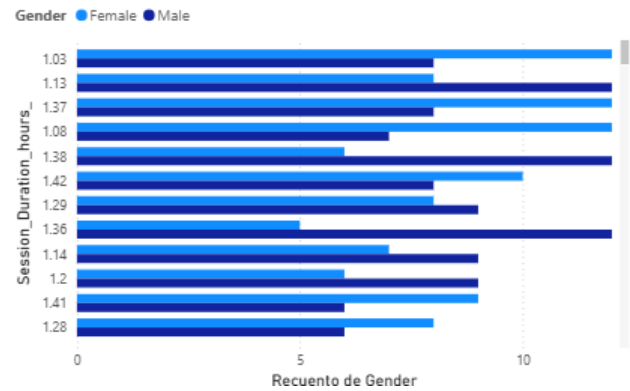


Análisis:

El indicador muestra un valor medio de **973 BPM**, lo que refleja la suma de todos los valores máximos registrados. Aunque no se detalla el valor por persona, permite una visión del esfuerzo cardiovascular general dentro del grupo analizado.

Resultado obtenido: El esfuerzo físico general, medido por BPM máximos, es moderadamente elevado, lo que indica un entrenamiento de intensidad media a alta.

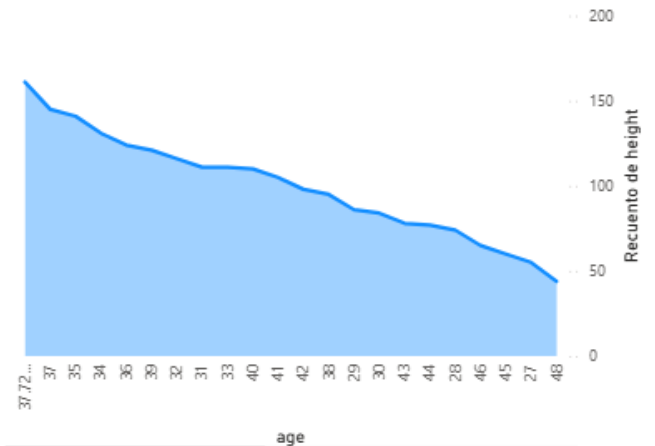
Recuento de Gender por Session_Duration_hours_ y Gender



Se observa que tanto hombres como mujeres mantienen sesiones de entrenamiento dentro de un rango promedio entre 1.03 y 1.42 horas. Sin embargo, los hombres tienden a tener una ligera mayor frecuencia en duraciones superiores a 1.32 horas, mientras que las mujeres presentan mayor representación en sesiones de menor duración.

Resultado obtenido: Los hombres dominan en sesiones más largas, mientras que las mujeres se concentran en sesiones intermedias o más cortas.

Recuento de height por age

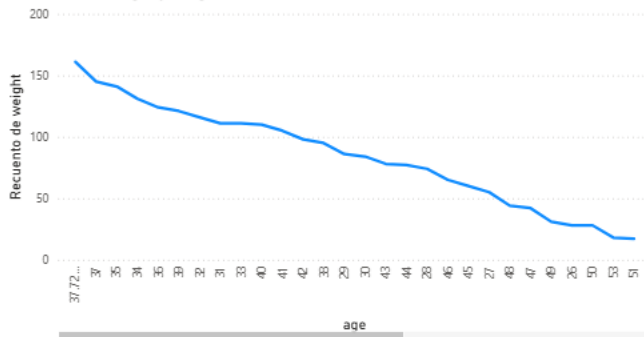


Análisis:

Se observa una disminución progresiva en el recuento de estatura a medida que aumenta la edad. Las edades más jóvenes presentan valores más altos, mientras que los mayores tienden a registrar menores alturas.

Resultado obtenido: A mayor edad, se reduce la estatura promedio registrada, lo cual puede deberse a factores físicos naturales como pérdida de masa ósea o encorvamiento.

Recuento de weight por age

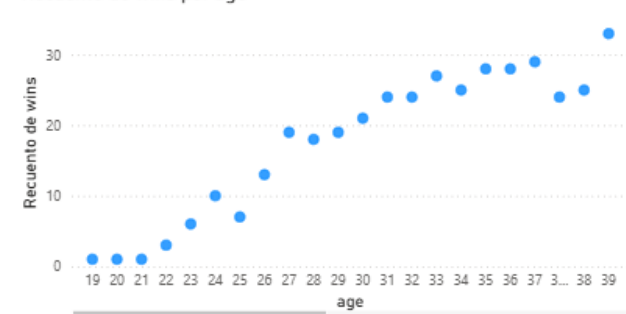


Análisis:

La tendencia del peso es descendente conforme aumenta la edad. Los valores más altos de peso se concentran en los rangos de edad más jóvenes (20 a 30 años), mientras que los mayores de 45 muestran pesos considerablemente más bajos.

Resultado obtenido: Al igual que la estatura, el peso tiende a disminuir con la edad, posiblemente por pérdida de masa muscular o cambios en el metabolismo.

Recuento de wins por age

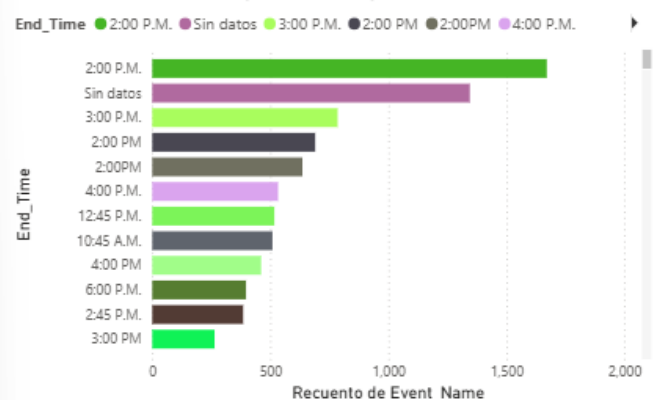


Análisis:

Las victorias aumentan notablemente entre los 22 y 32 años, alcanzando su punto más alto alrededor de los 34 años. A partir de los 36 años, las victorias tienden a estabilizarse o disminuir.

Resultado obtenido: El rendimiento (medido en número de victorias) es mayor en la etapa adulta temprana y media, lo que podría coincidir con el pico de capacidades físicas y experiencia acumulada.

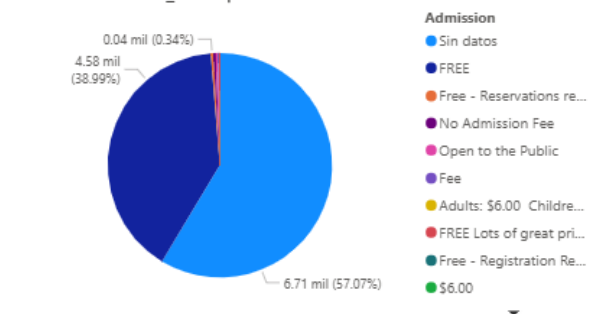
Recuento de Event_Name por End_Time y End_Time



Análisis: La mayor cantidad de eventos finaliza a las 2:00 P.M., disminuyendo notablemente en horas posteriores.

Resultado obtenido: Los eventos se concentran en la primera mitad de la tarde.

Recuento de Event_Name por Admission



Análisis: La mayoría de los eventos son gratuitos o sin costo registrado, mientras que los de pago son mínimos.

Resultado obtenido: La oferta de eventos es altamente accesible para el público.

Event_Location y End_Time



Análisis: Los eventos se concentran en Estados Unidos, sin presencia significativa en otros continentes.

Resultado obtenido: La cobertura de eventos es regional y limitada a Norteamérica.

Conclusiones

- **Integración de Bases de Datos Híbridas**
Este proyecto demuestra que es posible combinar bases de datos relacionales (SQL) y NoSQL en una misma infraestructura, permitiendo centralizar información de diferentes fuentes y formatos. Esto facilita su gestión y análisis, lo cual es fundamental en entornos empresariales complejos.
- **Interoperabilidad y Conectividad entre Sistemas**
El uso de MySQL como intermediario entre bases SQL y NoSQL resultó ser una estrategia eficiente para lograr interoperabilidad. Se resalta así la importancia de herramientas que permitan la transferencia fluida de datos en arquitecturas diversas.
- **Centralización y Análisis Avanzado con Power BI**
El desarrollo de un repositorio centralizado en SQL Server permitió estructurar los datos y analizarlos con herramientas como Power BI, facilitando la creación de dashboards interactivos y consultas temáticas que ofrecen una visión integral de la información.
- **Optimización del Flujo de Datos**
Se logró una transferencia de datos más eficiente entre bases relacionales y NoSQL, garantizando su integridad y consistencia. Esto redujo errores y redundancias, mejorando el rendimiento general del sistema.

Recomendaciones

1. **Monitoreo Continuo del Sistema**
Se recomienda implementar herramientas de monitoreo como *Prometheus* o *Grafana* para supervisar en tiempo real el rendimiento de la infraestructura. Esto permitirá detectar cuellos de botella, latencia o fallos, y asegurar un flujo de datos rápido y seguro entre SQL y NoSQL.
2. **Fortalecer la Seguridad del Sistema**
Debido a la naturaleza distribuida de las bases de datos, es esencial aplicar políticas de seguridad robustas. Se debe cifrar la información en tránsito y reposo (con tecnologías como SSL/TLS y AES),

e implementar control de acceso por roles (RBAC) para proteger la confidencialidad e integridad de los datos.

3. **Mantener Documentación Detallada**
Es importante contar con documentación actualizada sobre la arquitectura del sistema y los procesos de integración. Esto debe incluir diagramas, descripciones de conexiones entre bases de datos y procedimientos de consolidación, lo cual facilitará el mantenimiento y la incorporación de nuevos integrantes al equipo.
4. **Realizar Pruebas y Validaciones Regulares**
Para garantizar la calidad del sistema y de los datos, se deben realizar pruebas periódicas de integridad, carga y consistencia antes de tener un borrador ya listo. También se recomienda implementar pruebas automatizadas para detectar errores en la consolidación y análisis de datos.

Desafíos y Problemas Encontrados

Durante el desarrollo del proyecto surgieron varios retos que afectaron tanto el proceso de integración como el análisis de los datos. A continuación, se describen los principales:

1. **Dificultad en la Integración de Bases de Datos Diferentes**
Una de las mayores complicaciones fue unir bases de datos relacionales (como MySQL y PostgreSQL) con NoSQL (como MongoDB), ya que cada una maneja estructuras y formatos distintos, lo que dificultó la consolidación eficiente y coherente de los datos.
2. **Mantener la Consistencia e Integridad de los Datos**
Trabajar con fuentes de datos tan variadas hizo difícil asegurar que la información fuera siempre coherente y sin errores, especialmente al consolidarla en un repositorio común.
3. **Problemas de Rendimiento**
Cuando el volumen de datos empezó a crecer, notamos una baja en el rendimiento, sobre todo al transferir datos entre bases. Usar MySQL como intermediario fue útil pero también trajo retos que obligaron a optimizar consultas

y procesos para mantener tiempos de respuesta aceptables.

4. **Escalabilidad del Sistema**

A medida que se manejaba más información, fue necesario ajustar y reforzar la infraestructura para que pudiera seguir funcionando correctamente sin perder velocidad ni calidad en los análisis.

5. **Manejo de Errores y Recuperación ante Fallos**

Trabajar con varios sistemas aumentó las posibilidades de errores. Enfrentar estos fallos sin afectar la integridad de los datos fue un reto, y nos obligó a pensar en formas de recuperación rápida y segura.

6. **Capacitación del Equipo**

El uso de tecnologías variadas implicó que el equipo tuviera que aprender rápidamente a manejar nuevas herramientas como MongoDB, PostgreSQL y SQL Server. Esto tomó tiempo y en algunos casos retrasó el avance del proyecto, por lo que mantener una documentación clara y actualizada fue clave para facilitar el aprendizaje.

7. **Limitaciones de las Herramientas de Visualización**

También enfrentamos ciertas dificultades técnicas al integrar bases de datos NoSQL con Power BI, lo que limitó, en algunos casos, la forma en que podíamos visualizar los datos de manera eficiente y clara.

