

eda_fineturning

November 15, 2025

1 EDA Dataset Análisis de las Sociedades Argentinas

```
[1]: import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sqlalchemy import create_engine
from dotenv import load_dotenv
```

Creamos la conexión a la BD.

```
[2]: load_dotenv()

DB_USER = os.getenv("MYSQL_USER")
DB_PASSWORD = os.getenv("MYSQL_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_NAME = os.getenv("MYSQL_DATABASE")

DATABASE_URL = f"mysql+mysqlconnector://{DB_USER}:{DB_PASSWORD}@{DB_HOST}/
↳{DB_NAME}"

engine = create_engine(DATABASE_URL)
connection = engine.connect()
```

Leemos todos los avisos disponibles.

```
[3]: query = "SELECT * from avisos_fineturning;"
df = pd.read_sql(query, connection)
df.head()
```

```
[3]:   id  aviso_id                                     src \
0  27   A836918  Avisa su constitución: Escritura 78 del 09/05/...
1  28   A816562  CONSTITUCIÓN: 22/02/2019. 1.- DAMIAN FRANCISCO...
2  29  A1320548  Constituida: Por Escritura Pública número 220 ...
3  30   A9528   Escritura 36 el 20-7-2009. Esc. Susana Merlott...
```

dst

```

0 {"objeto": ["INMOBILIARIA: mediante la compra,...
1 {"objeto": ["Agropecuarias, avícolas, ganadera...
2 {"objeto": ["Servicio de transporte urbano, in...
3 {"objeto": ["Realizar por cuenta propia, de te...

```

```

[4]: print("Columnas: ", df.columns.to_list())
      print("# de avisos:", len(df))
      print("Tipos de Datos::\n", df.dtypes)
      print("\n# de Valores únicos:\n", df.nunique())

```

```

Columnas: ['id', 'aviso_id', 'src', 'dst']
# de avisos: 4
Tipos de Datos::
   id          int64
aviso_id  object
src       object
dst       object
dtype: object

```

```

# de Valores únicos:
   id          4
aviso_id      4
src           4
dst           4
dtype: int64

```

```

[5]: import json

      parsed_data = pd.json_normalize(df['dst'].apply(json.loads))

```

```

[6]: parsed_data.head(1)

```

```

[6]:                                objeto \
0  [INMOBILIARIA: mediante la compra, venta, admi...

                                socios \
0  [{'dni': '24.515.732', 'cuit_cuil': '20-245157...

                                duracion \
0  99 años desde su inscripción

                                directorio \
0  [{'cargo': 'DIRECTOR TITULAR Y PRESIDENTE', 'n...

                                sede_social suscripcion \
0  Teniente Benjamín Matienzo 1541, piso 6, de la...      None

                                fiscalizacion      nombre_sociedad fecha_instrumento \

```

```

0          None CIUDAD INVERSIONES S.A.          2019-05-09

  fecha_cierre_ejercicio ... capital_social.monto \
0          31/12 ...          200000

  capital_social.integracion capital_social.tipo_acciones \
0          None          None

  capital_social.votos_por_accion capital_social.cantidad_acciones \
0          None          NaN

  capital_social.plazo_integracion_saldo \
0          None

  capital_social.valor_nominal_por_accion publicacion_boletin.fecha \
0          NaN          2019-05-13

  publicacion_boletin.vencimiento publicacion_boletin.numero_aviso
0          2019-05-13          32079/19

[1 rows x 30 columns]

```

```
[8]: data = df['dst'].apply(json.loads)
```

```
[14]: data[2]
```

```
[14]: {'objeto': ['Servicio de transporte urbano, interurbano e internacional de:
mercaderías en general, cargas masivas o a granel de bienes homogéneos, cargas
peligrosas de sustancias o mercancías alimenticias, contenedores, paletizados, a
granel, incluyendo mercancías peligrosas, líquidos, petróleo crudo y sus
derivados consideradas como tal por la normativa vigente, cargas fraccionadas de
bienes compatibles que puedan ser consolidados en la misma bodega, acopiados en
uno o varios orígenes, de uno o varios dadores de carga, con uno o mas destinos
y con entregas completas o fraccionadas, caudales, cargas indivisibles
(ingeniería del transporte), correos, recolección de residuos, trabajos en la
vía pública y ganado mayor',
'Servicios de Gestión Logística y Distribución: Almacenamiento y
administración de stock de materia primas, cuarentenas, productos terminados y
devoluciones, desconsolidación de contenedores y paletización, picking,
preparación y acondicionamiento de empaques especiales, preparación y embalaje
de productos para exportaciones, estampillado y etiquetados y distribución
física en el país'],
'socios': [{'dni': '33.999.180',
'cuit_cuil': '20-33999180-5',
'domicilio': 'Corrientes 75, ciudad y pdo Lomas de Zamora, pcia Buenos
Aires',
'profesion': 'empresario',
```

```

'estado_civil': 'casado',
'nacionalidad': 'argentino',
'nombre_completo': 'Carlos Emanuel Enriquez',
'fecha_nacimiento': '1988-11-10'},
{'dni': '40.414.400',
'cuit_cuil': '20-40414400-7',
'domicilio': 'Caseros 342 ciudad y pdo Lomas de Zamora, pcia Buenos Aires',
'profesion': 'empresario',
'estado_civil': 'soltero',
'nacionalidad': 'argentino',
'nombre_completo': 'Lucas David Cardozo',
'fecha_nacimiento': '1997-08-20'}],
'duracion': '99 años',
'directorio': [{ 'cargo': 'Presidente',
'nombre_completo': 'Carlos Emanuel Enriquez',
'domicilio_especial': 'Avda Leandro N. Alem 584 CABA'},
{ 'cargo': 'Director Suplente',
'nombre_completo': 'Lucas David Cardozo',
'domicilio_especial': 'Avda Leandro N. Alem 584 CABA'}],
'sede_social': 'Avda Leandro N. Alem 584 CABA',
'suscripcion': [{ 'suscriptor': 'Carlos Emanuel Enriquez',
'monto_integrado': None,
'cantidad_acciones': 18000},
{ 'suscriptor': 'Lucas David Cardozo',
'monto_integrado': None,
'cantidad_acciones': 12000}],
'autorizacion': { 'fecha': '2024-07-31',
'registro': 4,
'matricula': '5375',
'autorizante': 'Guadalupe Zambiazzo',
'numero_escritura': 220,
'tipo_instrumento': 'Escritura Pública'},
'fiscalizacion': 'Los socios',
'administracion': { 'organo': 'Directorio',
'miembros': '1 a 5 directores titulares y 1 suplente',
'representacion_legal': 'Presidente y Director',
'duracion_mandato_ejercicios': 3},
'capital_social': { 'monto': 30000000,
'integracion': None,
'tipo_acciones': None,
'votos_por_accion': None,
'cantidad_acciones': 30000,
'plazo_integracion_saldo': None,
'valor_nominal_por_accion': 1000},
'nombre_sociedad': None,
'fecha_instrumento': '2024-07-31',
'publicacion_boletin': { 'fecha': '2024-08-06',

```

```
'vencimiento': '2024-08-06',
'numero_aviso': '50996/24'},
'fecha_cierre_ejercicio': '30-06'}
```

```
[ ]: partners = parsed_data['socios']
sociedad =
```

```
[ ]: 'INMOBILIARIA: mediante la compra, venta, administración y locación de bienes
inmuebles urbanos o rurales por cualquier titulo'
```

```
[1]: import os
import json
import time
import requests # Importamos la librería correcta
import mysql.connector
from dotenv import load_dotenv

# --- Configuración ---
load_dotenv()
DB_USER = os.getenv("MYSQL_USER")
DB_PASSWORD = os.getenv("MYSQL_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_NAME = os.getenv("MYSQL_DATABASE")

# API de Gemini (deja la KEY vacía, se inyectará en runtime)
API_KEY = ""
API_URL = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.
↪5-flash-preview-09-2025:generateContent?key={API_KEY}"

# Lote de procesamiento
BATCH_SIZE = 50

# --- Prompt y Schema ---
SYSTEM_PROMPT = """
Eres un asistente experto en análisis de documentos legales de Argentina.
Tu tarea es extraer entidades clave de un aviso del Boletín Oficial.
Devuelve SIEMPRE un JSON que cumpla con el schema provisto.
Si un campo no se encuentra, devuélvelo como `null`.
Extrae la información del siguiente aviso:
"""

# Este schema le dice al modelo EXACTAMENTE qué formato JSON debe devolver.
GENERATION_CONFIG = {
    "responseMimeType": "application/json",
    "responseSchema": {
        "type": "OBJECT",
        "properties": {
```

```

        "nombre_sociedad": {"type": "STRING"},
        "fecha_instrumento": {"type": "STRING", "description": "Formato
↪AAAA-MM-DD"},
        "duracion": {"type": "STRING"},
        "sede_social": {"type": "STRING"},
        "fecha_cierre_ejercicio": {"type": "STRING", "description":
↪"Formato DD/MM"},
        "capital_social": {
            "type": "OBJECT",
            "properties": {
                "monto": {"type": "NUMBER", "description": "Monto numérico,
↪ej: 200000"}
            }
        },
        "socios": {
            "type": "ARRAY",
            "items": {
                "type": "OBJECT",
                "properties": {
                    "nombre_completo": {"type": "STRING"},
                    "dni": {"type": "STRING"},
                    "cuit_cuil": {"type": "STRING"},
                    "profesion": {"type": "STRING"},
                    "estado_civil": {"type": "STRING"},
                    "nacionalidad": {"type": "STRING"},
                    "domicilio": {"type": "STRING"}
                }
            }
        },
        "directorio": {
            "type": "ARRAY",
            "items": {
                "type": "OBJECT",
                "properties": {
                    "cargo": {"type": "STRING"},
                    "nombre_completo": {"type": "STRING"},
                    "domicilio_especial": {"type": "STRING"}
                }
            }
        },
        "objeto": {
            "type": "ARRAY",
            "items": {"type": "STRING"}
        }
    }
}

```

```

def get_db_connection():
    """Conecta a la base de datos MySQL."""
    return mysql.connector.connect(
        user=DB_USER,
        password=DB_PASSWORD,
        host=DB_HOST,
        database=DB_NAME
    )

def call_gemini_api(text_aviso, retries=5, delay=5):
    """Llama a la API de Gemini con el texto y el schema (Versión Sincrónica)."""

    payload = {
        "contents": [{
            "parts": [{"text": f"{SYSTEM_PROMPT}\n\n{text_aviso}"}]
        }],
        "generationConfig": GENERATION_CONFIG
    }

    headers = {'Content-Type': 'application/json'}

    for attempt in range(retries):
        try:
            # Usamos requests.post para la llamada de red
            response = requests.post(API_URL, headers=headers, data=json.
↳ dumps(payload), timeout=60)

            # Chequea si la API devolvió un error (4xx o 5xx)
            response.raise_for_status()

            result = response.json()

            if "candidates" not in result or not result["candidates"]:
                raise Exception(f"Respuesta inesperada de la API: {json.
↳ dumps(result)}")

            # Extrae el texto JSON de la respuesta
            json_string = result["candidates"][0]["content"]["parts"][0]["text"]

            # Valida que sea un JSON
            json.loads(json_string)

            return json_string, None # (datos_json, error)

        except requests.exceptions.RequestException as e:

```

```

        # Errores de red, timeouts, o errores HTTP (4xx, 5xx)
        error_msg = f"Error de Red/API: {str(e)}"
        print(f"Intento {attempt + 1}/{retries} fallido: {error_msg}")
    except Exception as e:
        # Otros errores (ej. JSON mal formado en la respuesta)
        error_msg = str(e)
        print(f"Intento {attempt + 1}/{retries} fallido: {error_msg}")

    if attempt < retries - 1:
        # Backoff exponencial síncrono
        sleep_time = delay * (2 ** attempt)
        print(f"Reintentando en {sleep_time} segundos...")
        time.sleep(sleep_time)
    else:
        return None, error_msg # Devuelve el error después de todos los
↪reintentos

    return None, "Reintentos agotados"

def process_batch():
    """Procesa un lote de avisos de la base de datos (Versión Síncrona)."""
    cnx = get_db_connection()
    cursor = cnx.cursor(dictionary=True)

    try:
        # 1. Obtenemos el lote de avisos no procesados
        sql_select = """
        SELECT aviso_id, detalle_aviso
        FROM avisos
        WHERE procesado = FALSE
        AND id_rubro IN (1110, 1130)
        LIMIT %s
        """
        cursor.execute(sql_select, (BATCH_SIZE,))
        avisos = cursor.fetchall()

        if not avisos:
            print("No hay más avisos para procesar.")
            return 0 # 0 avisos procesados

        print(f"Procesando un lote de {len(avisos)} avisos...")

        # 2. Procesamos cada aviso secuencialmente
        results = []
        for aviso in avisos:
            print(f"Procesando aviso: {aviso['aviso_id']}...")

```

```

        json_data, error = call_gemini_api(aviso["detalle_aviso"])
        results.append((json_data, error))
        time.sleep(1) # Pequeña pausa para no saturar la API

# 3. Guardamos los resultados en la base de datos
        sql_insert = """
        INSERT INTO sociedades_procesadas (aviso_id, datos_json, error_msg)
        VALUES (%s, %s, %s)
        ON DUPLICATE KEY UPDATE datos_json = VALUES(datos_json), error_msg =
↪VALUES(error_msg)
        """
        sql_update_flag = "UPDATE avisos SET procesado = TRUE WHERE aviso_id =
↪%s"

        processed_count = 0
        for aviso, (json_data, error) in zip(avisos, results):
            aviso_id = aviso["aviso_id"]
            if error:
                print(f"Error final en Aviso {aviso_id}: {error}")
                cursor.execute(sql_insert, (aviso_id, None, error))
            else:
                cursor.execute(sql_insert, (aviso_id, json_data, None))
                processed_count += 1

                # Marcamos el aviso original como procesado (incluso si falló, para
↪no reintentar)
                cursor.execute(sql_update_flag, (aviso_id,))

        cnx.commit()
        print(f"Lote finalizado. {processed_count} exitosos, {len(avisos) -
↪processed_count} fallidos.")
        return len(avisos)

    except mysql.connector.Error as err:
        print(f"Error de base de datos: {err}")
        if cnx:
            cnx.rollback()
        return 0
    finally:
        if cnx and cnx.is_connected():
            cursor.close()
            cnx.close()

def main():
    """Bucle principal de procesamiento (Versión Síncrona)."""
    total_procesados = 0
    while True:

```

```

try:
    procesados_en_lote = process_batch()
    total_procesados += procesados_en_lote

    if procesados_en_lote == 0:
        print("Proceso completado. Total de avisos: ", total_procesados)
        break

    # Pequeña pausa entre lotes
    print("Esperando 5 segundos antes del siguiente lote...")
    time.sleep(5)

except Exception as e:
    print(f"Error crítico en el bucle principal: {e}")
    print("Esperando 60 segundos antes de reintentar...")
    time.sleep(60)

# Para iniciar, simplemente ejecuta este script de Python:
# python procesar_api.py
if __name__ == "__main__":
    print("Iniciando el procesador de avisos...")
    main()

```

Iniciando el procesador de avisos...

Procesando un lote de 50 avisos...

Procesando aviso: A322...

Intento 1/5 fallido: Error de Red/API: 403 Client Error: Forbidden for url:
<https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-09-2025:generateContent?key=>

Reintentando en 5 segundos...

Intento 2/5 fallido: Error de Red/API: 403 Client Error: Forbidden for url:
<https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-09-2025:generateContent?key=>

Reintentando en 10 segundos...

Intento 3/5 fallido: Error de Red/API: 403 Client Error: Forbidden for url:
<https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-09-2025:generateContent?key=>

Reintentando en 20 segundos...

KeyboardInterrupt

Traceback (most recent call last)

Cell In[1], line 236

```

234 if __name__ == "__main__":
235     print("Iniciando el procesador de avisos...")
--> 236     main()

```

Cell In[1], line 216, in main()

```

214 while True:
215     try:
--> 216         procesados_en_lote = process_batch()
217         total_procesados += procesados_en_lote
219         if procesados_en_lote == 0:

Cell In[1], line 172, in process_batch()
    170 for aviso in avisos:
    171     print(f"Procesando aviso: {aviso['aviso_id']}...")
--> 172     json_data, error = call_gemini_api(aviso[
    173     results.append((json_data, error))
    174     time.sleep(1) # Pequeña pausa para no saturar la API

Cell In[1], line 138, in call_gemini_api(text_aviso, retries, delay)
    136     sleep_time = delay * (2 ** attempt)
    137     print(f"Reintentando en {sleep_time} segundos...")
--> 138     time.sleep(sleep_time)
    139 else:
    140     return None, error_msg # Devuelve el error después de todos los
↪reintentos

KeyboardInterrupt:

```