# Trabajo Práctico

1º cuatrimestre 2025

Base de Datos
Cátedra Merlino


Leonardo Duchen
104885
lduchen@fi.uba.ar

Martín Guerrero
107774
mguerrero@fi.uba.ar

Maria Agustina Fontana
108090
mafontana@fi.uba.ar

Martín Guerra
106874
mguerrae@fi.uba.ar

Milton Fernández
108749
mitfernandez@fi.uba.ar

Agustín Demicheli
111032
ademicheli@fi.uba.ar

## Objetivo General

Desarrollar un proyecto centrado en bases de datos que integre técnicas de **data cleaning**, acceso masivo a datos, y el uso de **inteligencia artificial (IA)** para asistir en la toma de decisiones. Documentar todo el proceso, incluyendo los *prompts* utilizados con IA y sus resultados.

## Introducción

Este trabajo aborda la construcción de un sistema completo de gestión y consulta de datos de alquileres, combinando procesos de **ETL (Extract, Transform, Load)**, almacenamiento en **bases de datos relacionales**, y el desarrollo de una **API REST** para consultas. Se parte de un archivo CSV con información de alquileres, que es limpiado y transformado con técnicas de *data cleaning* y normalización antes de cargarse en una base de datos SQLite. Luego, se desarrolla una API para exponer los datos y permitir búsquedas, con una interfaz gráfica.

## Desarrollo de Proyecto

Para la selección de un dataset adecuado (de datos abiertos) realizamos la búsqueda en Kaggle. Evaluamos considerando que los datos sean de un volumen suficiente y que presenten ciertos problemas o inconsistencias típicas. Esto nos permitió desarrollar de manera práctica las etapas de extracción, transformación y carga, enfocándonos en la limpieza y normalización de datos.

**Extract (Extracción)**
Se carga un archivo CSV de alquileres usando pandas y se guarda en una base SQLite en memoria para procesamiento posterior. El DataFrame se guarda como una tabla SQL llamada alquileres.

**Transform (Transformación)**
Se inició el proceso de limpieza de datos, que consistió en eliminar valores nulos, detectar y tratar outliers, y corregir inconsistencias en la información, entre otras cosas.

Valores faltantes
Se analizó el porcentaje de valores faltantes por columna para seleccionar aquellas que resultaban útiles:

- Si el porcentaje es **mayor al 50%**, la columna se elimina.
- Si está entre **20% y 50%**, se evalúa su relevancia para decidir si se conserva o no. Si es **menor al 20%**, se considera su imputación.

Eliminación de duplicados
Se eliminan registros duplicados basados en los campos: *property_title, place_l2, place_l3, property_price, property_type, latitud, longitud.* Luego, se calculo el porcentaje de registros eliminados (12.39%).

Análisis de outliers
Se realizó un análisis de valores atípicos en la variable *property_price*. Para ello:

- Se calcularon el primer cuartil (Q1) y el tercer cuartil (Q3) de la distribución de precios.
- Se obtuvo el rango intercuartílico (IQR = Q3 - Q1) para determinar los valores fuera del rango.
- Se identificaron los registros con precios fuera de los límites del rango para extraerlos.

Análisis de inconsistencias
Buscamos datos incoherentes y analizamos cuantos nos afectaría eliminarlos. Los datos a analizar serían los siguientes:

1. Registros sin rooms y sin superficie total válida.
2. Registros con cantidad máxima de habitantes negativa.
3. Registros sin dormitorios pero con más de un ambiente.
4. Registros donde la superficie útil supera a la total por más de 1 m².
5. Registros con superficie total y útil menor o igual a cero.
6. Registros con más dormitorios que ambientes + 2.
7. Registros con ambientes pero sin datos de superficie (útil o total).

Observamos que eliminamos aproximadamente un 15% de los registros, lo cual nos parece aceptable para la calidad final del conjunto de datos.

Normalización de datos
Se identificaron y corrigieron diferencias en los nombres de provincias y localidades (por ejemplo, variaciones como "laplata", "La plata", "La Plata") utilizando rapidfuzz (fuzzy matching). Procedemos a agrupar las variantes similares y aplicamos las correcciones.

Se realiza una normalización de localidades, se crea una tabla e inserta todas las combinaciones únicas de localidad y provincia que aparecen en mi tabla. Agrego una columna *id_localidad* en la tabla alquileres para guardar la referencia a la localidad normalizada. Para cada fila de alquileres, busco el id de la localidad correspondiente y lo guardo en la nueva columna; haciendo que las columnas *place_l2* y *place_l3* ya no sean necesarias.

**Load (Carga)**
Para almacenamiento y persistencia, utilizamos una base de datos **SQLite** alojada localmente en un archivo llamado `mi_base.sqlite`. Los datos limpios y normalizados se guardan en esta base relacional.

**Implementar UI para búsqueda**
Se implementó una API para realizar búsquedas sobre los datos de alquileres y una interfaz de usuario para publicar alquileres por medio de una API.
La API puede ejecutarse con: `uvicorn api_alquileres:app --reload`

## Problemas Detectados

Durante el análisis exploratorio de los datos, se detectaron inconsistencias en la escritura de los nombres de las localidades, por ejemplo: 'San Cristobal' y 'San Cristóbal' aparecían como valores distintos en la columna place_l3. Este tipo de errores puede generar duplicados lógicos y dificultar tanto el análisis como la integración de los datos.

Para solucionar este problema, se realizó una corrección manual de los valores detectados, unificando las variantes bajo un mismo nombre estándar. Posteriormente, se implementó un proceso de normalización creando una tabla aparte de localidades (localidades), donde cada localidad tiene un identificador único y se almacena junto con su provincia.

En la tabla principal de alquileres, se reemplazaron los nombres de localidad y provincia por una clave foránea (id_localidad) que hace referencia a la tabla normalizada de localidades. Esta clave foránea garantiza la integridad referencial entre ambas tablas: cada registro de alquiler queda asociado únicamente a una localidad válida y existente en la tabla de localidades. De este modo, se evita la aparición de valores inconsistentes o inexistentes y se facilita la actualización o corrección de datos, ya que cualquier cambio en el nombre o la provincia de una localidad se realiza una sola vez en la tabla normalizada y se refleja automáticamente en todos los registros asociados.

En resumen, el uso de una clave foránea no solo mejora la calidad y consistencia de los datos, sino que también optimiza las consultas y el mantenimiento de la base, asegurando que todas las referencias a localidades sean válidas y estén centralizadas.

## Decisiones de Diseño

### 1. Elección del Dataset

- **Decisión:** Usar un archivo CSV de datos de alquileres obtenido desde Kaggle.
- **Justificación:** Este tipo de archivo es de fácil acceso, tiene un formato plano que permite el análisis con herramientas comunes como pandas, y contenía suficientes registros con problemas reales (errores, valores faltantes) que justificaban un proceso robusto de limpieza y transformación.

### 2. Uso de Python y pandas para ETL

- **Decisión:** Realizar el proceso ETL utilizando Python y la librería pandas.
- **Justificación:** Python ofrece una sintaxis simple y una comunidad amplia. Pandas es excelente para trabajar con datasets tabulares y permite limpiar, transformar y analizar datos de forma eficiente en notebooks interactivos.

**3. Elección de SQLite como base de datos**

- **Decisión:** Utilizar SQLite para almacenar los datos.
- **Justificación:** Es una base de datos liviana, sin necesidad de configuración de servidor, ideal para prototipos y proyectos individuales. Su integración con Python es directa mediante `sqlite3`.

**4. Normalización de la base de datos**

- **Decisión:** Crear tablas separadas para entidades como las localidades.
- **Justificación:** Mejora la integridad referencial, evita duplicaciones y facilita el mantenimiento de los datos. Esta estructura también optimiza las consultas y permite una representación más clara de las relaciones.

**5. Implementación de claves foráneas**

- **Decisión:** Reemplazar nombres directos por IDs que referencien otras tablas (como `id_localidad`).
- **Justificación:** Garantiza que solo se utilicen valores válidos. Mejora la calidad del modelo de datos y permite escalar el sistema si se agregan nuevas relaciones o atributos.

**6. Diseño de la API con FastAPI**

- **Decisión:** Usar FastAPI para crear la API REST.
- **Justificación:** FastAPI permite definir endpoints de manera rápida, con validación automática de datos y documentación integrada. Es liviana, moderna y está diseñada para alto rendimiento, incluso en entornos pequeños.

**7. Separación entre Backend y Frontend**

- **Decisión:** Implementar una interfaz gráfica conectada a la API.
- **Justificación:** Separar responsabilidades facilita el mantenimiento, permite cambios independientes y mejora la experiencia del usuario final. La UI hace más accesible el sistema para personas no técnicas.

**8. Uso de IA para asistencia técnica**

- **Decisión:** Consultar IA (ChatGPT) para definir pasos, resolver problemas técnicos y generar código.

- **Justificación:** Agilizó la toma de decisiones, redujo el tiempo de búsqueda de información y sirvió como guía práctica para resolver obstáculos técnicos.

## Prompts de IA Utilizados

*“¿Cómo puedo desarrollar un proyecto centrado en bases de datos que incluya acceso masivo a datos y técnicas de data cleaning? ¿Qué recomendaciones hay para elegir un dataset, dónde y cómo limpiarlo, y qué tipo de base de datos utilizar para realizar un desarrollo eficiente de lo antes mencionado? Generame una lista de opciones que pueda evaluar”*

*"¿Cómo cargo un CSV en pandas y lo guardo en memoria con SQLite para hacer consultas con SQL?"*

*"¿Cómo analizo los datos faltantes en un DataFrame de pandas y qué acciones puedo tomar según el porcentaje?"*

*"¿Cómo elimino filas duplicadas en pandas? Quiero también mostrar el porcentaje de registros eliminados"*

*"Dame código para calcular el porcentaje de valores nulos por columna en un DataFrame de pandas"*

*"¿Cómo elimino outliers en una columna numérica como 'property_price' usando el rango intercuartil?"*

*“Haceme un gráfico de distribución del precio por tipo de propiedad”*

*"¿Cómo normalizo nombres de columnas y valores string en pandas?”*

*“¿Cómo puedo estructurar una API en FastAPI que use SQLite, pandas y Pydantic para manejar registros con ubicación geográfica, permitiendo inserciones, filtros por criterios opcionales y listados dinámicos, además de habilitar CORS para desarrollo?”*

*“¿Cómo hago para que los usuarios puedan publicar su propiedad en mi app de alquileres usando FastAPI y SQLite?”*