https://www.kaggle.com/datasets/victorianomh/idealista-barcelona-raw-scraped-data

Name in our schema	Name in kaggle
ID vivienda	**id**- consider changing to simple 1, 2, 3 etc (row numbers.
Nombre vivienda	generate
ID mapa ubicacion	?
Nombre mapa de Ubicacion	?
ID de inmobiliaria	Can just register all the properties under one ID de immobilaria e.g. 1. Assume they are all with the same real estate agent, as the dummy set does.
Nombre de inmobiliaria	Same as above. ZY living in dummy set
Interes	Might need to randomly allocate half to rental and half to purchase
Tipo de vivienda	**property_type**
Estado	**Second_hand needs_renovating,** neither=good cond
Habitaciones	**n_bedrooms**
Baños	**bathrooms**
Muebles	Generate y/n
Precio de alquiler	**price**
Precio de compra (€)	**Price-** only one price listed in kaggle set. Need to decide how to differentiate between rental and purchase price
Superficie total (m²)	**sq_mt_built**
Calefacción	**heating**
Aire acondicionado	**Air conditioning**
Garaje	**parking**
Trastero o bodega	Randomly allocate y/n
Balcón	**balcony**

Terraza	**terrace**
Piscina	**Swimming pool**
Jardín	**garden**
Ascensor	**lift_x**
Armarios empotrados	Randomly allocate y/n
Tipo de planta	**Floor_y-** need to categorise
Vistas al exterior	**exterior_x**
Vistas al mar	Randomly allocate y/n- need to check which neighbourhoods are near enough to the sea though
Vigilancia	Randomly allocate y/n
Vivienda de lujo	Randomly allocate y/n
Vivienda accesible	Randomly allocate y/n
Superficie útil (m²)	**sq_mt_floor_area**
Certificado energético	**consumption_in_mkw/m2_year, emissions_in_kgco2/m2_year**
Año de construcción	**Year_built**
Ubicación: calle y número	**address**
Ubicación: piso/número/letra/portal	generate
Código postal	Can figure from address
Localidad	**city**
Provincia	Can figure from address
Distrito	Can figure from address
Barrio	**neighbourhood**
Gastos de comunidad (€ / mes)	Need to estimate somehow
Tipo de calefacción	**Central_heating, heating_type**
Número de planta	**floor_y**
Orientación	**orientation**

Precio por metro cuadrado (€) (IA)	calc
Nota (IA)	**description**

energy certificate

Letter	Non-renewable primary energy consumption(kWh/m²·año)	CO_2 emissions(kg CO_2/m²·año)
A	< 44.6	< 10
B	44.6 – 72.3	10 – 16.3
C	72.3 – 112.1	16.3 – 25.3
D	112.1 – 172.3	25.3 – 38.9
E	172.3 – 303.7	38.9 – 66.0
F	303.7 – 382.6	66.0 – 79.2
G	> 382.6	> 79.2

Rating each indicator separately against the national IDAE reference scale (see tables below).

Taking the worst of the two (e.g. if consumption is “C” but emissions are “E”, the final class is “E”).

Rounding down (you can never obtain a better global letter than either partial letter).

The block first translates each home’s raw efficiency metrics into the Spanish A-to-G scale: it runs the primary-energy figure (*consumption_in_mkw/m2_year*) and the CO_2 figure (*emissions_in_kgco2/m2_year*) through the official IDAE break-points, giving one letter for consumption and one for emissions. It then keeps the **worst** (i.e., the less efficient) of those two letters as the dwelling’s overall **“Certificado energético”** grade, replicating the rule used in real certificates. Finally, it cleans the table so it matches the Visitia schema: the numeric columns and the two helper letters are dropped, leaving a single, ready-to-use *Certificado energético* column in `df_clean` and a quick value-count print-out to check the distribution.

Estado column GPT prompt

I have two boolean columns in my dataset: second hand and needs renovating

I want to use these columns to create one overall column called 'Estado' then drop these two original columns, leaving just Estado. I want Estado to contain 4 possible values: a reformar, en buen estado, reformado, a estrenar.

i think we map:

second hand(true) and needs renovating (true) --> a reformar
second hand (true) and needs renovating (false) --> en buen estado, reformado (randomly select between the two options)
second hand (false) and needs renovating (true) --> a reformar
second hand (false) and needs renovating (false) --> a estrenar

can you give me code for this

Price

The script classifies each property in **`df_clean`** as **“obra nueva”**, **“alquilar”** or **“comprar”** and then splits the single `price` column accordingly. It first sets a rental-price cut-off (`RENT_THRESHOLD`, here €20 000) and compiles two regex patterns: one that spots new-build keywords and another that spots rental terms. For every row, `classify_interes()` looks at the listing’s free-text description plus a few metadata clues (brand-new flags, very recent `year_built`) to label it “obra nueva”; otherwise, if the text mentions renting or the price is below the threshold it labels it “alquilar”; all remaining listings default to “comprar”. The resulting label is stored in a new `Interes` column. Next, two `np.where()` calls copy the numeric price into either `Precio de compra (€)` or `Precio de alquiler`, leaving the other column as NaN so the dataset now matches the target schema. Finally, a value count of `Interes` is printed so you can see how many listings fall into each category.

Floor mapping

	Rationale	Implementation

Numero de Planta should simply reproduce the numeric storey indicator you already have, so we copy `floor_y` verbatim.	`df_clean['Numero de Planta'] = df_clean['floor_y']`
Tipo de planta groups floors into three intuitive strata: • **Baja** for any level at or below street (−2, −1, 0) • **Intermedia** for the low-to-mid band of typical living floors (1 → 5) • **Alta** for upper-storey units (6 +)	Use `np.select()` with those three conditions.

Address

The snippet first filters the dataset to keep only those listings whose `adress` text contains any common Spanish street-type word—“calle”, “avenida/avda.”, “paseo/pso.”, “plaza/pl.”, “rambla”, “carrer”, “camino”, “ronda”, “vía/via”, “travesía”, “callejón” or “glorieta”. It builds a case-insensitive regular-expression of those tokens, uses it to create a boolean mask, and copies just the matching rows into a new `df_clean`. Next, for every remaining row it ensures the address ends with a house number: if the string already contains a digit it is left untouched; otherwise the function appends a random number between 1 and 299, yielding a realistic street-and-number format. The result is stored in a new column named **“Ubicacion: calle y numero”**, after which the original `adress` column is dropped so the schema aligns with Dataset 1. Finally, it prints a few sample addresses and the number of rows retained, giving a quick sanity check that only plausible street addresses remain.

Gastos de la comunidad

The block builds a new column — **“Gastos de la comunidad (€ por mes)”** — for every listing.
First it tries to read a fee explicitly written in the ad’s description, using a regex that captures patterns like “gastos de comunidad 125 €”. If that fails, it produces a realistic estimate: it starts with a base cost of 0.70 €/m² and adds surcharges for amenities that raise the building’s budget (lift +0.15 €, pool +0.25 €, garden +0.10 €, garage +0.10 €, central heating +0.35 €). It multiplies the resulting €/m² figure by the dwelling’s **Superficie total (m2)**, then introduces a small ±5 % random jitter so neighbouring flats don’t all show identical numbers. The function picks the declared fee when available, otherwise the estimate, and writes the result to the new column. Finally it prints summary statistics and a few sample rows so we can verify that most listings now carry plausible monthly community-fee values (about 26 € to 560 €, median ~83 €).

Heating type

The snippet builds the new **“Tipo de calefacción”** field by inspecting three existing clues for each listing. It first looks at the Boolean flag **Calefacción** (renamed earlier from *heating*) and the text field **central_heating** to decide whether the property has heating at all; if neither is true, the listing is labelled **Sin calefacción**. If heating exists, it checks **heating_type**—converted to lower-case for robustness—to map well-known technologies to Visitia’s categories: any value containing “natural” or “gas” becomes **Caldera de gas natural**; “air”, “aerotermia” or “heat pump” becomes **Bomba de calor (Aerotermia)**; and

“biomasa/biomass”, “pellet”, or “wood” becomes **Caldera de biomasa**. When heating is present but the type is unknown or something unusual (electric, fuel oil, etc.), the code assigns **Otro tipo de calefacción**. After populating the consolidated column, it drops only the auxiliary columns **central_heating** and **heating_type**—keeping **Calefacción** intact—so the DataFrame now matches Visitia’s schema while preserving the simple on/off flag for further use.

Price per square metre

The code adds a new column, **precio por metro cuadrado (€)**, that expresses how much each property costs per square-metre.
For every row it first chooses the most relevant price: if a sale price (**Precio de compra (€)**) is present it uses that; otherwise, if the listing is a rental, it uses the monthly rent (**Precio de alquiler**); if neither exists it leaves the value as missing. It then divides that price by the built floor-area stored in **Superficie total (m2)** and rounds the result to two decimals, writing it to the new column only when both inputs are valid. Finally, `describe()` prints summary statistics so we can see the distribution of €/m² values and confirm that the calculation worked as expected. All existing columns stay untouched; the script merely appends this derived metric to facilitate price-per-area analyses.

District and province

The block adds the two geographical layers that were still missing in our processed dataset. First, because every `Localidad` in our feed—Barcelona, Badalona, Santa Coloma de Gramenet, L’Hospitalet and Sant Adrià—belongs to the province of Barcelona, it simply inserts a new **Provincia** column filled with “Barcelona” for all rows. Next, it creates **Distrito**, but only for listings that are actually inside the city of Barcelona: a dictionary translates each recognised barrio (e.g. *El Raval*, *La Sagrada Família*, *Pedralbes*) to its official municipal district (Ciutat Vella, Eixample, Les Corts, etc.). If the listing is in a surrounding town or its barrio is not yet in the dictionary, the code leaves *Distrito* as NaN—making unmapped cases easy to spot later. A quick printout then shows the first few lines plus a count of how many rows received a district label, confirming the enrichment worked as intended without altering any other columns.

Postcodes

The new cell completes the **Código postal** column for every listing. First, we rebuilt the `barri_zip` dictionary so it now contains a postcode for **every barrio that appears in our data**, including the 14 neighbourhoods that were still missing (e.g., *La Barceloneta → 08003*, *La Trinitat Nova → 08033*, *Vallvidrera-El Tibidabo → 08017*). When the `assign_cp` function runs, Barcelona-city rows pick their ZIP from this dictionary, while properties in the four surrounding municipalities take a single central code from `localidad_zip`. After applying the function, the verification printout shows that the “Missing postcodes” count drops to **0**, confirming every one of the 821 listings now carries a valid postcode without altering any other columns.

Sea view

The snippet adds a new **“Vistas al mar”** flag to every listing. It first defines two lookup sets: one containing all Barcelona neighbourhoods that physically border the Mediterranean and another listing the two surrounding towns (Badalona and Sant Adrià de Besòs) whose entire municipality is coastal. For each row the helper function checks whether the property’s *Localidad* or *Barrio* appears in those coastal sets. If the home is inland the column is automatically **“No”**; if it is in a coastal area the function uses NumPy’s random generator to assign **“Sí”** or **“No”** with equal probability, simulating the real-world fact that not every seafront dwelling actually enjoys sea views. The result is stored in the new column without touching any of the existing data, and a quick count confirms the expected distribution of Yes/No values.

Accessibility, wardrobes, storage locker

The block adds three new yes/no features—**Armarios empotrados, Vivienda accesible,** and **Trastero o bodega**—but instead of a blind 50-50 shuffle it weights the random choice with simple, realistic cues already in the data. For built-in wardrobes the code gives an 80 % chance of “Sí” when the flat is brand-new, recently refurbished, or labelled *obra nueva*, and 40 % otherwise. For accessibility it sets the probability of “Sí” at 70 % when the building has a lift (*Ascensor* =*True*) and only 20 % if it doesn’t. For a storage room it makes “Sí” slightly more likely (60 %) when the listing includes a garage, reflecting the common pairing of parking and a storage locker, and 30 % elsewhere. Each row is processed with NumPy’s random generator so the final dataset contains plausible—but still synthetic—distributions while keeping every original column intact.

Muebles, Vigilancia, Vivienda de lujo

Column	Main driver(s)	Probability for “Sí”	Rationale
Muebles	`Interes == 'alquilar'`	75 % (rent) / 30 % (sale)	Furnished units are far more common in the rental market.
Vigilancia	Count of premium amenities (Ascensor, Piscina, garage, Jardín)	15 % baseline +15 % per amenity, capped at 75 %	Buildings with several shared services are likelier to pay for security/concierge.

Vivienda de lujo	Price > €600 k **or** €/m² > €4 500	80 % if one of those thresholds is met, else 10 %	High absolute price or very high price-per-m² is a strong luxury signal; others get an occasional yes.

Ubicacion: Piso y numero, letra o portal

To give every listing a realistic flat identifier, we generated **“Ubicación: Piso y número, letra o portal.”** The code converts the existing numeric *Numero de Planta* into a floor label (e.g. **“3º”**, **“Bajo”**, **“Sótano 1”**), appends a randomly chosen door letter **A–D**, and—only for buildings that either have a lift or rise above four storeys—adds a random **“Portal 1–3”** to mimic larger stair-wells. This keeps basement and ground-floor wording correct, varies unit letters across the block, and reflects the fact that taller or better-equipped buildings often have multiple portals. No original columns are touched; we simply attach this plausible, fully synthetic internal address to every property.

ID and name of real estate agent

ID	Nombre de inmobiliaria	Municipio de origen	Source
1	**Lucas Fox Barcelona**	Barcelona	(inmoselect.com)
2	**VIP House Badalona**	Badalona	(idealista.com)
3	**Fincas Mirafer**	Santa Coloma de Gramenet	(fincasmirafer.com)
4	**Punt Buenos Aires BCN**	L’Hospitalet de Llobregat	(idealista.com)
5	**Asa Habitatges**	Sant Adrià de Besòs	(asahabitatges.es)

Name of property

The snippet builds a human-readable **Nombre vivienda** for each listing by combining two existing fields. It first capitalizes the property type (*Tipo de vivienda*—e.g. “Piso”, “Ático”) to give context, then converts the *Ubicación: calle y número* string to title-case and shortens common words (“Calle”→“C.”, “Avenida”→“Av.”) so the address looks concise and professional. The two pieces are concatenated—“Piso C. Del Raval 173”, “Ático Av. Paral·lel 149”, etc.—and stored in a new column, leaving all other data untouched. This provides a clear, unique label for each property that sales teams can quote or customers can recognise at a glance.

ID and name mapa ubicacion

The code assigns a unique integer to every distinct barrio (starting at 1) and stores the mapping in **ID mapa ubicacion**; it then duplicates the barrio name itself into **Nombre mapa de Ubicacion**.