Master thesis

Alessandro Cosci y Miguel Ángel Alberola

Master in Data Science (4th ed. 2022-2023)

KSCHOOL

Index

Index **2**

Business Understanding **2**
- Project introduction and goal 3
- The real estate market 4
- State of the art 5
- Methodology 6
- Tools used 7

Data understanding **8**
- Origin of the data 8
  - Fotocasa scraping 9
  - Idealista scraping 12
- Features definition 14
  - Fotocasa feature list 15
  - Idealista feature list 16
  - Final features 17

Data Preparation **18**
- Data cleaning 19
  - Features 19
  - Duplicate records 23
  - Wrong values 26
- Data exploration and feature analysis 30
  - Distribution analysis and anomalies handling 30
  - Features correlation 33
  - Features analysis 35
- Feature engineering 48

Modeling **50**
  - Models' evaluation 52
  - Clustering 53
  - Hyperparameters 54
    - CatBoost 55
    - LightGBM 57
- Content based recommender 58

Evaluation **59**

Deployment **66**
  - User manual 66
    - How to use it 67

Conclusions and next steps **74**

References **75**

Business Understanding

Project introduction and goal

This project comes with the objective of knowing the ideal market prices of second-hand homes in the municipality of Madrid. To do so, the model analyzes a dataset of real estate properties. Given a specific set of property's characteristics, the model is able to estimate its ideal price range.

The ideal end user is a person or family willing to buy or sell a property aimed for residential purposes, having an average annual income. This means that in our project we are not considering properties meant for investors, such as properties to be converted into hostels or to be used for students renting, or even luxury ones. The features we used to classify these particular types of properties are price, number of bedrooms and number of bathrooms. The thresholds we used to distinguish properties within our scope from the ones that are not, depended on the data itself and is explained in the “Data preparation” section. The objective of this project is to give such users an approximate pricing for a property, having as “technical objective” a RMSE of at most 30.000€.

To meet this objective, we started by obtaining more than 30.000 properties (before cleaning) from the best-known purchase and rental portals in Spain: Idealista and Fotocasa. With the combination of both databases we obtain a series of common characteristics that allow us to delimit the determinants of the price of each house and achieve the desired prediction.

We started from different problems related to the housing sector and how the aforementioned portals worked, such as possible scams, properties that did not meet the requirements mentioned in the last paragraph (no luxury, no investment), duplicate advertisements with different prices and houses with overestimated prices that can harm the algorithm's own learning. Therefore, a series of decisions, that will be explained further ahead, have been made throughout the project in order to minimize the impact of these issues.

The project is based on the CatBoost regression model, which allows obtaining the desired prediction with the best performance according to the tests carried out. This boosting algorithm was developed in 2017 by Yandex. It is based on the descending gradient, which means that it creates sequential decision trees that learn from the previous ones.

Throughout this document we will review the entire process followed to achieve the explained objective. We will talk about the different tools used and why we used them, as well as the entire exploratory data analysis procedure and the reasons

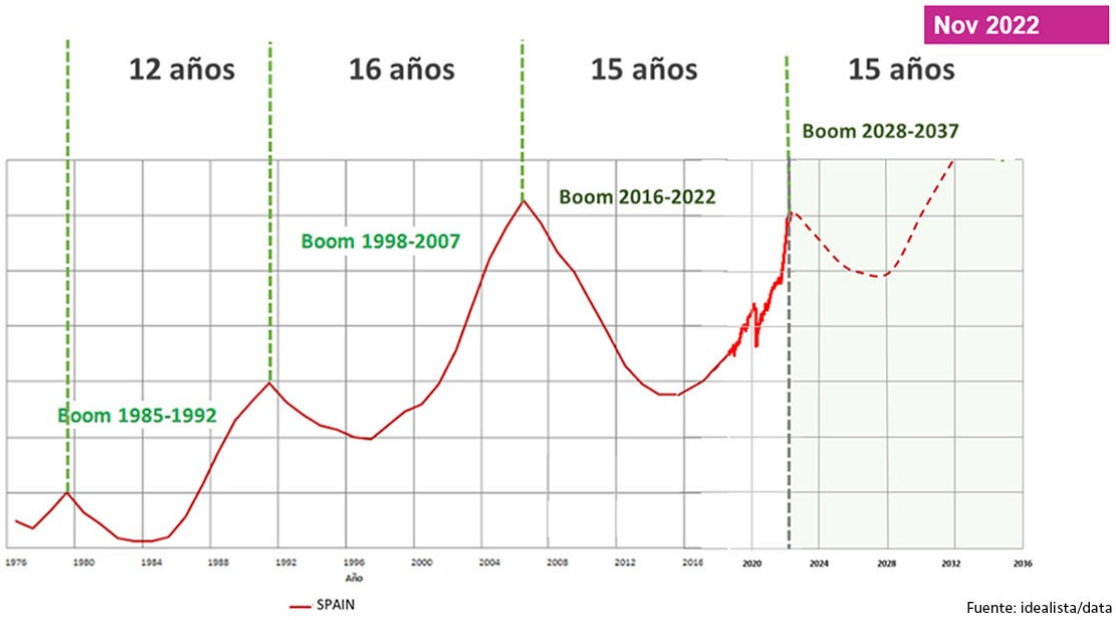
behind the decisions we made. All this led us to the application of the model and the development of a frontend with which the end user can indicate the characteristics of his/her desired property and obtain the estimated price and a series of recommendations for existing properties in the database on which we have worked that meet such criterias.

The real estate market

To understand the context in which we find ourselves, we must take a look at the recent years in which different external situations have conditioned all markets, including real estate.

In Spain, we started from the real estate crisis that occurred in 2008 after a time of prices, wages and population rising that produced a time of economic growth without a clear horizon. This made banks offer mortgage loans without paying too much attention to the solvency of the mortgaged, which turned out to be insufficient after the rise in the Euribor, and left these mortgages uncovered.

This situation caused housing prices to plummet between 2008 and 2009 and a domino effect in the Spanish economy, with a broken banking sector and the population suffering from an economic crisis.



After the collapse caused by the 2008 crisis, house prices continued to fall until they reached a minimum in 2013, years after which they began to rise again. In 2020 the COVID-19 pandemic arrived, affecting the entire world. The confinement considerably reduced the spending of Spanish families and increased savings and also changed the needs of people in terms of housing requirements; This, together

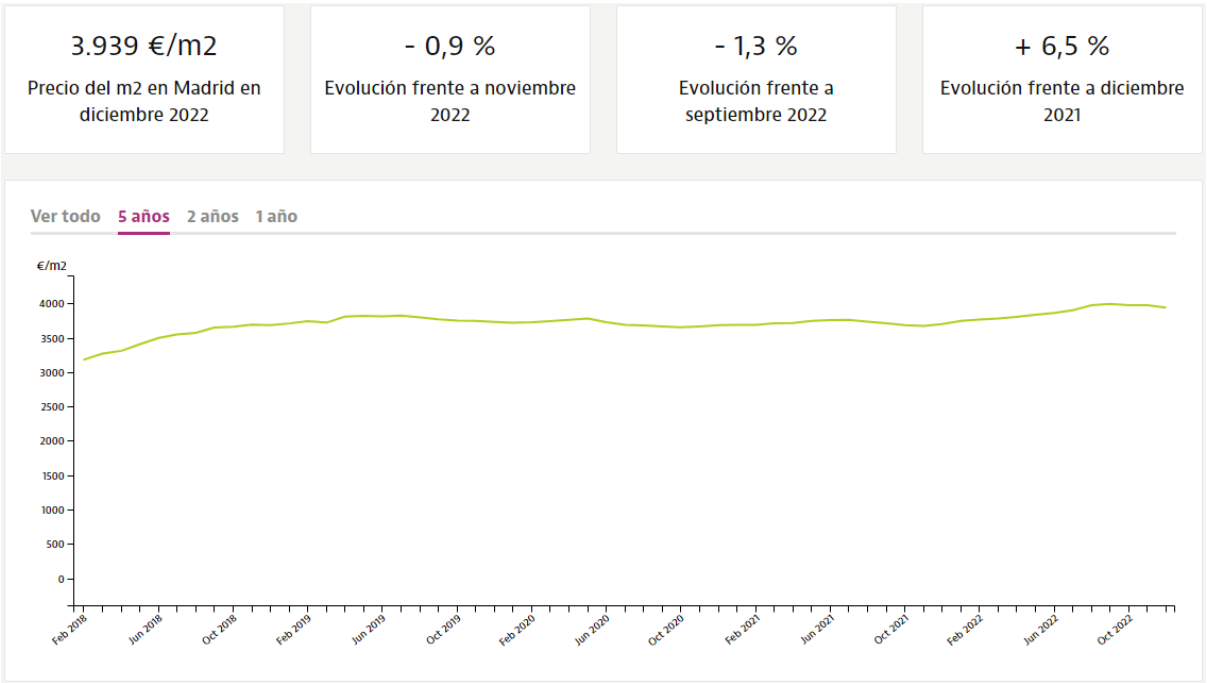
with the good conditions of mortgage loans, increased the demand for housing, which, in turn, caused prices to rise further.

As if the pandemic were not enough, at the beginning of 2022 the war between Russia and Ukraine began, causing a new world crisis and aggravating Spain's economic problems due to the supply of resources. All this caused an even greater insecurity in Spanish families in the face of skyrocketing inflation.

This increase in prices in all sectors, including housing, has forced the European Central Bank, at the end of 2022, to raise interest rates several times in order to cool the market down and prevent further price rising. This, in turn, is causing the mortgages to become more expensive, putting once again the solvency of mortgagees in check and reducing the demand for housing.

As can be seen in the graph below from Idealista, the housing market in Spain breaks down into fairly similar boom and bust cycles. According to the experts of this platform, we are currently at the end of one of those cycles that will bring a new drop in prices from 2023 due to the lack of demand.

If we focus on Madrid, the chosen location for this project, we find an upward trend until September 2022, when prices started dropping slightly, although they are still higher than at the end of 2021 (according to Idealista)[1].



State of the art

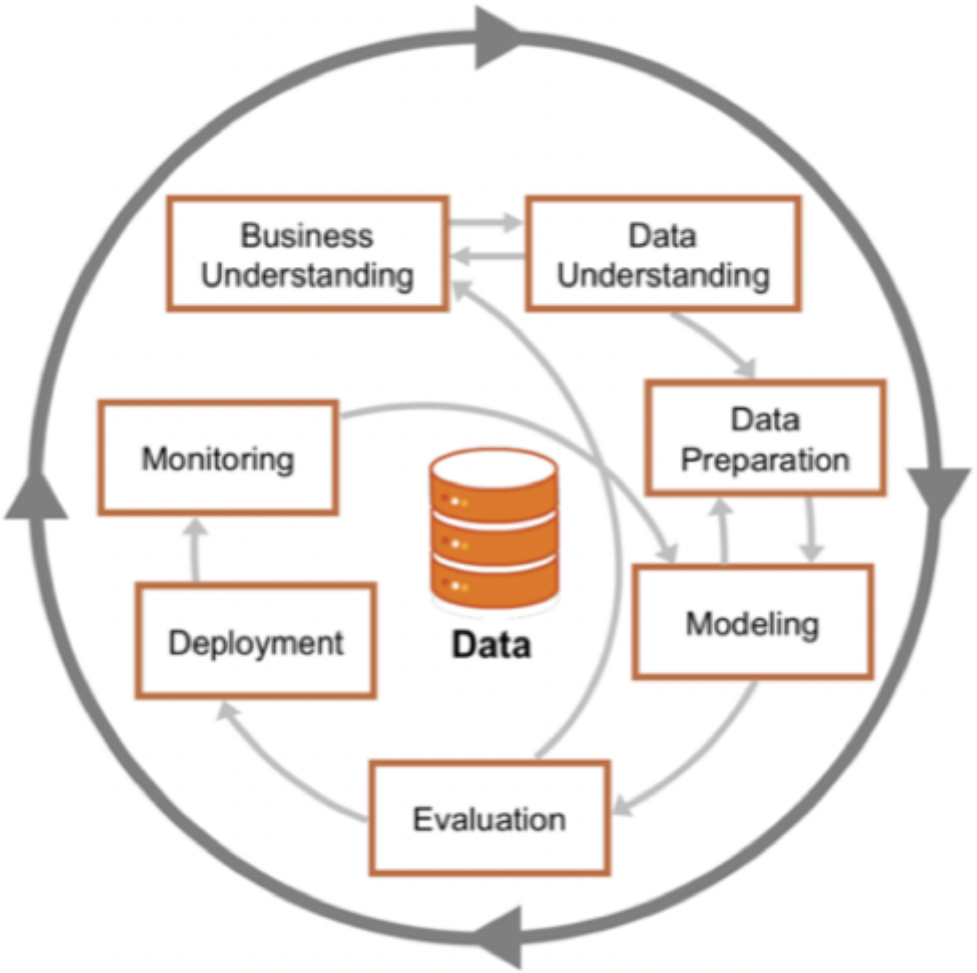
The price prediction of the real estate market is something that has been done for years and we can find many examples throughout the network. The Idealista and Fotocasa real estate platforms themselves have data departments in charge of this type of analysis, although they focus on its evolution over time.

In the same way, we find examples of similar projects on platforms such as Kaggle, as well as numerous national and international papers that follow similar processes.

Therefore, the objective of this data science project is mainly educational, without post-scalability claims to which it would be necessary to have complete access to the data, without having to rely on scraping.

Methodology

The methodology followed during the project has been CRISP-DM[2]. This consists of a constant process of repetition in which each phase allows us to return to the previous ones and continue advancing in order to be able to apply the knowledge acquired in previous phases and improve the final result of the project.



In the development of the project, we have followed the phases of this methodology. First, we analyzed the web structure of Fotocasa and Idealista for the scraping; Once we obtained the data, we unified the common features to create a single dataset that we prepared and analyzed. Subsequently we proceeded to the modeling, its evaluation and finally its interface. All this by constantly going back to previous phases to make corrections in the scraping, in the data cleaning and preparation or in the modeling itself according to its evaluation.

This process matured the project little by little, from a first MVP in which we only obtained the data and cleaned it superficially to test different models until we were able to decide which one was the most suitable and work on improving performance by adding and transforming features and tuning the models.

Tools used

The tools chosen for the development of the project have been the following:

- Operating systems:
    - **MacOS Ventura**: the operating system used for the development of the Fotocasa scraping code.
    - **Pop OS Linux**: the operating system used for the development of the Idealista scraping code.
- Software:
    - **Anaconda**: it is a free and open distribution of the Python and R languages, used in data science and machine learning. In this case it has been used in order to use Jupyter Notebook.
    - **Jupyter Notebook**: it is a web application for the creation and sharing of computational documents integrated into Anaconda. It allows the use of different programming languages, being Python the one chosen for the development of the project. The development of the code for scraping has been done using this tool.
- Programming languages:
    - **Python**: It is a programming language widely used in web applications, software development, data science, and machine learning (ML). The entire project has been developed in this programming language.
        - Environment manipulation: os
        - Scraping: BeautifulSoup, Selenium.
        - Data transformation and manipulation: Pandas, GeoPandas, Numpy, Re.
        - Data visualization: Matplotlib, Seaborn, Plotly, Streamlit.
        - Machine Learning: Scikit-learn, Shap, catboost, xgboost, lightgbm.
- Cloud based:

- **Google Drive**: it is a file hosting and synchronization service developed by Google where we downloaded and from where we uploaded the files generated by the project.
- **Google Colab**: it is a cloud based app, made by Google, that allows the development of Python code in the web browser and it is integrated with other Google tools like Google Drive.
- **GitHub**: it is a code hosting service for version control and collaboration. It is the service where we host the entire project.
- **Heroku**: it is a tool that allows the execution of different programming codes to be published in the cloud.

Data understanding

Next, we will explain how we obtained the data and what were the first results obtained as a first approximation using the features we worked on throughout the entire project.

Origin of the data

As previously mentioned, the data comes from the most visited real estate portals in Spain.

On one hand we find Fotocasa.es, a portal that belongs to the tech company Adevinta, also known for portals that belong to different sectors such as Infojobs or coches.net, as well as another real estate portal, Habitaclia.

On the other hand, we have Idealista.com, a competitor. It is a one-person limited company based in Madrid that works in the vacation rental sector through other companies such as Avaibook or Rentalia.

In both cases we find a similar structure as both real estate portals allow browsing through homes of different types classified according to various criteria. We can find properties for sale and for rent, not only homes, but also storage rooms, garages and other types. The properties are shown in different grids according to different administrative divisions of the Spanish territory: communities, provinces, municipalities, neighborhoods, etc. And finally one can access the page of each of these properties to find their characteristics.

For this project we have decided to work with the properties located in the municipality of Madrid, distinguishing them by neighborhood. This decision made us work with a static dataset with a specific amount of properties. To guarantee the scalability of the project, it would be necessary to automate the extraction of the

properties of each of the portals, which is a problem due to the scraping protections adopted by them.

Therefore, as we stated already, we assume that this project has only academic purposes at the moment, since the only way of scaling it is by using the portal's APIs and permission to automate data extraction. Said automation could allow us to know the estimated market prices of desired properties at different times of the year, depending on the time of data extraction, which would allow us to obtain prices that are better related to external variables such as inflation and thus represent the reality of the real estate market, since the predicted price, obtained with the data we have worked with, does not correspond to the future estimations.

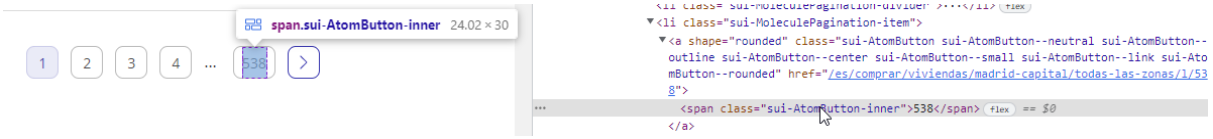
Fotocasa scraping

The data extraction from Fotocasa.es was carried out through web scraping at different times in order to increase the number of properties with which to train the subsequent model. This update has been done at close points in time in order to avoid learning contamination, although it cannot be excluded.

How does it work? Technical approximation

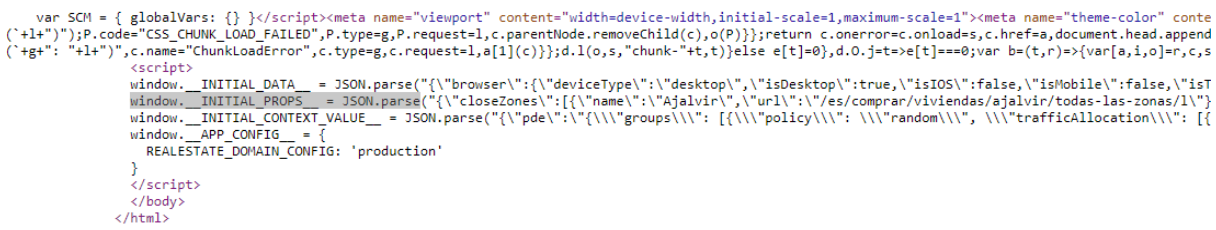
The Fotocasa.es web scraping code has been developed as a class in Python. The logic of its structure corresponds in a certain way to how the real estate portal itself is organized, and can be divided into functions that make the corresponding requests to the portal itself depending on the level of depth that you want to extract. Thus, in order to explain it in a simpler way, we can divide the code and functions as follows:

main: The main function of the class acts as an initializer, checking the parameters entered and determining one operation or another based on them.
pages_to_scrape: Thanks to the Selenium library, this function opens Chrome on the first page of the properties grid and performs different scrolls until the end of the page so as to dynamically load the content of the page, and then reads and saves the number of properties of the page so as to get the total sum of properties to scrape.

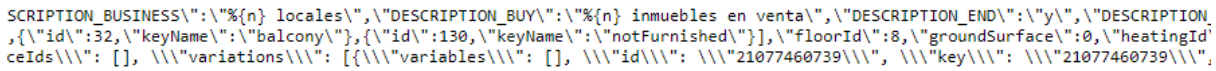


check_range: checks if the date range introduced is correct
divide_pages: this function sets five equal groups of pages to be checked during scraping whenever there are more than 10 pages. This allows the concurrency function to act later to read properties simultaneously in five groups in parallel and speed up data collection.

concurrent_scraping: function that reads properties data using up to five concurrent processes using the concurrent_futures library. For each of the processes the following function is initialized and the parallel scraping of the groups of page ranges passed as parameters begins.
scraping_loop: function that runs the scraping loop. Firstly, it collects the urls of each housing grid page and then passes them to the scraping function of each housing file.
property_list: In order to avoid server blocking, which happens if we use Selenium to cause the dynamic loading of the web in parallel, it was decided to use the BeautifulSoup library. The function extracts the code from each grid page passed as a parameter and creates a list of properties urls. To obtain the list of urls we realized that they are stored in json format in the footer of the code, which made it possible to do without Selenium, then the work consisted of cleaning the format with regular expressions until we could isolate the urls through css selectors.



parse_properties: having the list of previous urls, this function uses BeautifulSoup again to extract the json code of each property and clean it until it isolates its features..



check_features: before dumping the features into the DataFrame, this function verifies that the data exists by browsing the variable in json format corresponding to each property. If it does not find the data, it introduces a “^” character that will later be treated in data cleaning.
files_mode: if the file reading mode is enabled, this function reads the files within the default directories where the code saves them in download mode.
download_realestates: If the download mode is enabled, this function creates the files where each of the properties’ data is saved.

How to use it?

Initially, the implemented code has been developed in order to be reusable according to specific criteria, which are:

- province = <string>: name of the province from which we want to extract the data.
- page_range = [<int>,<int>]: range of grid pages from which we want to extract the properties. Can be set as null or an empty list if we want to extract everything.
- download = <bool>: enable download mode. This mode allows us to download a text file on our hard drive for each of the properties in order to be able to do subsequent readings from the local files without the need to access the website.
- files_mode = <bool>: enable file read mode. This mode allows the creation of the dataset only from the reading of the previously downloaded property text files.

There are, therefore, different ways of using the Fotocasa.es scraping class:

- province = “Madrid”: will extract the homes for sale in the municipality of Madrid.
- page_range = null / [1,10]: if we do not indicate the parameter, the scraping of the total pages available for the specified province will be performed. If we put a range, only those pages will be extracted, both included.
- (download = False, read_files = False): This is the default state. The script will make multiple calls to the website through all the properties and storing them in a data frame. It will not save any properties to the local drive or read any files from it.
- (download = True, read_files = False): the script will make multiple calls to the website through all the properties and will save them in local files inside the “fotocasa” folder. It will also create the DataFrame from memory.
- (download = False, read_files = True): The script will read the previously downloaded local property files to create the DataFrame.
- (download = True, read_files = True): the script will make multiple calls to the website through all the properties, save them to a local file and then read them to create the DataFrame.

The implementation of different modes allows us to work with files that have already been downloaded, increasing the speed with which we obtain the DataFrame if we want to recreate it in the future. In the same way, it has been developed in order to be reusable in extracting data from any province.

Web scraping issues

In any case, we must also take into account the possible blocks and restrictions that Fotocasa.es implements in the future and that might make this code useless.

In this specific case, the scraping of Fotocasa.es has not required the use of any type of proxy system, it was only enough to use rotating headers and add time.sleep of a few milliseconds between each call so that the web server does not reject the calls.

For example, the initial code used Selenium to obtain the url list from the grid, but this behavior had been recently blocked, forcing us to change the scraping technique.

Idealista scraping

This has also been developed as a class in Python and has been a painful development since the beginning. Selenium never really worked to access this website programmatically, because of the many protections that the portal has against such practice, so we had to look for a scraping API and ended up choosing www.scraperapi.com.

How does it work? Technical approximation

The script is fairly simple. It first gets to Madrid's properties' first page, then checks on the breadcrumb for the number of properties for each district. Using that information, it creates a dataset having, for each row, the area name, url and number of properties. Then it uses concurrency to go through each page of each area and retrieve the links of the properties.

This has to be this way because if it started directly from Madrid, it would have retrieved only 1.800 properties because the website limits the results to that number per each search that the user makes.

Once the full list of properties' links has been retrieved, it goes through each one, using again the concurrent futures module to speed up the process, and dumps the webpage html code as a text file. After that, another function retrieves data from each text file, still using the concurrent futures module, and saves it in a DataFrame.

Here is the list of methods of the class and what they do:

- *__init__*: this is the constructor. It receives the starting url as input and then simply set the Idealista hostname as a property of the class and sets up the scraper api endpoint and api key
- *_proxy_requests*: this method is being called by several functions. It receives any type of url as input, uses the scraper async api to scrape it and returns a dictionary with the response
- *_links_from_breadcrumb*: gets an area url as input and returns a list of links for all the areas (district or sub district) with 1.800 houses or less. If it finds

areas with more than 1.800 houses, it runs itself recursively. The data is obtained from the breadcrumb element of the HTML page.
- *get_areas_df*: It creates a list of urls for each area, using the *__links_from_breadcrumb* method. Writes a new class property named "areas_df" (pandas DataFrame)
- *generate_properties_links_df*: runs after get_areas_df as it needs areas_df to work. It iterates through each area stored in areas_df DataFrame, and extracts houses' links from each page of the area (district or sub district). It uses *_generate_single_area_property_links* and wraps it using concurrency. It creates a new class property "properties_links_df"
- *_generate_single_area_property_links*: used to retrieve the properties' links for a specific area.
- *get_properties_data*: runs after generate_properties_links_df. It takes the properties links and runs *_dump_single_property_data* to access them and dump the html as text files locally.
- *_dump_single_property_data*: gets a single property data and dumps it as text file
- *create_dataset*: gets the dumped html code saved as text files for all the properties and retrieves the properties' features from them. It creates a new class property "dataset"
- *_get_single_property_data*: retrieves the data of a single property. It is called by *create_dataset* that runs it using parallel processing
- get_location_ids_mapper: the location retrieved by *_get_single_property_data* for each property is expressed using Idealista IDs. This function scrapes Idealista to get the equivalent district or area name for each ID
- *full_scrape*: runs all the functions in the proper order but always starts from scratch

How to use it

The simplest way to use the code is to initialize an object of the class and then simply call the “full_scrape” function, no parameters required.

Since the scraping process is subject to problems that might be external to the code, such as API or target website downtimes or simply unexpected API failures, the code has been designed not to do a perfect scraping but rather to be able to restart it from where it left. The scraping has therefore the following phases, each corresponding to a function that can be called to resume the process from there:

1. Getting the links of the areas (function: *get_areas_df*): we first create a list of the areas inside the starting area (Madrid) which can be districts or sub districts. These areas have a link related to them and a number of grid pages

that have to be scraped to get the list of properties' links. All this information is saved in a csv called "area_df.csv"

2. Getting the link of the properties (function: *generate_properties_links_df*): reading the area_df.csv, it would go through each and every single page of every area, getting al the properties' urls it finds and updating the area_df.csv every time it completes the scrape of a page, so to start from the next page if something interrupts its execution. It saves the properties' urls in a csv named "properties_links_df.csv"
3. Getting the properties' pages source code (function: *get_properties_data*): reading the properties_links_df.csv, this function accesses each property url, retrieves its code and dumps it in a folder called "properties" naming each file with the ID of the property. It updates the properties_links_df.csv informing, for every record, if the dumping process ended properly or not.
4. Creating the dataset (function: *create_dataset*): this function goes through every dumped source code in the properties folder and retrieves the data from it, creating a pandas DataFrame as a result. It finally saves the dataset as a csv named "idealista_dataset.csv" in the datasets folder.

As it has been mentioned above, if the process gets stuck in one of these phases, it will be enough to run the function that corresponds to that phase and it will automatically resume from where it left.

Web scraping issues

As mentioned before, Idealista scraping had to be performed using a scraping API as it was impossible to extract the html code of the web pages using Selenium. The web scraping api used was www.scraperapi.com that, at the moment of writing this, grants 5.000 api credits at signup for 7 days, after which it defaults to a free tier that includes 1.000 api credits per month.

Features definition

After the data extraction from both real estate platforms, we had different features for each dataset depending on what was available. For this reason, it has subsequently been necessary to unify them by leveraging common features or transforming similar ones.

At the moment of extraction, the features from each of the platforms have been the following:

Fotocasa feature list

Name	Type	Description
origin	object	Origin of the dataset (Fotocasa).
page	int64	Extraction page number
title	object	Title as per property sheet
link	object	Property link
image_url	object	Link to the first photo of the property
country	object	Country of the property
district	object	District of the property.
neighborhood	object	Neighborhood of the property
street	object	Street of the property
zipCode	object	Zip Code of the property
province	object	Province of the property
buildingType	object	Property type (piso, chalet, etc.)
clientAlias	object	Advertiser’s name
latitude	object	Latitude of the property
longitude	object	Longitude of the property
isNewConstruction	object	Whether the property is a new construction
rooms	object	Bedrooms number
bathrooms	object	Bathrooms number
parking	object	Whether it includes a parking spot
elevator	object	Whether the building has an elevator
furnished	object	Whether the property comes furnished
surface	object	Expressed in square meters
energyCertificate	object	Energy certificate of the property

hotWater	object	How hot water is being provided
heating	object	Heating type
conservationState	object	Conservation state of the property
antiquity	object	How old is the property
floor	object	The floor of the property
surfaceLand	object	Size of the terrain that belongs to the property
otherFeatures	object	Other optional features of Fotocasa
price	int64	Expressed in Euros

Idealista feature list

Name	Type	Description
origin	object	Origin of the dataset (Idealista).
id	object	Id of the property
propertyType	object	Type of property (piso, chalet, etc.)
title	object	Title as per property sheet
description	object	Description as per property sheet
price	int64	Expressed in Euros
size	int64	Expressed in square meters
hasParking	int64	Whether the property has parking
roomNumber	int64	The number of bedrooms
bathNumber	int64	The number of bathrooms
hasSwimmingPool	int64	Whether the property has a swimming pool
hasTerrace	int64	Whether the property has a terrace
hasGarden	int64	Whether the property has a garden
hasLift	int64	Whether the building has an elevator
hasAirco	int64	Whether the property has air conditioning
hasFittedWardrobes	int64	Whether the property has fitted wardrobes

isGoodCondition	int64	Whether the property is in good condition
isNeedsRenovating	int64	Whether the property needs renovation
isNewDevelopment	int64	Whether the property is a new construction
energyCertification	object	The energy certification of the property
featureTags	object	Other optional features of Idealista
yearBuilt	int64	The year in which the property was built
orientation	object	The orientation of the property
heatingType	object	The heating type of the property
interiorExterior	object	Whether the property is interior or exterior
floor	object	The floor of the property
area_name	object	The district name

Final features

Given the difference in the features available in each of the real estate platforms and in order to unify both datasets into one, we selected a list of features available in both datasets.

But before that, it was necessary to carry out a small cleaning and transformation of the features of each dataset, which consisted of :

- Extract new interesting features from the additional fields of “otherFeatures” in Fotocasa and “featureTags” in Idealista.
- Create a unique ID for Fotocasa records by extracting it from the link.
- Unify the “size” feature in Fotocasa by adding the square meters of the house and the land.
- Drop the records related to properties outside the Madrid area in the Fotocasa dataset.
- Drop the unnecessary features in both datasets.

After that, we had the same features for each dataset with the same name, so we could join them to get a unique dataset to start the process of cleaning and exploring the data.

name	type	description
id	object	Id of the property
propertyType	object	Type of property (piso, chalet, etc.)
title	object	Title as per property sheet
price	int64	Expressed in Euros
size	int64	Expressed in square meters
hasParking	int64	Whether the property has parking
roomNumber	int64	The number of bedrooms
bathNumber	int64	The number of bathrooms
hasSwimmingPool	int64	Whether the property has a swimming pool
hasTerrace	int64	Whether the property has a terrace
hasGarden	int64	Whether the property has a garden
hasLift	int64	Whether the building has an elevator
hasAirco	int64	Whether the property has air conditioning
isGoodCondition	int64	Whether the property is in good condition
isNeedRenovating	int64	Whether the property needs renovation
isNewDevelopment	int64	Whether the property is a new construction
heatingType	object	The heating type of the property
floor	object	The floor of the property
energyCertificate	object	The energy certification of the property
district	object	The district name
dataset	object	Origin of the dataset (Idealista or Fotocasa)

Later on, these features will be transformed, which will be explained in the course of the project.

Data Preparation

	id	propertyType	title	price	size	hasParking	roomNumber	bathNumber	hasSwimmingPool	hasTerrace	...	hasLift	hasAirco	isGoodCondition	isNeedsRenovating	isNewDevelopment	energyCertification	heatingType	floor	district	dataset
5092	39149691	piso	piso en venta en calle matilde diez, 12	270000	50	0	1	1	0	0	...	1	1	1	0	0	unknown	Calefacción individual: Bomba de frío/calor	planta 2ª	Chamartín	idealista
27848	163835540	Flat	Piso en venta en Calle Marismas	198000	99	FEATURES.PARKING_PRIVATE	3	2	0	1	...	YES	1	1	0	False	G	NaN	0	Puente de Vallecas	fotocasa
10901	94559816	piso	piso en venta en jerónimos	1200000	158	0	3	3	0	0	...	1	1	1	0	0	e	Calefacción individual: Eléctrica	entreplanta	Retiro	idealista
27435	163849307	Flat	Piso en venta en Blasón	159000	86	NaN	3	2	0	1	...	NaN	0	1	0	False	G	NaN	6	Carabanchel	fotocasa
30822	164114008	Flat	Piso en venta en Silvio Abad	133000	58	NaN	2	1	0	0	...	NaN	1	1	0	False	G	NaN	0	Usera	fotocasa

Data cleaning

Since the dataset came from two different data sources, we observed that there were many discrepancies in the values that we had for each feature, so we had to review each of them in order to make them homogeneous and consistent.

Features

First of all, there are several groups of features that we can clearly differentiate.

- **Binary features:** ['hasParking', 'hasSwimmingPool', 'hasTerrace', 'hasGarden', 'hasLift', ‘hasAirco’, 'isGoodCondition', 'isNeedsRenovating', 'isNewDevelopment']

They are features whose values can only be 1 and 0, so the rest of the values must be transformed to their equivalents. At first glance, each real estate platform had its own naming to define the values, plus there were some NaNs values that needed to be cleaned up:

```
[ ]  # All these columns must be 1 or 0
     cols_to_clean = ['hasParking','hasSwimmingPool','hasTerrace','hasGarden','hasLift','hasAirco','isGoodCondition','isNeedsRenovating','isNewDevelopment']
     for col in cols_to_clean:
       print(col)
       print(df[col].unique())
```

```
hasParking
['0' '1' nan 'FEATURES.PARKING_PRIVATE' 'FEATURES.PARKING_COMMUNITY']
hasSwimmingPool
[0 1]
hasTerrace
[0 1]
hasGarden
[0 1]
hasLift
['0' '1' 'YES' nan]
hasAirco
[0 1]
isGoodCondition
[1 0]
isNeedsRenovating
[0 1]
isNewDevelopment
['0' '1' 'False' False]
```

As we can see in the image, the features that needed to be fixed were hasParking, hasLift and isNewDevelopment. So we decided to turn “NaN” and “False” values to 0 and every other categorical or YES value to 1.

- **Categorical features:** ['heatingType', 'floor', 'energyCertificate', 'propertyType', 'district']

We only listed those categorical features that are relevant to the model. There are others such as the id, title or the dataset source that are simply informative for the development of the analysis or that we will use for later transformations, but that have not been used for training the model.

Floor

This feature refers to the floor that the property occupies. Again, we find different nouns and some values that we must transform such as 'no info', 'does not apply', 'basement', 'entreplanta', 'bajo' and 'semi-sótano'. The rest are easily comparable using whole numbers.

```
[ ]  df['floor'].unique()
```

```
array(['no info', 'planta 3ª', 'planta 1ª', 'bajo', 'planta 5ª',
       'planta 4ª', 'planta 8ª', 'planta 9ª', 'does not apply',
       'planta 2ª', 'planta 6ª', 'planta 14ª', 'planta 7ª', 'planta 10ª',
       'entreplanta', 'semi-sótano', 'planta 12ª', 'planta -1',
       'planta 15ª', 'planta 20ª', 'sótano', 'planta 13ª', 'planta 11ª',
       'planta 16ª', 'planta 19ª', 'planta 18ª', 'planta 17ª',
       'planta -2', 'planta 40ª', 'planta 26ª', 'planta 21ª', '0', '7',
       '10', '8', '3', '11', '12', '6', '9', '15', '14', '2', '13', '5',
       '16', '4', '17', '18', '19', '20', '21', '1', 6, 10, 0, 7, 9, 8,
       11, 3, 12, 13, 5, 15, 14, 1, 16, 2, 4, 17, 18, 19, 21, 20],
      dtype=object)
```

To homogenize these values, we set 'sótano', 'semi-sótano' and 'entreplanta' to -1; 'bajo' as 0 and we just extracted the number for the rest of the values.

```
[ ]  # We only need the floor number
     df.loc[df['floor'].str.contains('sótano|sotano|semi-sótano|entreplanta', regex=True),'floor'] = '-1'
     df.loc[:,'floor'] = df['floor'].str.replace(r'planta|ª| ','', regex=True)
     df.loc[df['floor'] == 'bajo','floor'] = '0'
```

At this point we only had to take care of 'noinfo' and 'doesnotapply'. To do this, we observed what type of property they refer to:

```
[ ]  df.loc[df['floor'].str.contains('doesnotapply|noinfo'),['propertyType','floor','dataset']].value_counts()
```

```
propertyType  floor         dataset
piso          noinfo        idealista    1359
chalet        doesnotapply  idealista    1179
loft          noinfo        idealista       5
casa rural    doesnotapply  idealista       2
dtype: int64
```

Since 'doesnotapply' corresponds to houses (chalets and rural houses), we assume that floor will be 0. Then, we check the values of floor again:

```
[ ]  df['floor'].unique()
```

```
array(['noinfo', '3', '1', '0', '5', '4', '8', '9', '2', '6', '14', '7',
       '10', '-1', '12', '15', '20', '13', '11', '16', '19', '18', '17',
       '-2', '40', '26', '21'], dtype=object)
```

Finally, we checked how many records 'noinfo' corresponds to and, since it only affects 1364, we decided to delete them in order to have this feature with 100% valid data.

propertyType

This feature refers to the property type. Here as well, we found different namings to homogenize:

```
[ ]  df['propertyType'].unique()
```

```
array(['piso', 'chalet', 'loft', 'atico', 'casa rural', 'Flat'],
      dtype=object)
```

We simply changed 'Flat' to 'piso' and we already have all the same values, however, a new problem arises. If we take a look at the title of each of the properties, we see that there is a broader classification that we can use:

```
[ ]  df['propertyTypeTitle'] = df['title'].apply(lambda x: x.lower().split()[0])
     df['propertyTypeTitle'].unique()
```

```
array(['ático', 'piso', 'chalet', 'estudio', 'dúplex', 'casa', 'finca',
       'apartamento', 'planta', 'loft'], dtype=object)
```

This way, we decided to transform this field and expand it, obtaining the following classification:

```
df['propertyType'].unique()
```

```
array(['atico', 'piso', 'chalet', 'estudio', 'duplex', 'loft',
       'casa rural'], dtype=object)
```

district

This feature refers to the location of the property within Madrid. In this case we only detected a little naming mismatch for 3 district values.

```
[ ]  df['district'].unique()
```

```
array(['Villaverde', 'Tetuán', 'Barajas', 'Centro', 'Carabanchel',
       'Puente de Vallecas', 'Chamartín', 'Barrio de Salamanca',
       'Chamberí', 'Vicálvaro', 'Fuencarral', 'Retiro', 'San Blas',
       'Arganzuela', 'Ciudad Lineal', 'Moncloa', 'Villa de Vallecas',
       'Hortaleza', 'Latina', 'Usera', 'Moratalaz',
       'Fuencarral - El Pardo', 'Moncloa - Aravaca', 'Chamartin'],
      dtype=object)
```

```
[ ]  df.loc[df['district'] == 'Fuencarral - El Pardo','district'] = 'Fuencarral'
     df.loc[df['district'] == 'Moncloa - Aravaca','district'] = 'Moncloa'
     df.loc[df['district'] == 'Chamartín','district'] = 'Chamartin'
```

```
[ ]  df['district'].unique()
```

```
array(['Villaverde', 'Tetuán', 'Barajas', 'Centro', 'Carabanchel',
       'Puente de Vallecas', 'Chamartin', 'Barrio de Salamanca',
       'Chamberí', 'Vicálvaro', 'Fuencarral', 'Retiro', 'San Blas',
       'Arganzuela', 'Ciudad Lineal', 'Moncloa', 'Villa de Vallecas',
       'Hortaleza', 'Latina', 'Usera', 'Moratalaz'], dtype=object)
```

heatingType

This feature corresponds to the type of heating available in the property. First, we decided to match all null values to no info, obtaining the following cardinality of values:

```
[ ]  df['heatingType'].unique()
```

```
array(['no info', 'Calefacción individual: Gas natural',
       'Calefacción individual: Eléctrica', 'Calefacción central',
       'Calefacción central: Gas', 'Calefacción individual',
       'Calefacción individual: Bomba de frío/calor',
       'No dispone de calefacción', 'Calefacción central: Gasoil',
       'Calefacción individual: Gas propano/butano',
       'Calefacción central: Gas natural', 'Calefacción individual: Gas',
       'Calefacción central: Gas propano/butano',
       'Calefacción individual: Gasoil',
       'Calefacción central: Bomba de frío/calor', 'Gas Natural',
       'Electricidad', 'Gasóleo', 'Solar', 'Propano'], dtype=object)
```

In order to make those values consistent, we used regular expressions to get this final result:

```
[ ]  df.loc[:,'heatingType'] = df['heatingType'].str.lower()
     df.loc[:,'heatingType'] = df['heatingType'].str.replace(r'^Calefacci[óo]n.*:.','', regex=True)
     df.loc[df['heatingType'].str.contains('no info|no dispone de calefacción|calefacción central|calefacción individual'),'heatingType'] = 'no info/no calefacción'
     df.loc[df['heatingType'].str.contains('^propano$', regex = True),'heatingType'] = 'gas propano/butano'
     df.loc[df['heatingType'].str.contains('^gas natural$', regex = True),'heatingType'] = 'gas natural'
     df.loc[df['heatingType'].str.contains('^gasóleo$', regex = True),'heatingType'] = 'gasoil'
     df['heatingType'].unique()
```

```
array(['no info/no calefacción', 'gas natural', 'electricidad', 'gasoil',
       'solar', 'gas propano/butano'], dtype=object)
```

We also added a **new feature called hasHeatingInfo** with binary values in order to specify whether each record has heating information or not.

energyCertificate

This feature corresponds to the energy efficiency classification of the properties. In this case, it was enough to change the values that do not correspond to an energy certificate to "no info".

```
[ ]  df['energyCertification'].unique()
```

```
array(['inProcess', 'd', 'g', 'e', 'b', 'c', 'unknown', 'f', 'a',
       'exempt', 'G', 'E', 'F', 'A', 'C', 'B', 'D', '^'], dtype=object)
```

```
[ ]  df.loc[:,'energyCertification'] = df['energyCertification'].str.replace(r'inProcess|unknown|\^|exempt','no info', regex=True)
     df.loc[:,'energyCertification'] = df['energyCertification'].str.upper()
```

```
[ ]  df['energyCertification'].unique()
```

```
array(['NO INFO', 'D', 'G', 'E', 'B', 'C', 'F', 'A'], dtype=object)
```

Duplicate records

Although basic duplication cleaning had already been done, the nature of the dataset required a more in-depth review of this type of case.

To increase the number of records, different scrapings have been performed on Fotocasa. That resulted in having different datasets and the possibility of having duplicate properties whose price has been updated. For example:

```
[84] df[df['id']=='164816170']
```

	id	propertyType	title	price	size	hasParking	roomNumber	bathNumber	hasSwimmingPool	hasTerrace	...	hasLift	isGoodCondition	isN
17219	164816170	piso	Piso en venta en Calle de la Serradilla	168800	75	0	3	1	0	1	...	1	0	
21277	164816170	piso	Piso en venta en Calle de la Serradilla	167000	75	0	3	1	0	1	...	1	0	
22507	164816170	piso	Piso en venta en Calle de la Serradilla	168000	75	0	3	1	0	1	...	1	0	
23391	164816170	piso	Piso en venta en Calle de la Serradilla	165000	75	0	3	1	0	1	...	1	0	

4 rows × 21 columns

Dealing with this type of duplicates was easy: we could spot these duplicates by simply looking at the duplicated IDs. Counting the property ID values, we spotted the duplicated records and solved these duplicities by simply keeping the most recent one, usually the last, as it was the last added, and dropping the others.

```
[87] checking_duplicated_properties = df.groupby(['propertyType','price','size','district'])['id'].apply(list).reset_index()
     checking_duplicated_properties['id_count'] = checking_duplicated_properties['id'].apply(len)
     checking_duplicated_properties = checking_duplicated_properties[checking_duplicated_properties['id_count']>1].sort_values('id_count',ascending=False)
     checking_duplicated_properties
```

	propertyType	price	size	district	id	id_count
16912	piso	1080000	129	Barrio de Salamanca	[98049627, 98022393, 98022245, 98232507, 98278...	44
6930	piso	191000	106	San Blas	[95912951, 96096366, 97206804, 97524700, 16493...	40
12662	piso	425000	49	Centro	[98100577, 98059607, 98091122, 98187628, 98306...	36
16371	piso	889000	111	Barrio de Salamanca	[97324554, 97418278, 98377390, 98061525, 97199...	32
16223	piso	859000	115	Centro	[97672768, 98094650, 97639117, 98106782, 97626...	26
...	...	...	...	...	...	...
7552	piso	210000	45	Arganzuela	[90723092, 156817213]	2
7567	piso	210000	54	Retiro	[98084408, 97952825]	2
7569	piso	210000	55	San Blas	[25791163, 97475791]	2
7572	piso	210000	57	Retiro	[97139409, 163002859]	2
9291	piso	260000	93	Moratalaz	[98413957, 164569880]	2

3473 rows × 6 columns

We also detected another type of duplicates: the same properties being uploaded multiple times (usually by different real estate agencies).

To spot this type of duplicates, we made a “group by” using specific properties’ characteristics: 'propertyType', 'price', 'size’, 'district' and ‘title’. The reason behind this was simple: we thought it was not likely at all to have different properties sharing

the same price, size, type, title and located in the same district as well. This was also confirmed after a thorough check through several sheets of properties that had these common characteristics, seeing by the photos that they were all the same property uploaded multiple times but with different zeal when it came to filling the information properly (e.g. some sheets showed the same house having less bedrooms or bathrooms or not mentioning the parking lot anywhere but in the description).

To deal with these duplicates we created a feature called “temp_score”, which was a simple sum of all the numerical features leaving out pricing and size, this way we could see which of the duplicated properties had more numerical features filled and keep it, dropping the others.

```
[93] df_duplicated_titles = pd.DataFrame(df.reset_index().sort_values('price',ascending=False).groupby(by=['title','size','district','temp_score','roomNumber
     df_duplicated_titles['index_count'] = df_duplicated_titles['index'].apply(len)
     df_duplicated_titles = df_duplicated_titles[df_duplicated_titles['index_count'] > 1].sort_values('index_count',ascending=False)
     df_duplicated_titles
```

	title	size	district	temp_score	roomNumber	bathNumber	dataset	index	index_count
14788	piso en venta en calle laguna del marquesado, 4	49	Villaverde	5	1	1	idealista	[12159, 4664, 5631, 4023, 4999]	5
9993	piso en venta en calle cerro del campo, 31	95	Vicálvaro	10	3	2	idealista	[14876, 13682, 2426, 14788, 10780]	5
1242	Piso en venta	73	Carabanchel	7	2	2	fotocasa	[26463, 18659, 23129, 19999]	4
7176	chalet adosado en venta en calle cerro minguet...	260	Fuencarral	11	3	3	idealista	[7730, 5600, 9234, 4395]	4
7168	chalet adosado en venta en calle arte figurati...	135	Villa de Vallecas	10	3	3	idealista	[15688, 3731, 4397, 2497]	4
...	...	...	...	...	...	...	...	...	...
4513	Piso en venta en Calle de Vigo, 5	143	Retiro	12	4	2	fotocasa	[18963, 25534]	2
4496	Piso en venta en Calle de Valle Inclán	95	Latina	7	3	1	fotocasa	[19121, 25857]	2
4480	Piso en venta en Calle de Ulises	79	Hortaleza	10	2	2	fotocasa	[21879, 24671]	2
4453	Piso en venta en Calle de Trafalgar	35	Chamberí	5	1	1	fotocasa	[27309, 17424]	2
20175	ático en venta en recoletos	200	Barrio de Salamanca	11	3	4	idealista	[201, 2493]	2

491 rows × 9 columns

The last duplicate type was represented by houses showing equal values in an even wider amount of characteristics, except the price which was slightly different. We made a “group by” of the following features: 'title', 'size', 'district', 'temp_score', 'roomNumber', 'bathNumber', 'dataset', 'floor'. After that we realized that there were almost 750 properties that shared such characteristics but with different prices.

Again, we checked thoroughly several properties’ sheets and saw that all of them were actually different properties. They usually belong to the same building or complex and were (or still are) being advertised by the same real estate promoter or agency. There is no way to determine algorithmically whether there are some real duplicates but we tend to deny this hypothesis as it would make little sense to announce the same property with different prices.

Wrong values

Finally, we had to deal with human errors. We found erroneous values that were simply impossible or had been extracted through scraping before being corrected.

The only viable way was to analyze the properties manually, looking at minimums and maximums, as well as anomalous values, for example properties with more than 20 bathrooms, with strange size considering the property type, price or size set to 0, etc.

We firstly dropped properties having 0 price and 0 size, as we considered those as impossible values.

	count	mean	std	min	25%	50%	75%	max
price	20241.0	576626.543896	761212.417927	0.0	180000.0	320000.0	650000.0	13000000.0
size	20241.0	123.737217	124.458167	0.0	64.0	90.0	134.0	2400.0

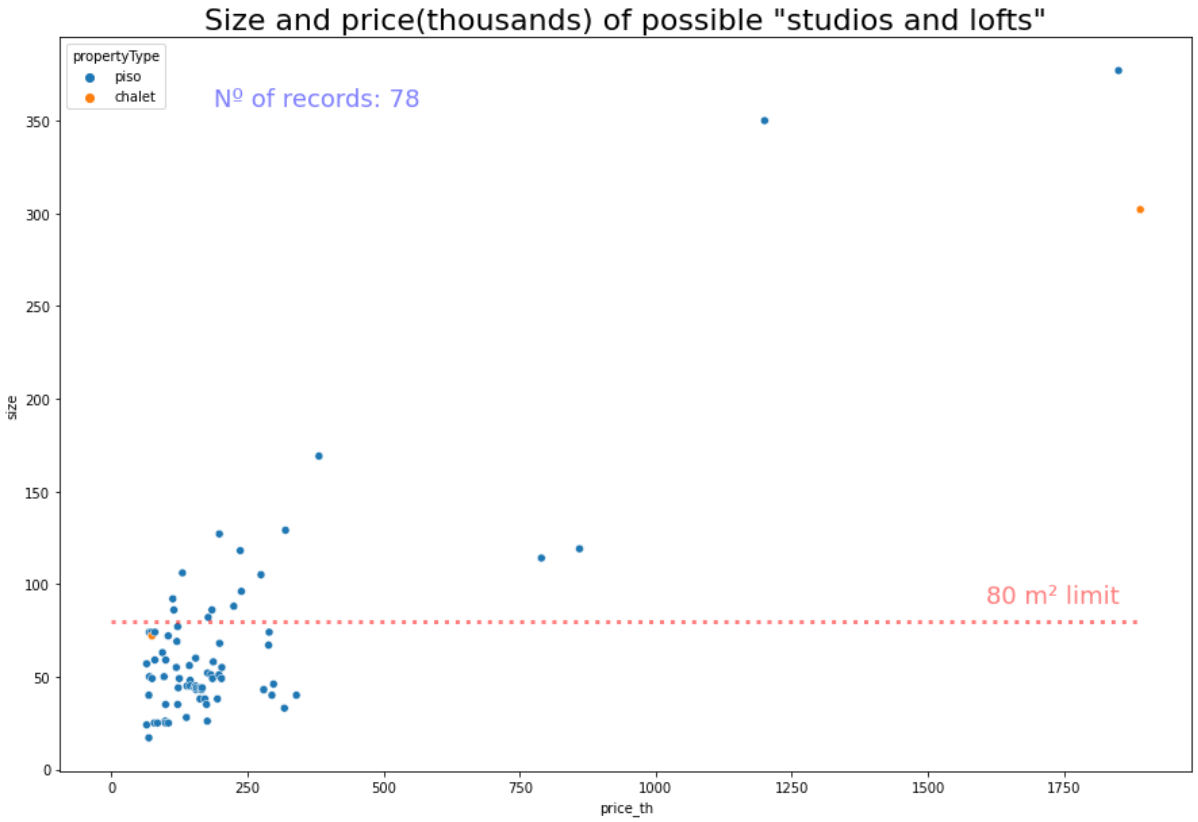
Before dropping them, we checked how many they were and made sure their amount was irrelevant.

```
[252] len(df[(df['price']==0)])
      18

[253] len(df[(df['size']==0)])
      13

[107] df = df[(df['size']>0) & (df['price']>0)]
```

On the other hand, we also found properties without bathrooms or bedrooms, but this case required further analysis since there may be flats with such characteristics, called studios and lofts.

roomNumber	20209.0	2.734178	1.601538	0.0	2.0	3.0	3.0	109.0
bathNumber	20209.0	1.829680	1.198148	0.0	1.0	2.0	2.0	21.0

Therefore, we established that this type of property, with up to 1 bathroom and 0 bedrooms (excluding penthouses and duplexes), should not exceed 80 square meters.

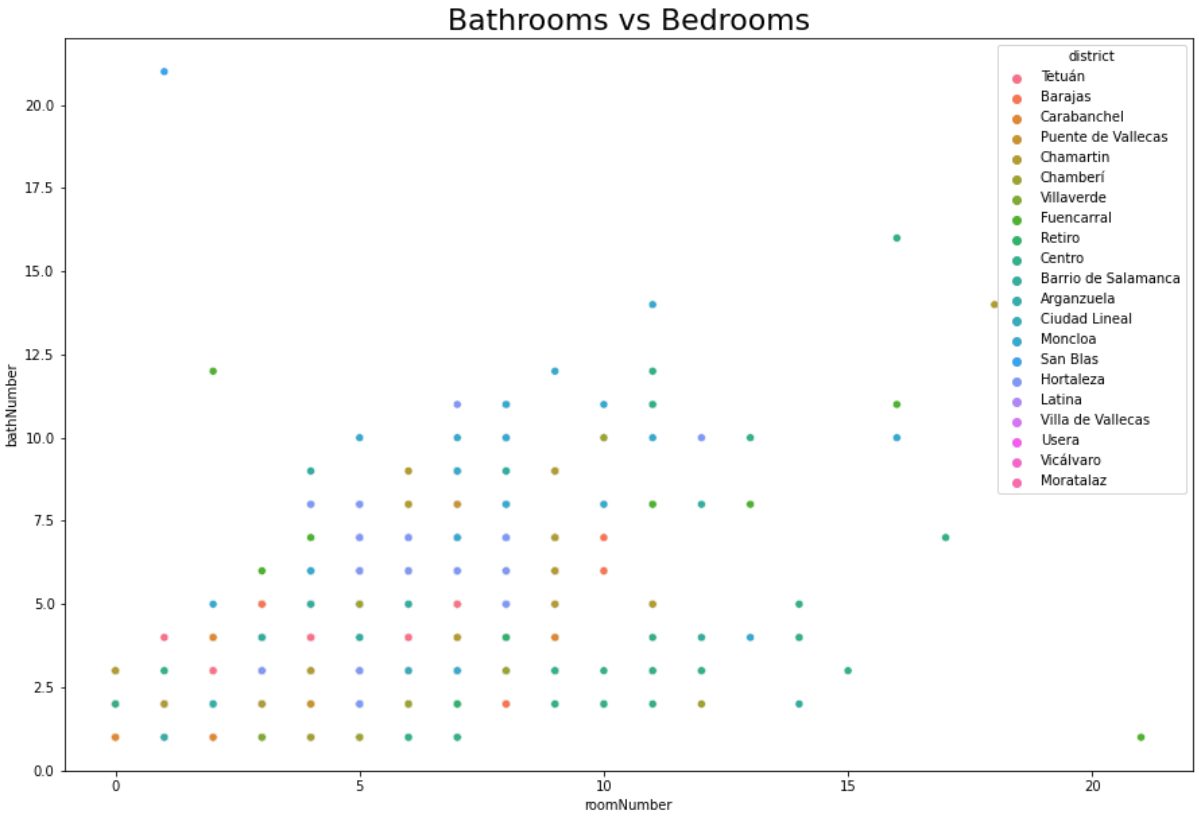


However, we can see how properties of different types are meeting these criterias.

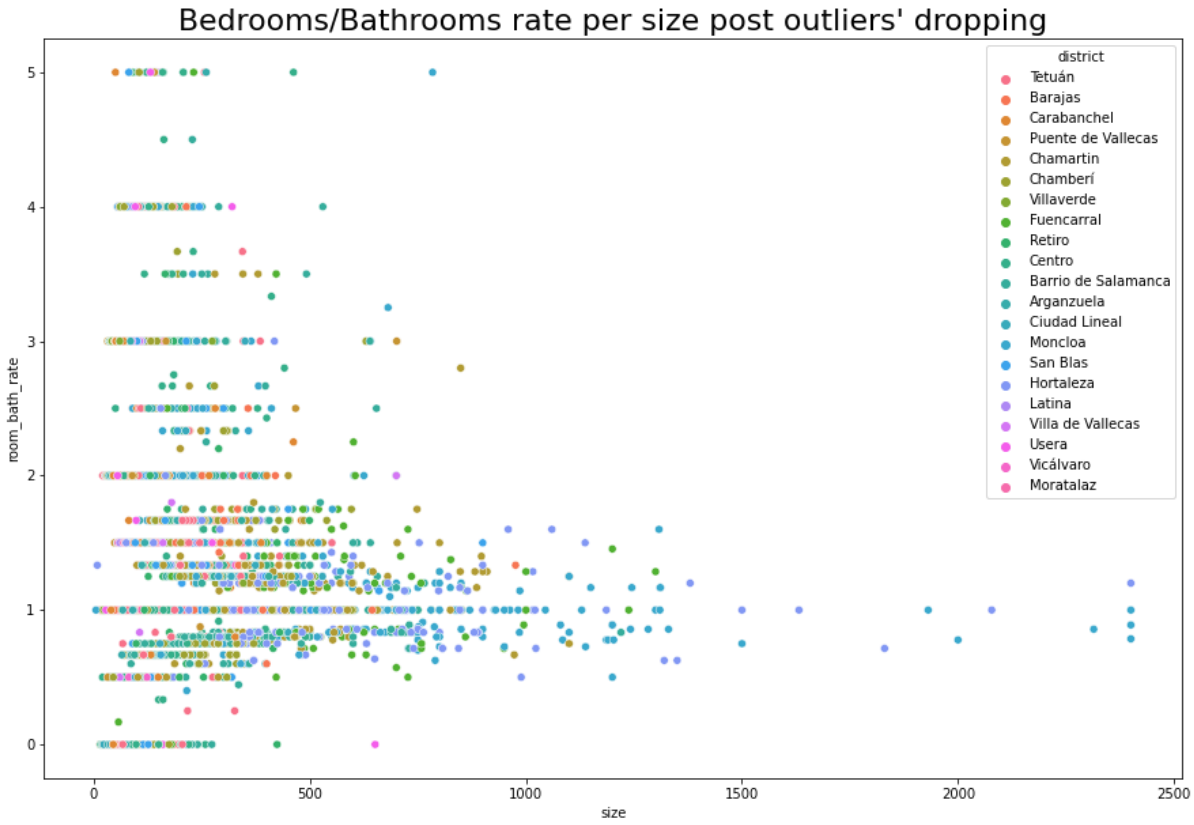
```
[ ]  possible_studio_loft['propertyType'].value_counts()

     piso      76
     chalet     2
     Name: propertyType, dtype: int64
```

Therefore, we decided to eliminate all the chalets that met these conditions, since we assume that it is an obvious error since we cannot have a chalet without rooms. Similarly, we also removed any property that does not have at least 1 bathroom.

Continuing with the bathrooms and the rooms, we find strange cases such as properties with 20 rooms and 1 bathroom or more than 20 bathrooms and 1 room.



To analyze these cases, it was decided to create a new metric, a ratio between both features that allowed us to see more clearly the relationship between them. At this point we decided that the acceptable ratios were between 5 and 0.5, which means that every house must have at least 1 bathroom and at best 5 bedrooms for each bathroom, dropping some records. This cleanup allowed us to normalize the relationships between both features.



Finally, we note that there are properties with a size of less than 10 square meters, so we decided to set a lower limit of at least 20 square meters.

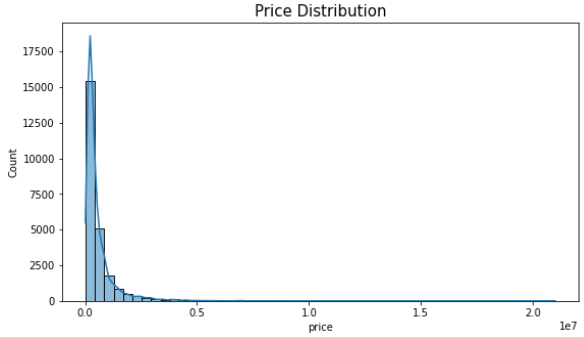
Data exploration and feature analysis

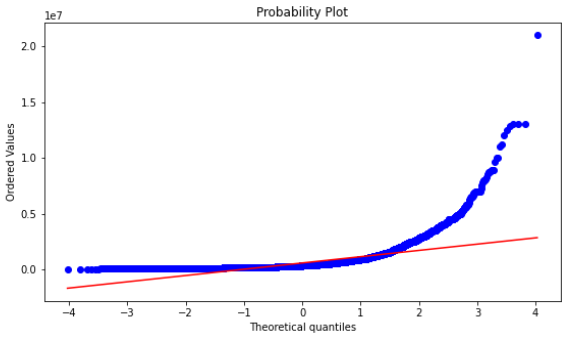
Distribution analysis and anomalies handling

Since the data had been obtained through scraping, it is highly affected by anomalies, caused by the very nature of the published properties or even by human errors when entering the characteristics of each one, since neither Fotocasa nor Idealista have any kind of data validation.

We must also take into account that in the initial dataset we have properties of a very different nature, from luxury villas to houses in humble neighborhoods. And it is these extremes that require our attention because of the possibility that they may harm future predictions. It also affects us the presence of properties ready for rent or offices, impossible to identify if not manually on real estate platforms due to the features that we have.

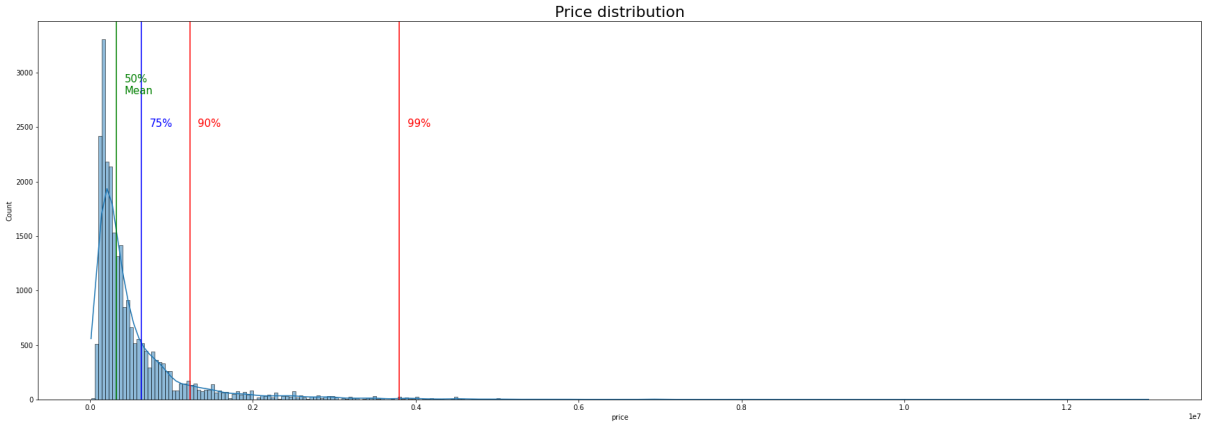
To analyze this fact, we checked the distribution offered by our dependent variable: the price.



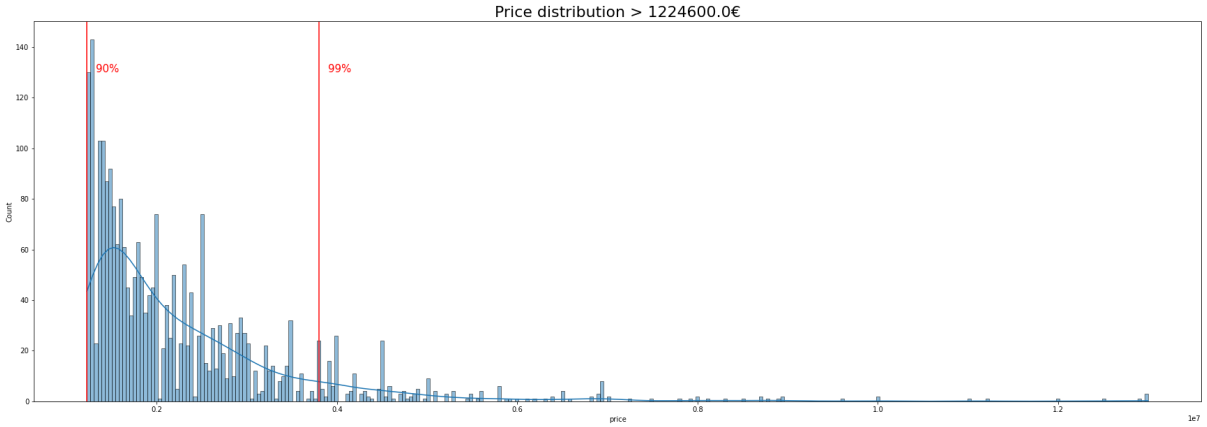


As can be seen, we see a positive skewness, with the largest number of homes concentrated in prices below one million euros and the upper tail lengthening considerably, something that we can also see in the probability graph compared to a normal distribution.

If we focus more on this price distribution:



We can see how from the 90% quantile (€1.224.600) the properties begin to be residual and, from the 99% quantile (€3.800.000) they are already scarce.



But what characteristics do these properties have? First of all, flats are the most abundant, since it is the type of property that stands out the most in the dataset. In second place, we have the chalets, something that fits the price, followed by penthouses and duplexes. The data is reasonably acceptable.

```
[ ]  df[df['price'] > df_desc.loc['90%']['price']][['id','propertyType']].groupby(by='propertyType')['id'].count()
```

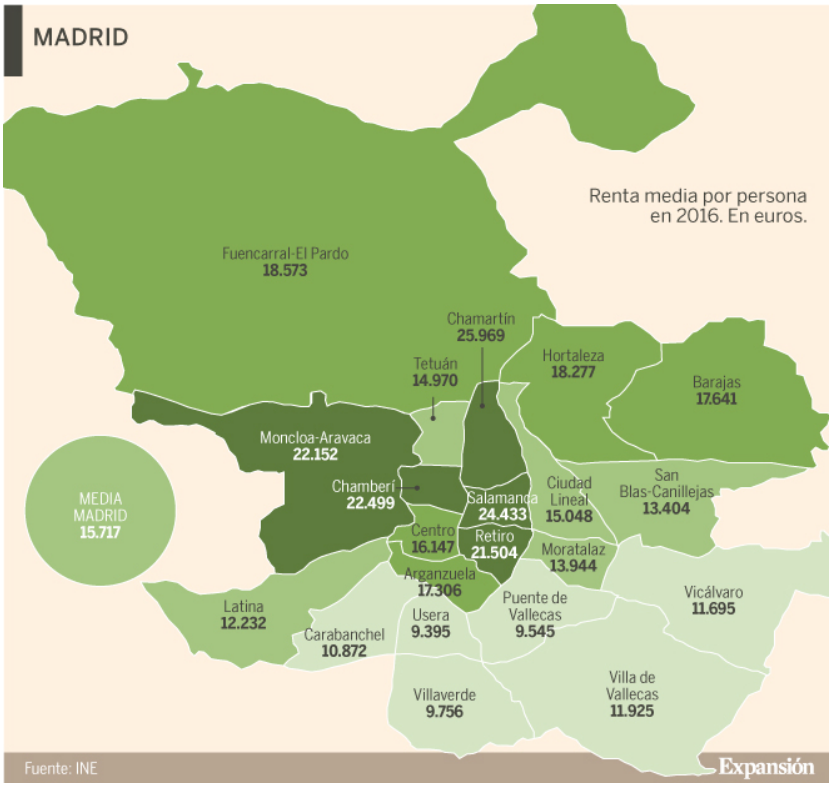
```
propertyType
chalet      456
dúplex      127
estudio       1
finca         1
piso       1248
ático       155
Name: id, dtype: int64
```

If we take a look at the locations of the most expensive flats:

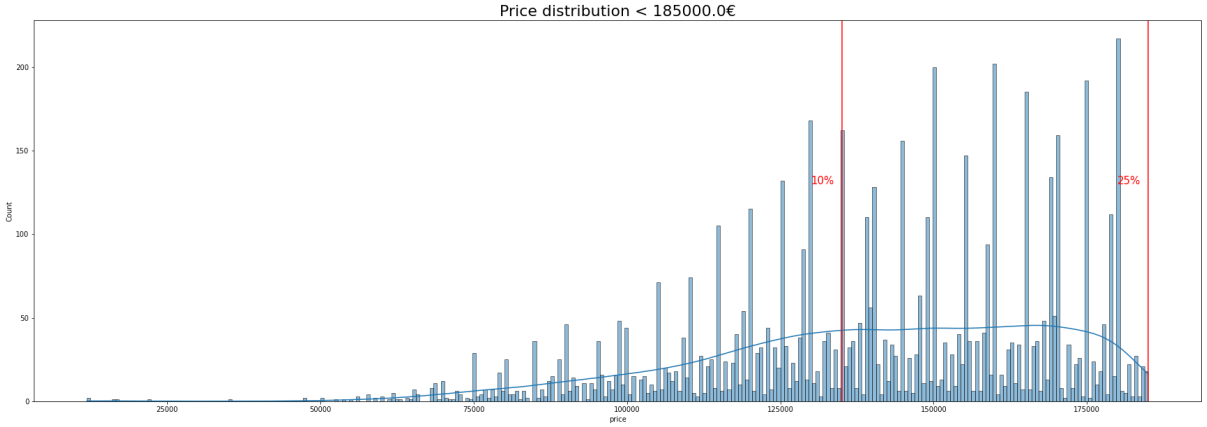
```
[ ]  df[(df['price'] > df_desc.loc['90%']['price']) & (df['propertyType'] == 'piso')][['id','district']].groupby(by=['district'])['id'].count()
```

```
district
Arganzuela              1
Barajas                 1
Barrio de Salamanca    492
Centro                 139
Chamartin              154
Chamberí               272
Ciudad Lineal           12
Fuencarral               2
Hortaleza               16
Moncloa                 72
Retiro                  63
Tetuán                  24
Name: id, dtype: int64
```

We clearly see how they are in the neighborhoods where the upper-middle social classes reside, such as the Barrio de Salamanca, Chamberí and Chamartín, although we also find Centro, with a lower average income compared to the rest. Something that we can contrast in the following choropleth map extracted from the newspaper Expansión with the average income per Madrid neighborhood according to the INE in 2016 (some years ago, but still valid today).



If we look at the lower end, we don't find too many surprises either:



The propertyType with the highest frequency is, unsurprisingly, flats. Still we observed a high amount of chalets, which was unexpected. However those are semi-detached or small houses, located in districts such as Vallecas, Tetuán Usera.

```
[ ]  # Chalet
     df.loc[(df['price'] < df_desc.loc['10%']['price']) & (df['propertyType'] == 'chalet'),'district'].value_counts()
```

```
Puente de Vallecas    8
Tetuán                3
Usera                 2
San Blas              1
Carabanchel           1
Vicálvaro             1
Centro                1
Name: district, dtype: int64
```

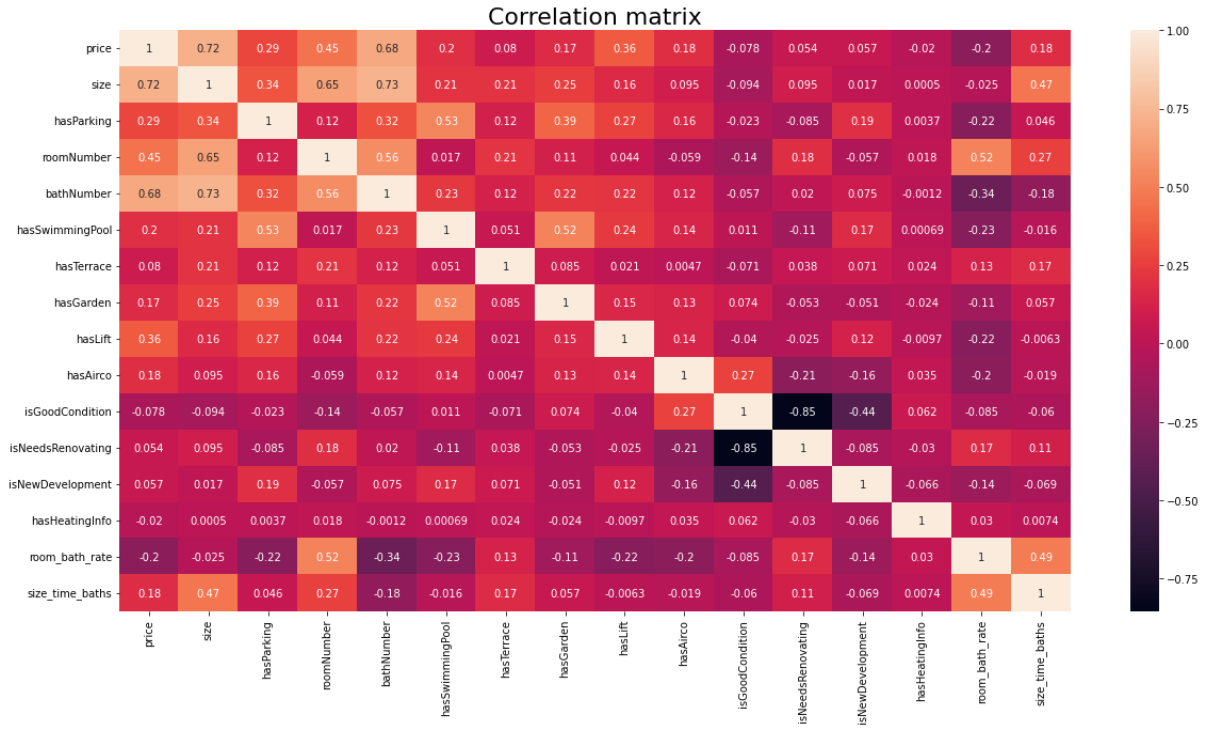
Finally, due to the large difference in prices we found at the upper end and after checking that many of the properties can be considered luxury or for commercial purposes (hostels or offices), we decided to reduce the dataset and keep those records with a price below or equal to the 90% percentile (€1,224,600).

```
print(f"We currently have {df.shape[0]} records")
print(f"We would lose the top {df[df['price'] > df_desc.loc['90%']['price']].shape[0]} records according to price")
print(f"We would keep {df[df['price'] <= df_desc.loc['90%']['price']].shape[0]} records")
```

```
We currently have 24655 records
We would lose the top 2466 records according to price
We would keep 22189 records
```

Features correlation

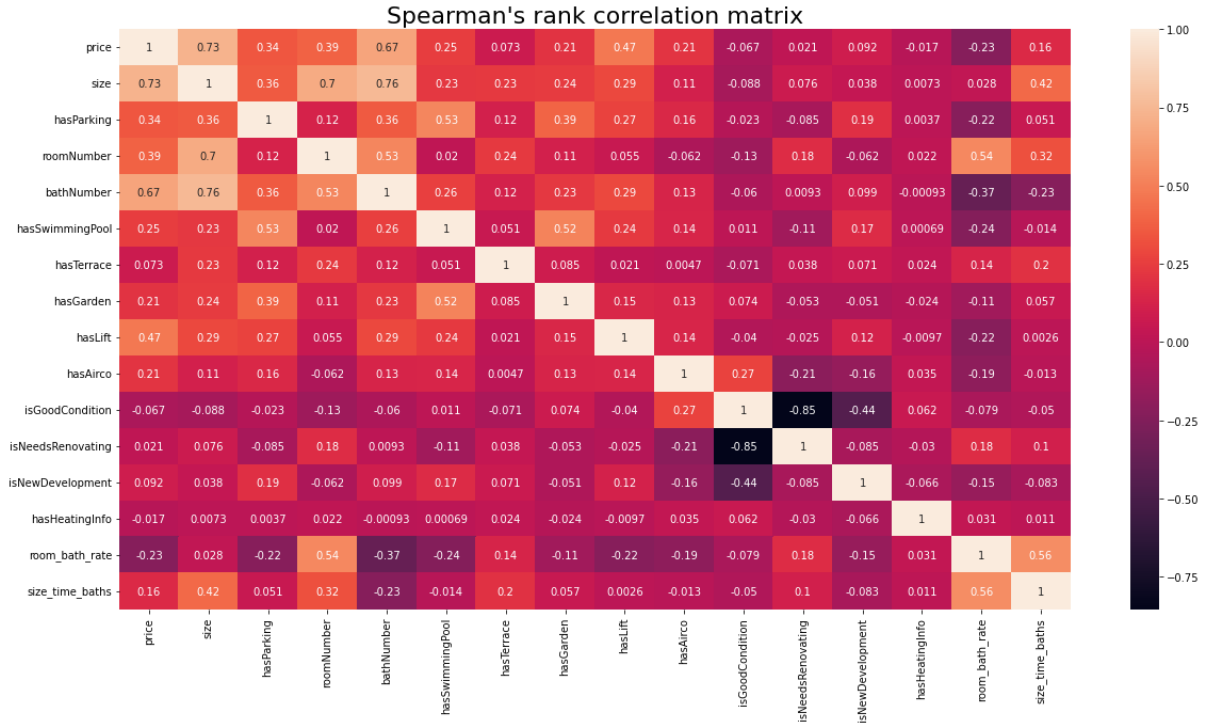
Through a correlation matrix (with Pearson correlation), we can begin to analyze the homoscedasticity and heteroscedasticity of the available features. Although, we won't start removing features until later, when we can correlate both numeric and categorical ones.



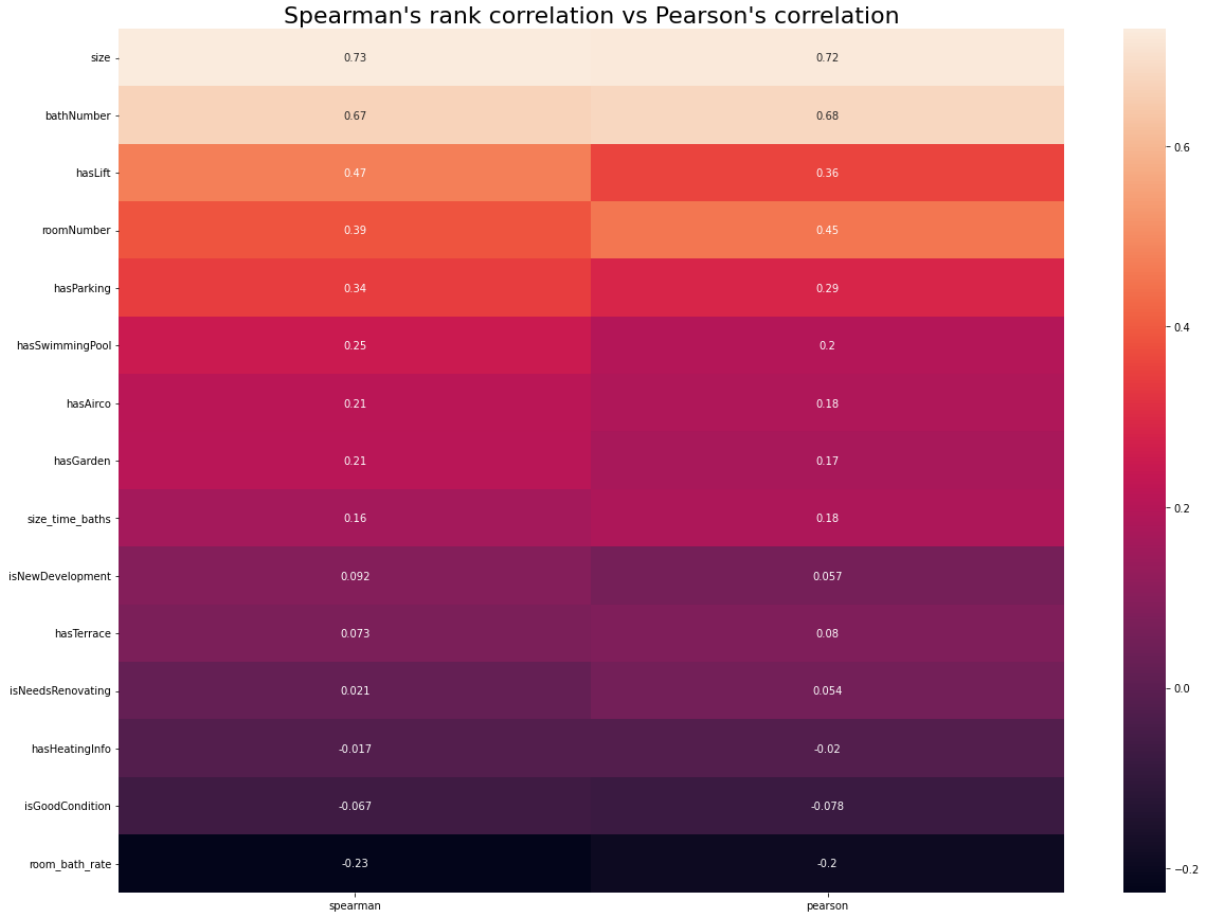
Looking at the matrix, we could not see any major correlation among features. The only thing that stands out is the correlation between isGoodCondition and isNeedsRenovating, something logical given that if a property is in good conditions, it will not need any renovation. We decided to analyze these two features later on.

If we look at the correlations with the dependent variable, we see equally logical behaviors. The size or number of rooms and bathrooms are the features that most affect the price, followed by whether the property has an elevator or parking.

Now let's look at Spearman's rank correlation, which measures the strength and direction of the association between two variables. This is non-parametric and more robust against outliers.



In the case of the Spearman matrix we see correlations very similar to those of the Pearson matrix. If we compare the correlations of both matrices with the dependent variable:



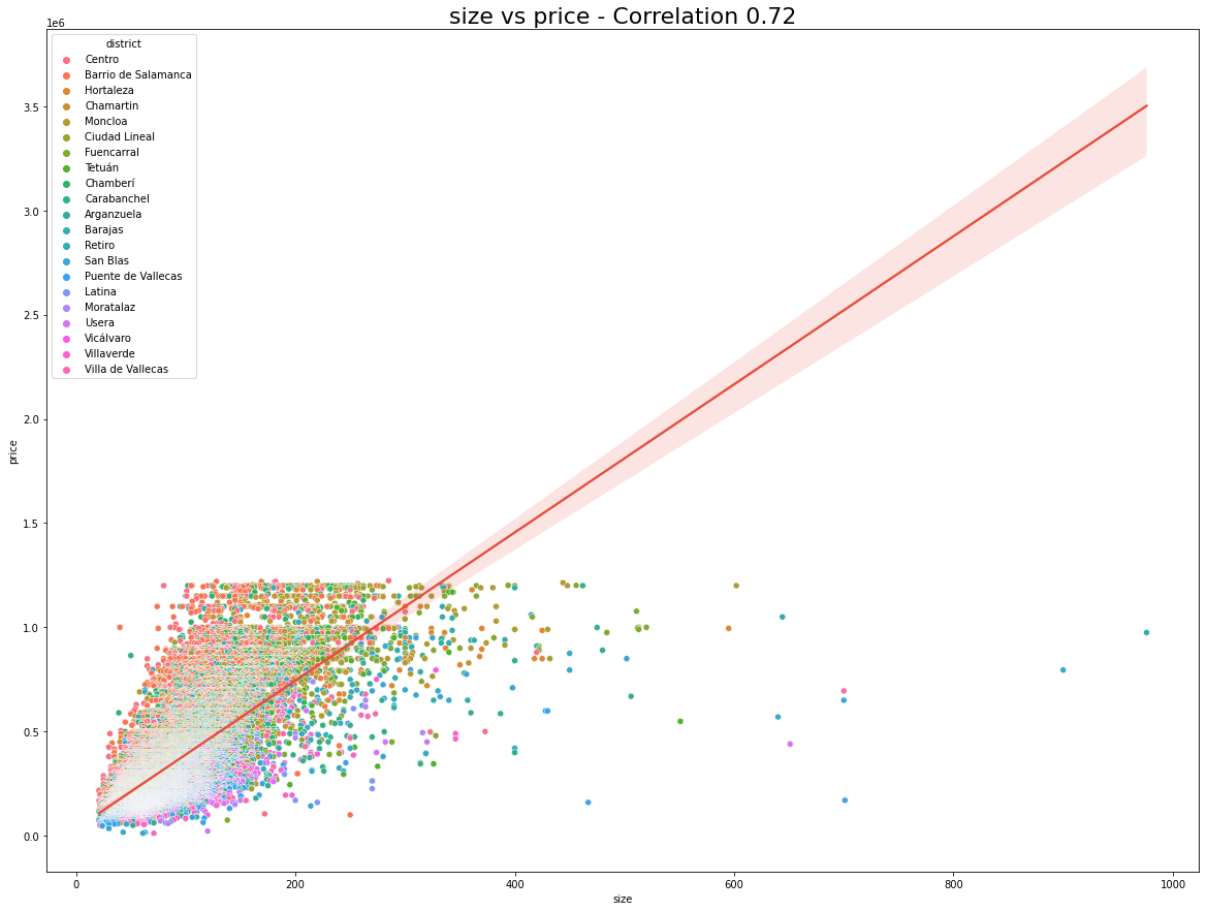
We observe how both matrices present very similar correlations with price. The most notable difference would be that in Spearman's matrix, 'hasLift' and 'hasParking' are given greater relevance than Pearson's.

Features analysis

At this point, we are going to delve deeper into the relationship that the different features have with the target in order to discover the reasons for the correlations or if there are situations to take into account for the future model.

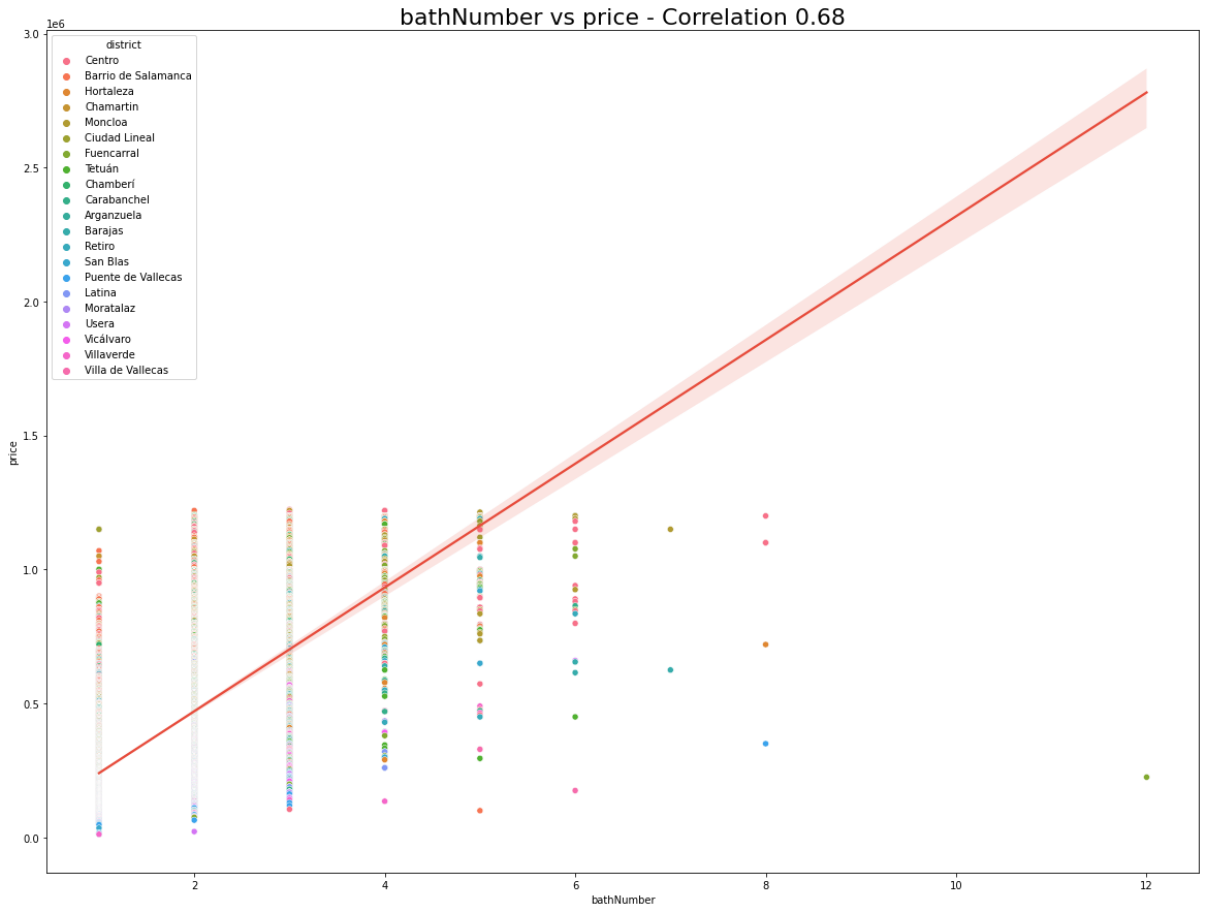
To do this, we again separate the features into two categories. First, the numerical features, which are 'size', 'bathNumber', 'roomNumber', 'hasParking', 'hasTerrace', 'hasLift', 'hasAirco', 'hasGarden', 'hasSwimmingPool', 'floor', ' hasHeatingInfo', 'isNeedsRenovating', 'isGoodCondition', and 'isNewDevelopment'. On the other hand, the categorical ones, made up of 'district', 'energyCertification', 'heatingType' and 'propertyType'.

Let's start with the numerical ones:



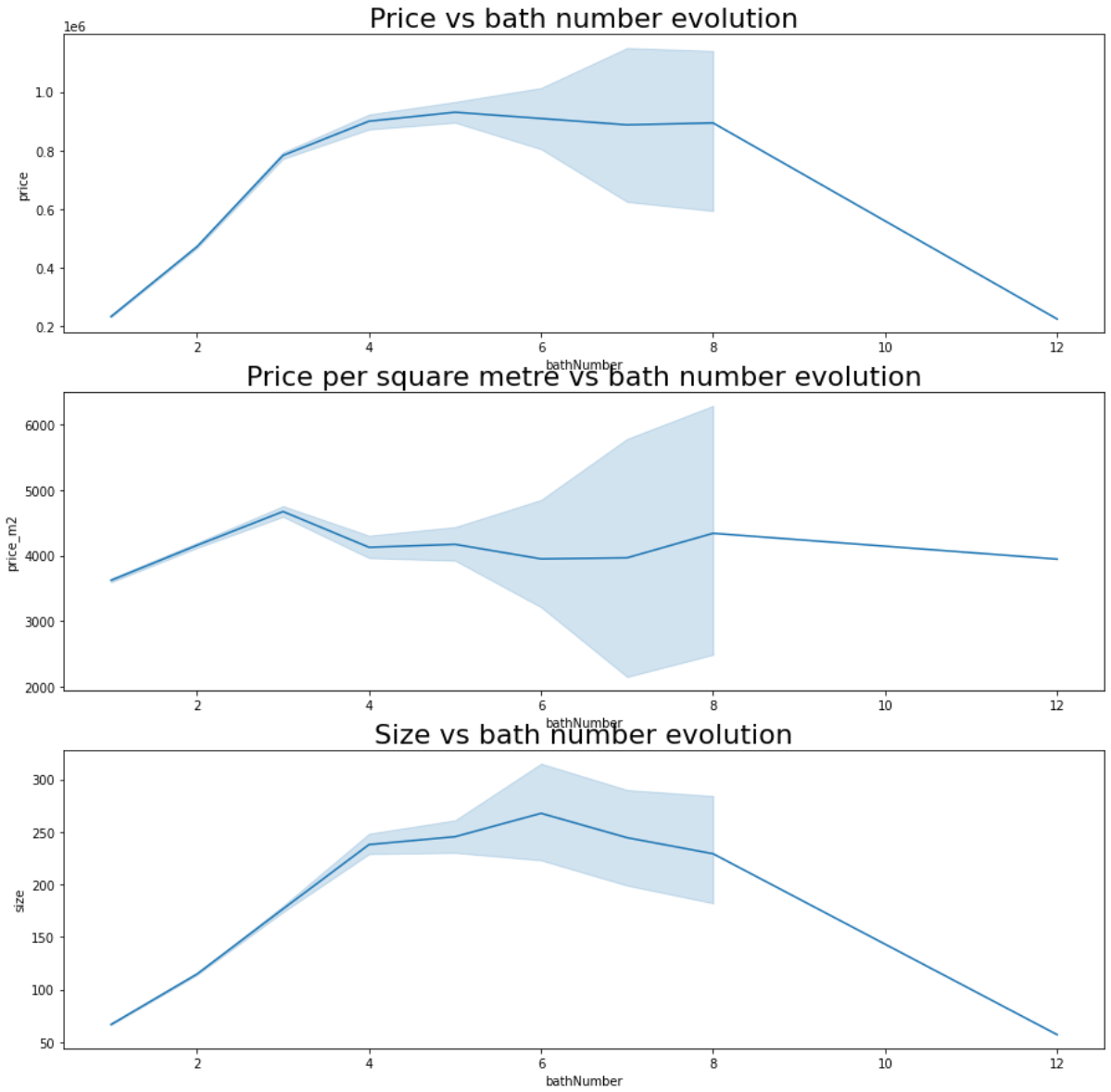
The first thing that draws attention is the cut that occurs around €1,200,000, this is due to the previously explained decision to limit the properties to the price marked by the 90% percentile.

Beyond that, there is a strong correlation between 'size' and price, where we can also see how the 'district' categorical feature influences a lot, with those with higher incomes standing out at the price level, but also scaling in a different way.



Now we will look at the correlation between 'bathNumber' and the price, something that we have found curious in the previous correlation matrices. 'bathNumber' has a strong correlation with the dependent variable, making it appear that the more bathrooms a property has, the more expensive it is.

This made us wonder how many bathrooms a normal house should have and if, again, possible hostels or properties with different characteristics are coming into play. In addition, it is very striking that there is a property with 12 bathrooms.

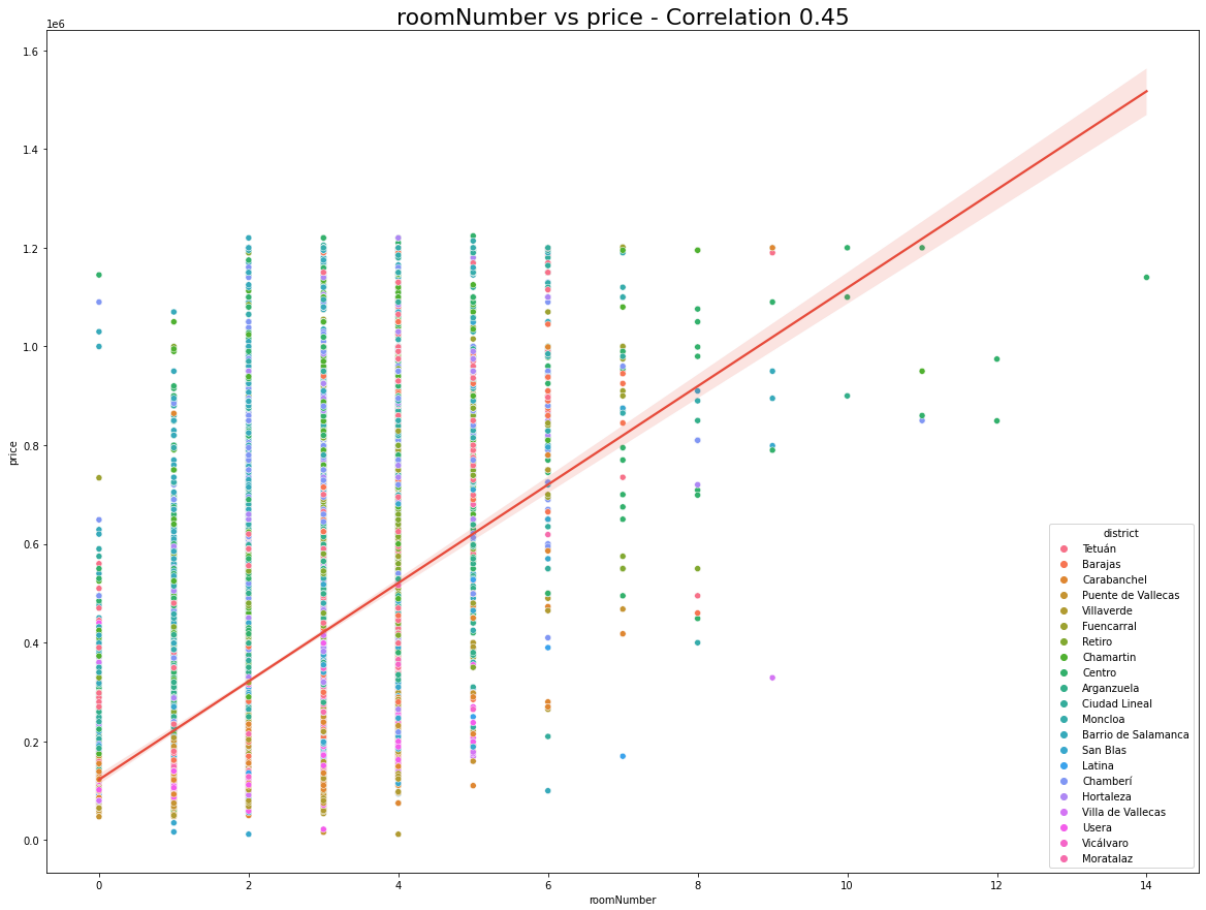


If we take a look at the evolution of 'bathNumber' regarding price, price per square meter and size, we clearly see how the 12 bathrooms mentioned above is an outlier that has to be dropped.

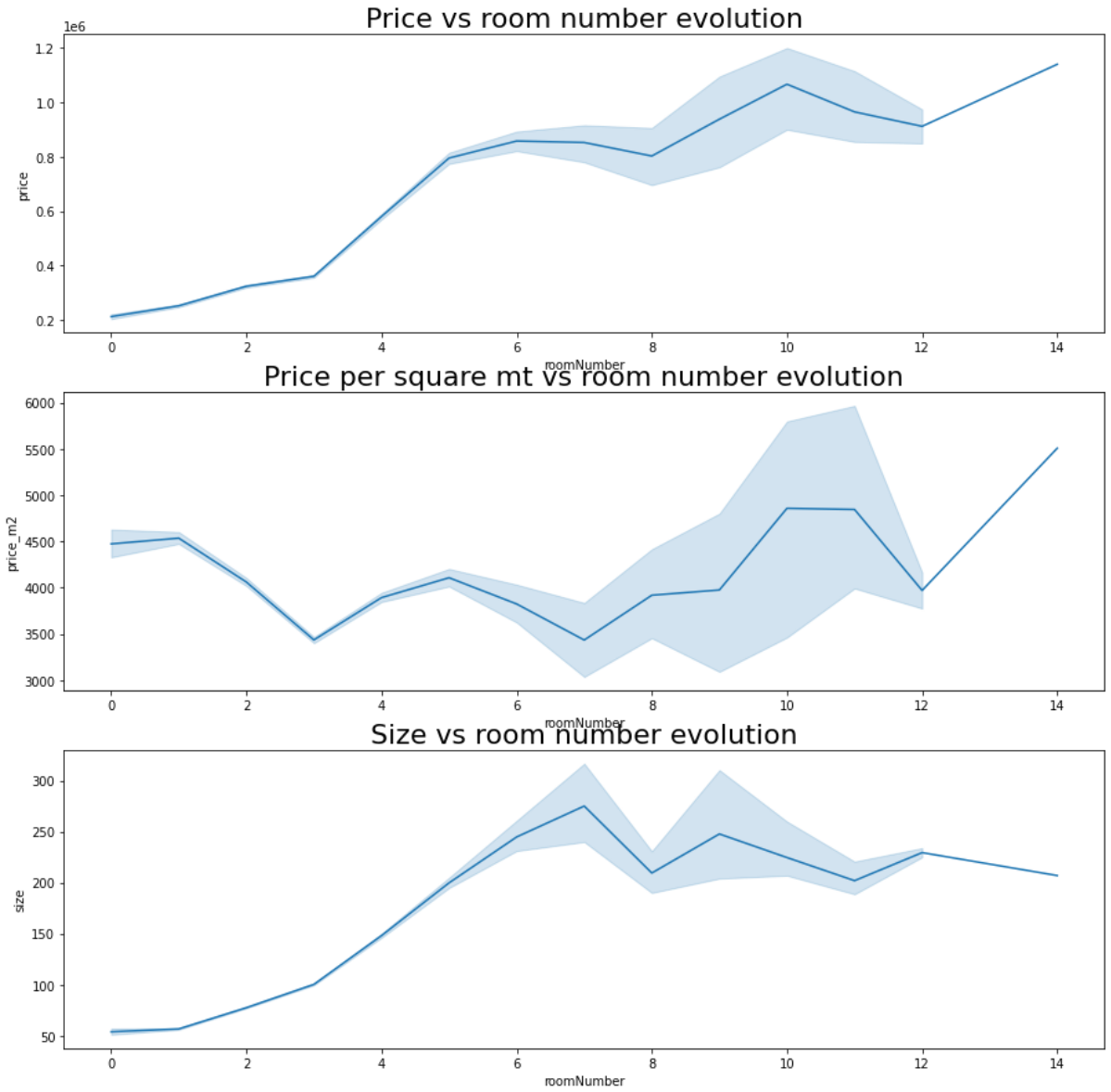
But also, if we look at the distribution of the number of bathrooms:

	bathNumber				price	price_m2
	mean	max	min	count	mean	mean
roomNumber						
0.0	1.110644	3.0	1.0	714	2.125368e+05	4472.752773
1.0	1.052219	4.0	1.0	3313	2.524272e+05	4534.827416
2.0	1.397732	12.0	1.0	6791	3.241794e+05	4060.836914
3.0	1.597982	5.0	1.0	7731	3.608149e+05	3434.791961
4.0	2.232659	5.0	1.0	2768	5.804399e+05	3891.950741
5.0	3.026101	6.0	1.0	613	7.954048e+05	4098.439299
6.0	3.315152	6.0	2.0	165	8.656769e+05	3880.391273
7.0	3.600000	8.0	2.0	45	8.603296e+05	3396.752000
8.0	3.791667	8.0	2.0	24	8.407083e+05	4109.873333
9.0	3.916667	6.0	2.0	12	9.139583e+05	4072.775000
10.0	4.166667	7.0	2.0	6	8.966667e+05	3840.823333
11.0	3.750000	5.0	3.0	4	9.650000e+05	4844.942500
12.0	3.000000	3.0	3.0	2	9.119260e+05	3969.715000
14.0	4.000000	4.0	4.0	1	1.140000e+06	5507.250000

We observe that after 5 bathrooms the number of properties begins to drop rapidly. Therefore, we made the decision to put a limit on this feature and eliminate records above that number of bathrooms.

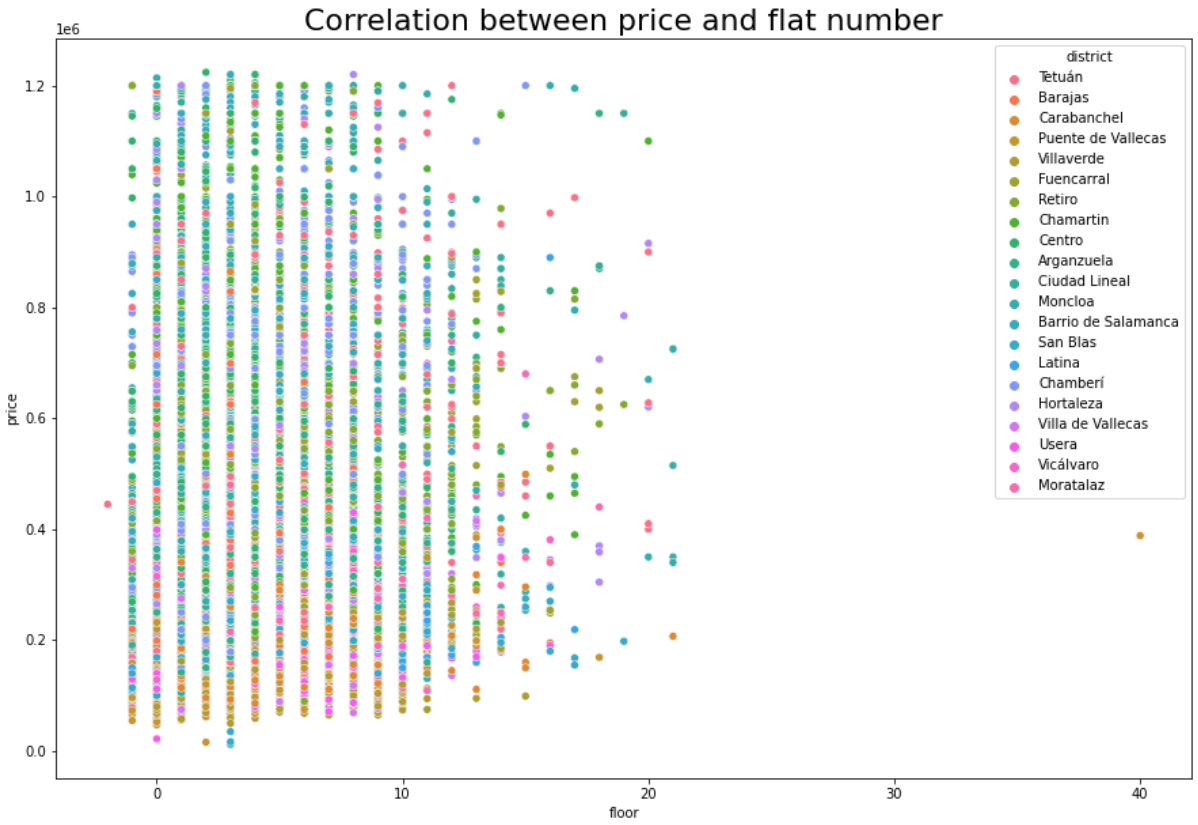


We continued with 'roomNumber', for which we also saw a moderate correlation with price. Again, we found suspicious records with high room numbers,and we asked ourselves what type of house has 10, 12 or 14 rooms?



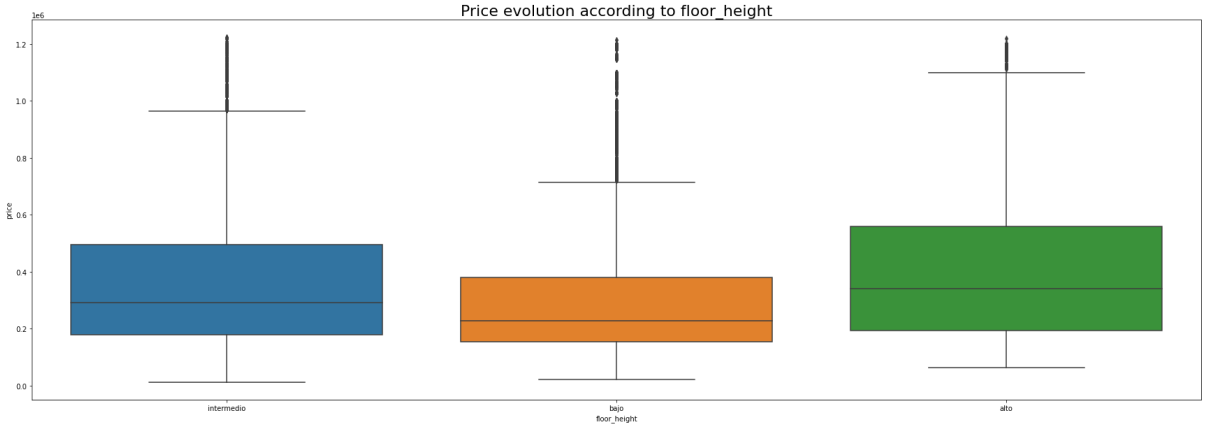
Again the evolution is similar to that of 'bathNumber', finding the 14 rooms as an outlier. In addition, at this point we decided to do a manual investigation that led us to verify in Idealista and Fotocasa that, indeed, most properties with such a high number of rooms were meant for students rental, hostels, entire blocks of flats or offices.

For this reason, we decided to also set a limit to this feature marked by the 99% percentile, which places it at a maximum of 6 rooms per property.



We continue with 'floor'. In this case, the most striking thing is to find a 40th floor, but it does not have to mean anything out of the ordinary. Yes, we can see how our dataset begins to show fewer records from the tenth height, while at the lower end, we only find -2 and some basements.

If we classify the heights into three groups (“bajo” <= 0, “intermedio” > 0 & < 5, “alto” >= 5):

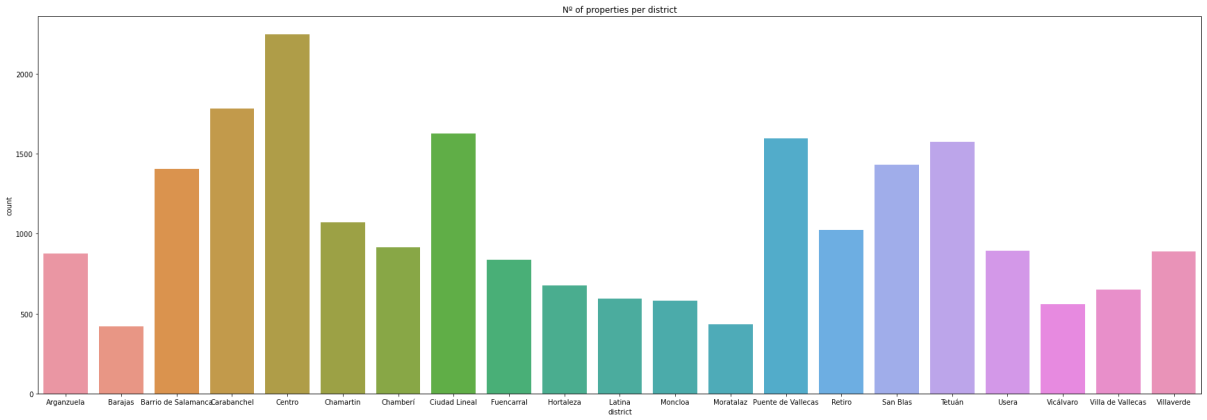


We observe how the largest number of records are concentrated at heights greater than or equal to a fifth floor. It is striking to find a large number of outliers in the "low" category at high prices, possibly motivated by the chalets.

Regarding the rest of the numerical categories, all binary, we were expecting them to have a positive correlation with the price, except isNeedsRenovating. However, there were strange behaviors, such as a negative correlation between price and isGoodCondition and a positive correlation with isNeedsRenovating, something that does not seem normal.

Plus, since only 9% of the records we have showed information about the type of heating, this feature loses importance.

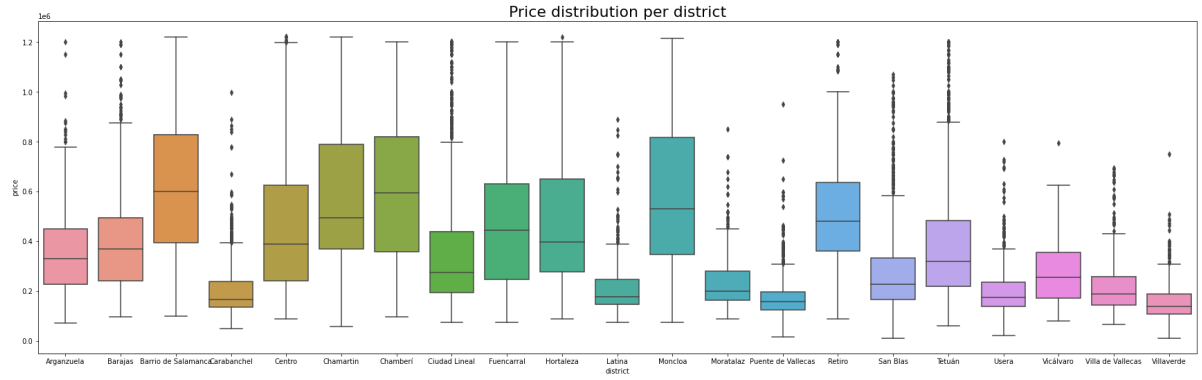
We now continue with the categorical features, starting with 'district'.

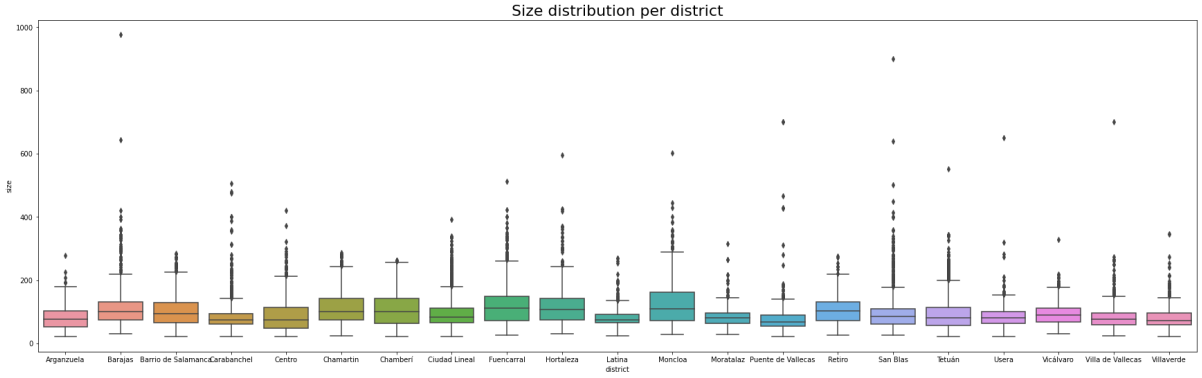


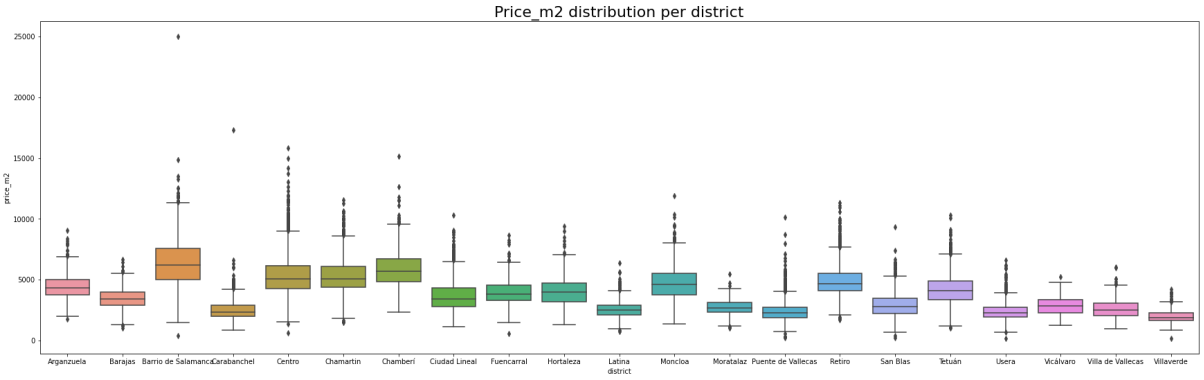
First of all, we observe that the district with the greatest number of records in our dataset is Centro, followed by Carabanchel, Ciudad Lineal and Puente de Vallecas, which gives us an idea that, apart from the first, where the most properties are on sale is in middle-income districts.

If we now take a look at the price distribution by district (image below), we see that the highest prices are found in the Barrio de Salamanca, followed by Moncloa, Chamartín and Chamberí. While the lowest are found in Puente de Vallecas, Villaverde, Usera, Latina and Carabanchel.

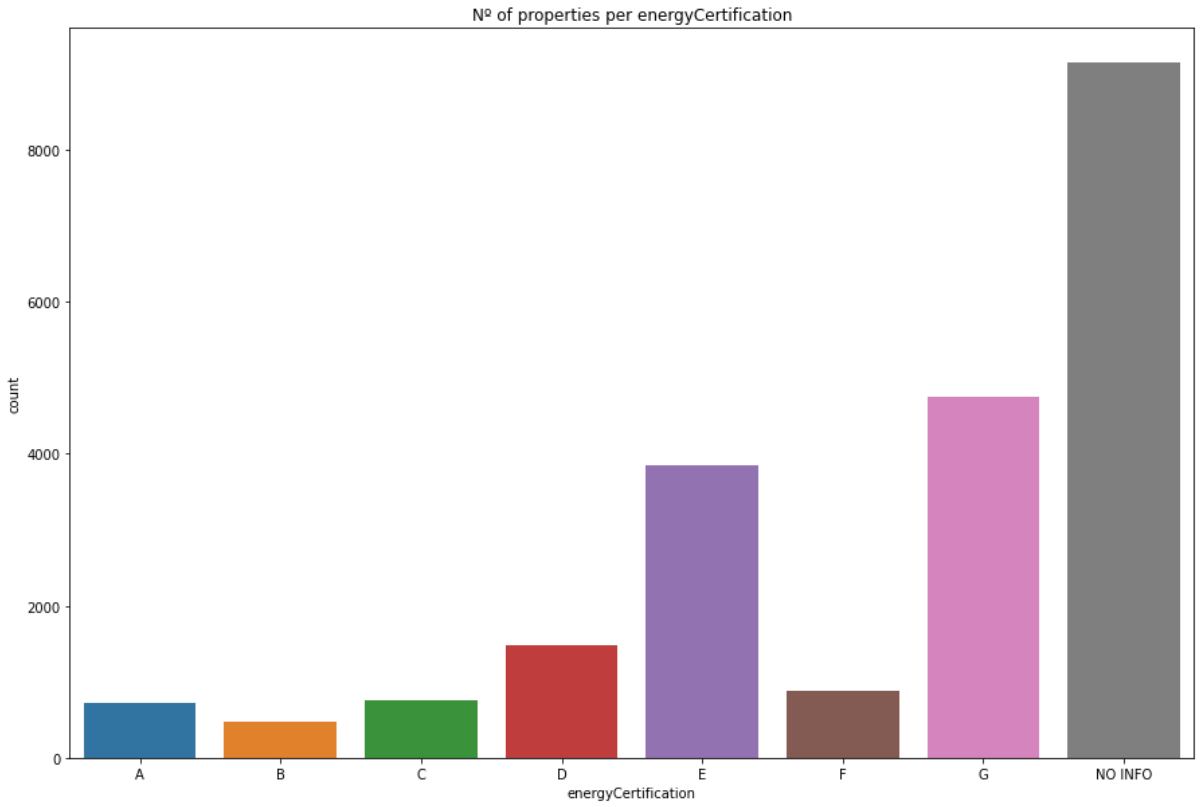
If we look at the size distribution, we see a more equitable scenario, however, Moncloa stands out as the district with the largest housing size without counting the outliers. Finally, going back to the price, in this case per square meter, Barrio de Salamanca is once again the winner.



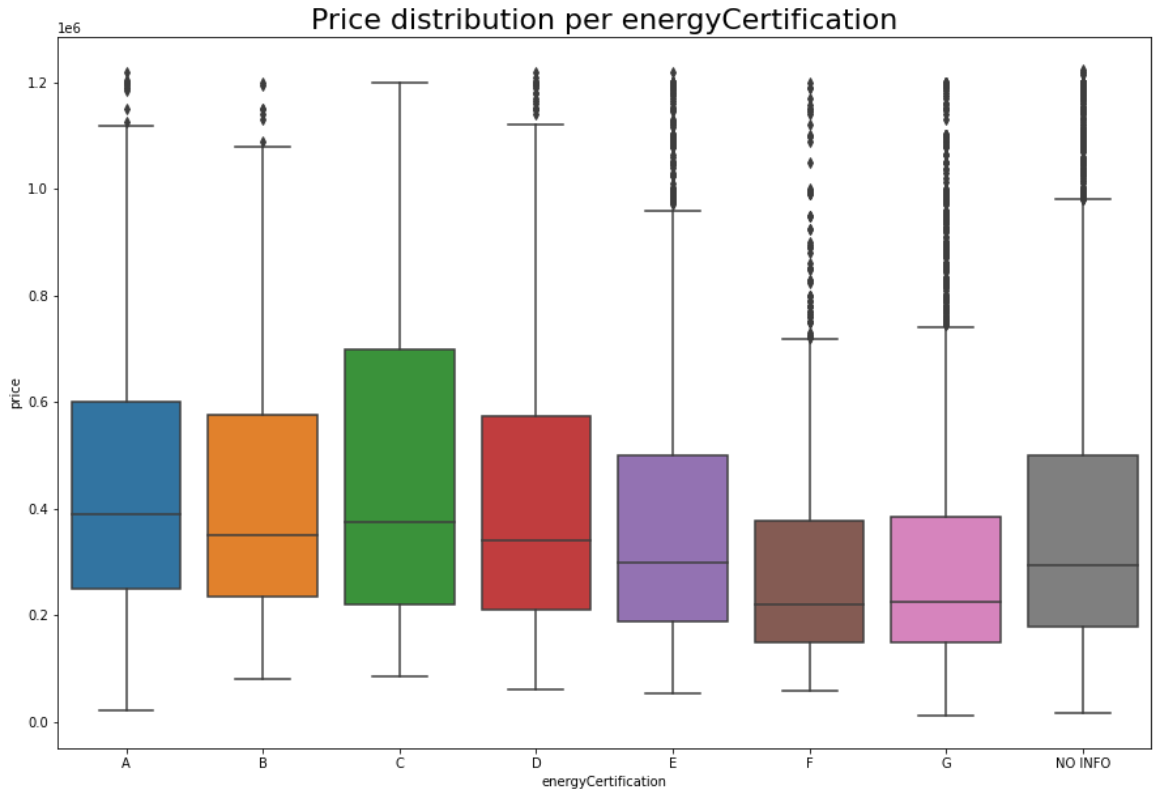




If we look at the 'energyCertification' feature, we see that there is a large number of records (>40%) in the dataset for which we have no information.



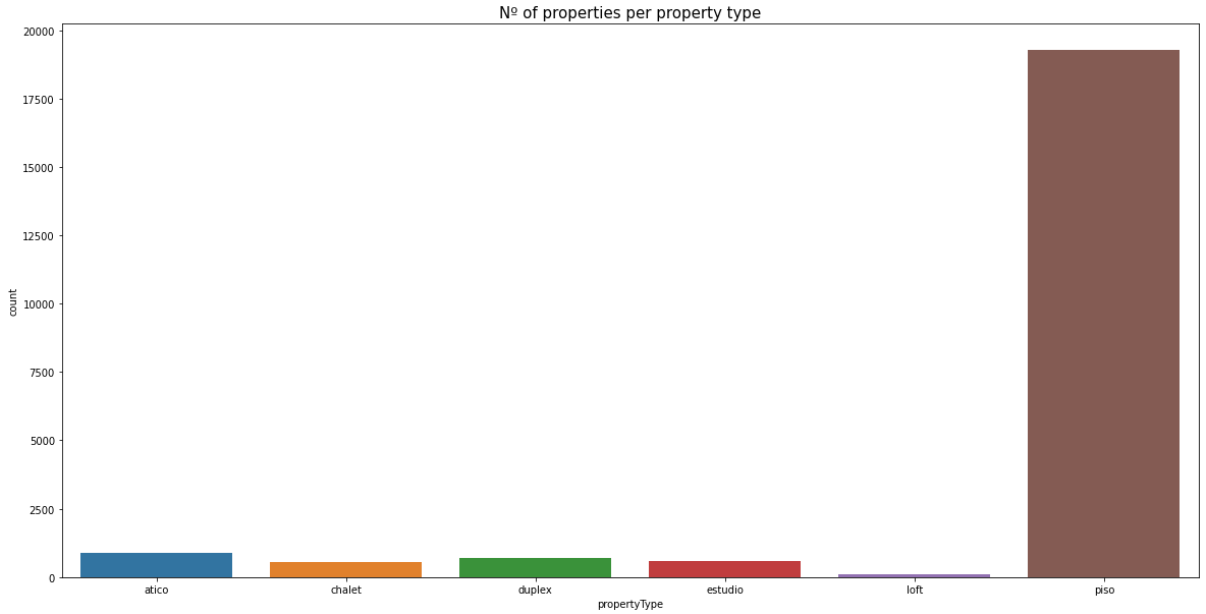
Even so, if we face the feature with the price, although we find a greater distribution in type C and A, the fact that 40% of the dataset has no information means that this feature cannot be used.



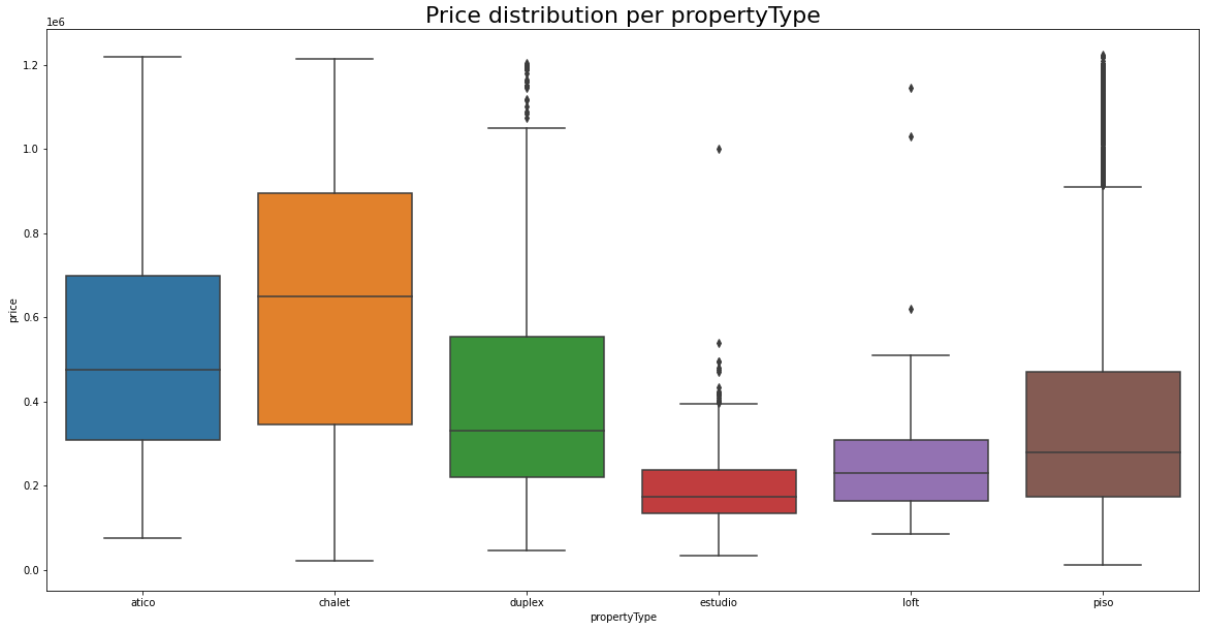
The same happens with the 'heatingType' feature, where we find that >90% of our dataset does not have heating or does not have information about it:

	heatingType	heatingType
no info/no calefacción	20071	90.91%
gas natural	1567	7.1%
electricidad	370	1.68%
gasoil	63	0.29%
solar	4	0.02%
gas propano/butano	3	0.01%

Finally, the one we do have information about, and a quite important one, is the feature 'propertyType'. First we check that most of the dataset properties are flats:



However, the highest prices are found in chalets, penthouses and duplexes. Something that seems totally normal.

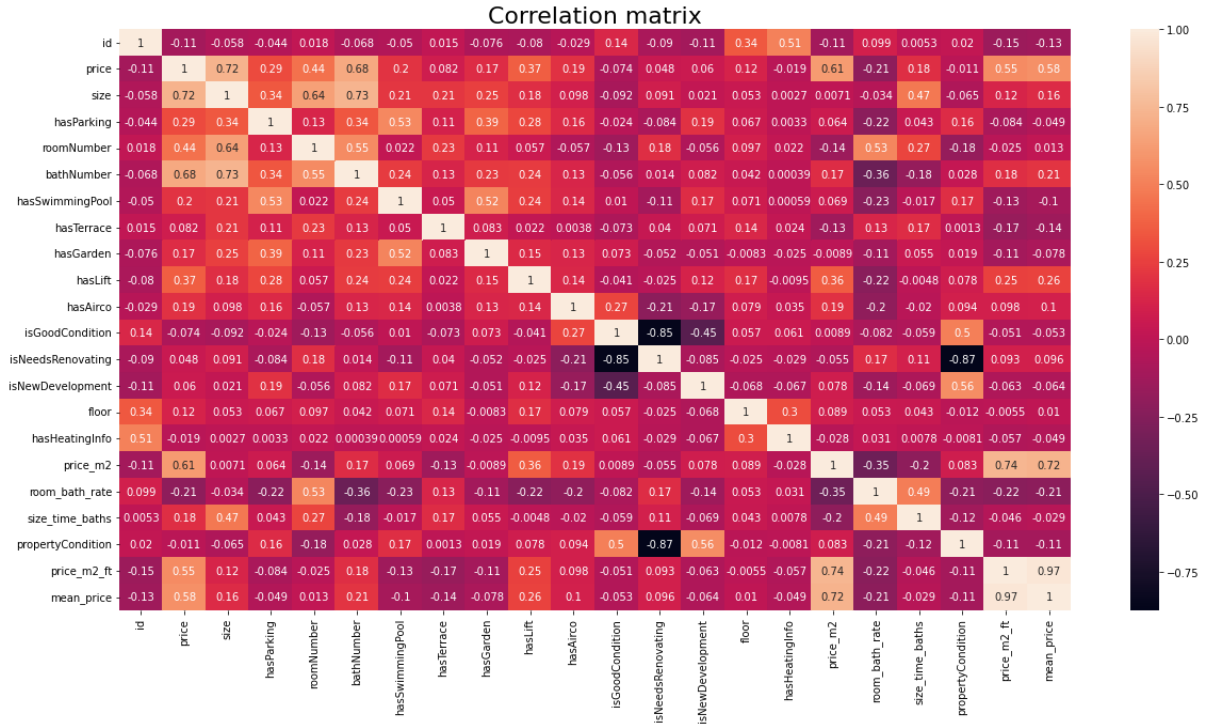


Obviously, we also find a large number of outliers in apartments, since it is the most common type of property and they will respond to many different variables that can make them more expensive, such as the district or the size.

Feature engineering

Although in previous steps we had already transformed and added features that can be considered as feature engineering, there are still the final transformations that would allow us to make the data fit to the models.

At this point, although we tried to obtain new features from external sources by extracting average prices from districts through scraping from Fotocasa.es, the most notable changes are the codifications of the categorical features. But before that, we prepare the final dataset that we are going to submit to the models:

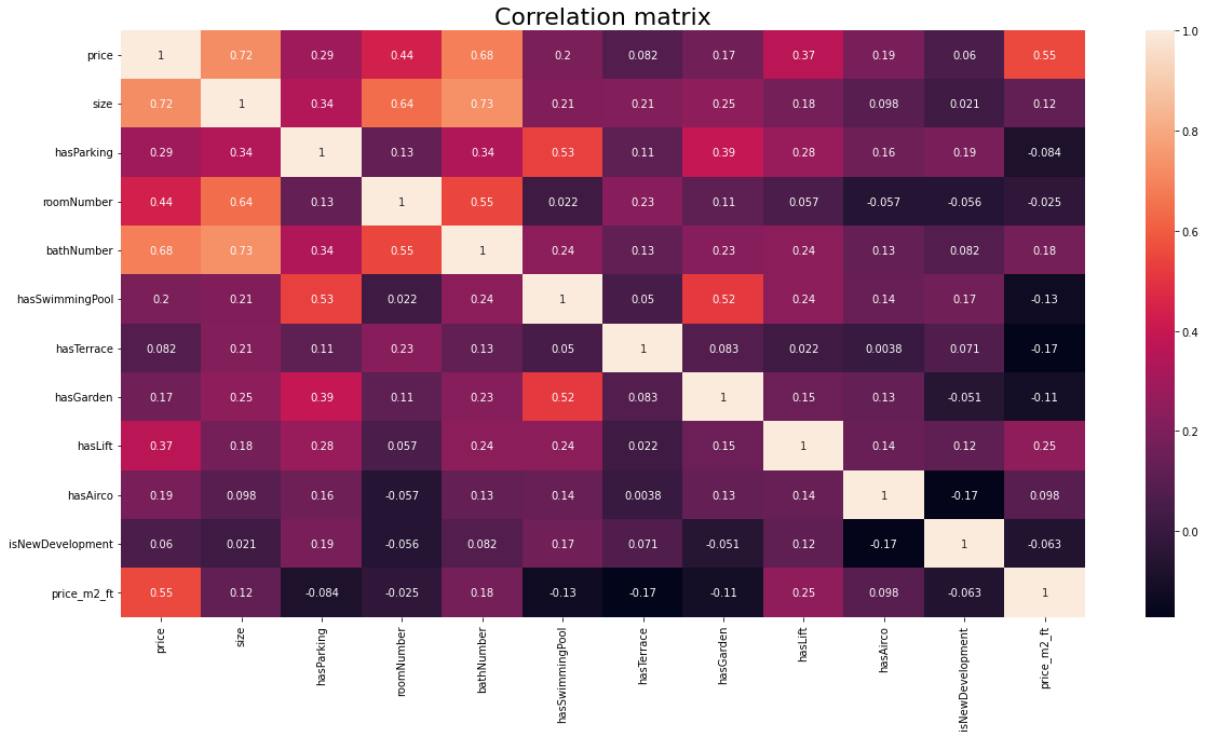


In this new Pearson correlation matrix, we can see that some of the latest features we have added show high collinearity, specifically the last two we have introduced: 'price_m2_ft' and 'mean_price'. So we decided to remove the second one.

Also, at this point after the analysis, we decided to remove the following features due to their little relevance to the dataset or for their strictly analytical purposes:

```
features_to_drop =['id','title','isGoodCondition', 'isNeedsRenovating', 'energyCertification',
                   'heatingType', 'floor', 'dataset', 'hasHeatingInfo', 'price_m2',
                   'room_bath_rate', 'size_time_baths', 'floor_height', 'propertyCondition','mean_price']
```

Once done, we obtain the following matrix, which is getting closer to the final one:



Next, we decided to apply three different categorical variable transformations that we might use later to see their performance with the evaluation metrics of the regression models. In this case, we chose Target Encoding, One Hot Encoding, and a mix of the above two with feature transformations.

- **Target Encoding** [7]: It is a method of coding categorical features that does not increase the number of these, but instead converts them to numeric using the mean of the dependent variable over the different groups of independent variables existing in the dataset. In this process we also finally removed the 'price_m2_ft' feature given its high correlation with 'district'.

- **One Hot Encoding**: This coding method increases the number of features by creating a new binary one for each different value of the categorical feature. The result is a matrix of ones and zeros, being one when the record has that value and zero when it does not.

- **Mixed Encoding**: This type of coding is also a One Hot Encoding, but first the 'propertyType' and 'hasLift' features are unified in order to test if in this way we obtain a feature that may be more relevant to the model and also to make the hasLift feature not to affect chalets which is a propertyType that physiologically does not have elevators. Finally the result is also subjected to One Hot Encoding.

Modeling

Once we have the clean dataset and with the corresponding transformations, it is the turn of the models. Before deciding which is the definitive one for the project, we made a comparison of several of them without any hyperparameter, in order to know their general performance.

Given that the main reason for the project is obtained through a multiple linear regression, given that we need to obtain the value of a dependent variable (Y), the price, with respect to a set of independent variables (X), the characteristics of the properties, our options are considerably reduced.

For this reason, we chose a small sample of the different types of models that allowed us to carry out the regression, from the simplest to the most complex, focusing on the boosting type, which currently offers the best results.

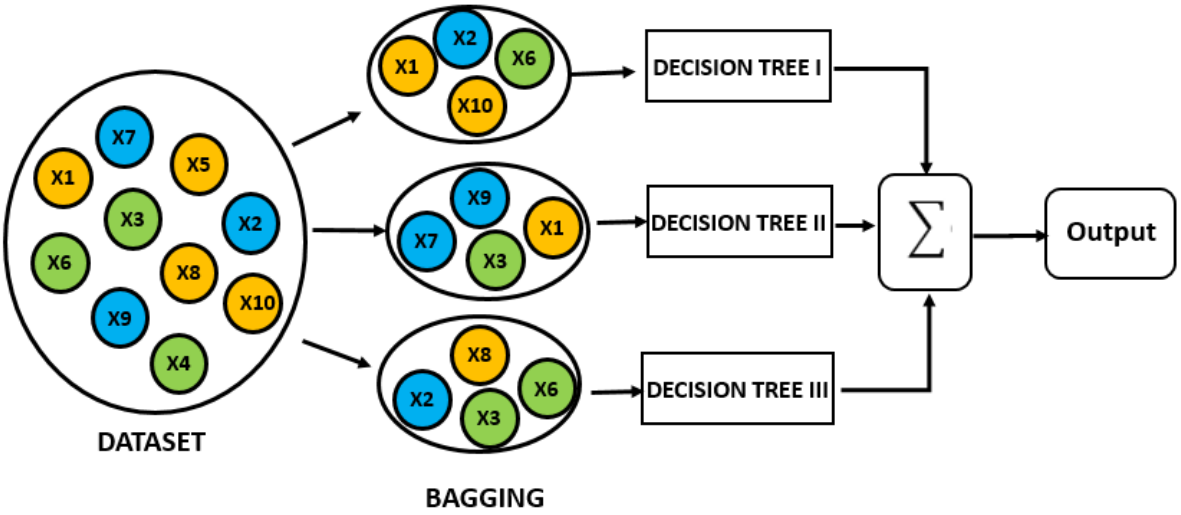
The boosting models are composed of assembled algorithms that use other simpler algorithms sequentially, until obtaining a calculated prediction that, in the case of regression, is obtained by weighing the results of each one of the algorithms used.

Thus, the models chosen in the project have been the following:

- **Linear Regression**: It is the most basic model for the development of linear regression. It is found in the Scikit-Learn library, within its section on classical linear regressions.

- **Decision Tree Regression**: this model performs regression using a decision tree. The regression trees result in a quantitative variable that is obtained through the distribution of the observations by different bifurcations until reaching a terminal node. This prediction is obtained from the mean of the response variable of the observations that are in the same terminal node. Furthermore, it does not require distributional assumptions.

- **Random Forest Regression**: it is an ensemble model that applies different decision trees in order to reduce bias. The calculation of the prediction is made with the average of the outputs of all the trees, since it is a bagging algorithm in which the models work in parallel.



- **AdaBoost Regression**[3]: It is one of the first boosting-type models (sequential execution of weaker models) and it works by assigning weights to the predictions made according to their error, so that the simple model executed tries to adjust what the previous one could not.

- **Gradient Boosting Regression**[4]: Also of the boosting type, this model builds sequential trees based on the error of the previous one, scales them according to the learning rate and successively searches for the smallest error between the prediction and the dependent variable until the maximum number of trees is reached or they do not improve the adjustment.

- **XGBoost Regression**[5]: XGBoost is a type of gradient boosting algorithm that has been optimized for speed and performance and has become one of the most popular machine learning algorithms in recent years. It is especially useful when having a large number of features and non-linear relationships between features and target variable. It's performance is achieved thanks to the introduction of accurate approximation algorithms (improvisation) on Gradient Boosting framework.

- **CatBoost Regression**[6]: another model of the boosting type, this one also involves key features such as bagging, boosting and stacking, but on top of this it adds symmetric trees, ordered boosting (to reduce overfitting) and native categorical feature support.

- **LightGBM Regression**: still part of the gradient boosting framework family. It features a fast training speed (even faster than XGboost) and efficiency, lower memory usage, improves the accuracy of the gradient boosting and supports distributed processing, ideal to handle large-scale data. Like catboost, it can handle categorical features natively. It's speed is due to the fact that it's

histogram-based, which means that it aggregates continuous values into discrete bins to increase speed and reduce memory usage.

Models' evaluation

Once the models to test had been chosen, we carried out a small function that would allow us to evaluate each one of them on the datasets encoded with Target Encoding, and both versions of the One Hot Encoding, in order to decide which one to continue with and which models have better performance.

One Hot Encoding

```
TEST - MODEL BENCHMARKING
```

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.8707	8.369643e+09	91485.7534	59980.8480	test
1	LightBM Regression	0.8668	8.623379e+09	92862.1523	61213.5767	test
2	Random Forest Regression	0.8599	9.068787e+09	95230.1768	61077.8567	test
3	XGBoost Regression	0.8522	9.566165e+09	97806.7761	64936.7896	test
4	Gradient Boosting Regression	0.8521	9.572575e+09	97839.5388	64979.2453	test
5	AdaBoost Regression	0.8475	9.868961e+09	99342.6427	62068.0568	test
6	Linear Regression	0.7990	1.301214e+10	114070.7645	79826.6809	test
7	Decision Tree Regression	0.7712	1.480877e+10	121691.2930	75020.7974	test

Target Encoding

```
TEST - MODEL BENCHMARKING
```

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.8681	8.539317e+09	92408.4234	60653.9098	test
1	LightBM Regression	0.8657	8.692283e+09	93232.4134	61452.7646	test
2	Random Forest Regression	0.8577	9.207797e+09	95957.2666	61312.1399	test
3	Gradient Boosting Regression	0.8488	9.785742e+09	98922.9094	65901.3380	test
4	XGBoost Regression	0.8487	9.790818e+09	98948.5633	65875.6699	test
5	AdaBoost Regression	0.8438	1.010894e+10	100543.2393	62582.9342	test
6	Linear Regression	0.7766	1.445927e+10	120246.7201	85216.2845	test
7	Decision Tree Regression	0.7693	1.493310e+10	122201.0651	74915.7179	test

One Hot Encoding (hasLift+propertyType)

TEST - MODEL BENCHMARKING

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.8694	8.450017e+09	91923.9748	60313.9286	test
1	LightBM Regression	0.8667	8.625238e+09	92872.1588	61092.6557	test
2	Random Forest Regression	0.8610	8.996177e+09	94848.1766	60929.0619	test
3	XGBoost Regression	0.8520	9.577574e+09	97865.0792	65007.3203	test
4	Gradient Boosting Regression	0.8517	9.599603e+09	97977.5650	65056.4929	test
5	AdaBoost Regression	0.8432	1.015101e+10	100752.2005	62823.8748	test
6	Linear Regression	0.7990	1.300747e+10	114050.3019	79708.8795	test
7	Decision Tree Regression	0.7654	1.518658e+10	123233.8239	75234.2351	test

If we look at the test results of the three datasets, we see how **the best performance is obtained in the CatBoost and LightGBM models of the One Hot Encoding dataset**, so we decided to continue with this one and try to improve the performance of the two models. mentioned.

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.8707	8.369643e+09	91485.7534	59980.8480	test
1	LightBM Regression	0.8668	8.623379e+09	92862.1523	61213.5767	test

Clustering

Before working on tuning the hyperparameters of the CatBoost and LightGBM models, we wanted to incorporate a new feature based on clustering models. To do this, we tested KMeans and HDBSCAN.

- **KMeans** [8]: It is an unsupervised learning algorithm that allows classifying data sets into clusters based on distance. Its operation is based on reducing the sum of the distances between each object and the centroid of its cluster, in such a way that its position is modified based on the average of the data assigned to the centroid until it stops moving.

- **HDBSCAN** [8]: It is an algorithm built on DBSCAN capable of identifying groups of variable density. Unlike distance-based algorithms, HDBSCAN

establishes random points and checks the criteria established as hyperparameters to introduce the data within one cluster or another.

To verify its impact, we did the test on CatBoost, running the model on the dataset without the cluster as a feature and on two more datasets, one with the KMeans cluster and the other with the HDBSCAN cluster:

Without cluster feature

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.9084	6.005724e+09	77496.6049	51549.8699	train
1	CatBoost Regression	0.8707	8.369643e+09	91485.7534	59980.8480	test

With KMeans cluster feature

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.9086	5.994176e+09	77422.0662	51439.0468	train
1	CatBoost Regression	0.8720	8.285730e+09	91025.9829	59756.0979	test

With HDBSCAN cluster feature

	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.9099	5.906365e+09	76852.8809	51126.9386	train
1	CatBoost Regression	0.8704	8.389845e+09	91596.0998	60081.6251	test

As can be seen, the only cluster that improves CatBoost performance is KMeans, so we decided to incorporate this feature into the final dataset.

Hyperparameters

We reached this point with the dataset encoded with One Hot Encoder, to which we have added a new feature based on KMeans and with the CatBoost and LightGBM models to be improved.

In this improvement process we will use GridSearchCV, a cross-validation technique that allows us to check the different combinations of given values for each hyper parameter chosen for said test. Although only one case is mentioned in this report, reaching an improvement in the metrics has involved different attempts, testing

values close to those that were getting better and better results, initially starting from the default values.

CatBoost

The CatBoost hyperparameters [9] on which we have worked have been the following:

- iterations: maximum number of trees to build until the problem is solved.
- learning_rate: is the ratio used to reduce each step of the gradient.
- depth: depth of the tree (default 0.03).
- l2_leaf_reg: L2 regularization coefficient of the cost function (default 3).
- loss_function: set to use RMSE as the training metric to optimize in order to reduce the error.
- random_strength: The multiplier of the variance of a normally distributed random variable added to the score of each split. It helps overcome the overfitting of the model

The parameters set for the test are the following:

```
%%time
CB_model = CatBoostRegressor()
cv = RepeatedKFold(n_splits=3, n_repeats=2, random_state=42)
parameters = {'iterations': [1000,1200],
              'learning_rate': [0.04,0.05],
              'depth': [7,12],
              'l2_leaf_reg': [2,3],
              'loss_function': ['RMSE'],
              'random_strength': [5,8]}

grid_CB = mf.search_cv(X_train, y_train, CB_model, parameters, cv)
```

Here are the best performing results:

```
CB_model = CatBoostRegressor(learning_rate = 0.04,
                             depth = 7,
                             loss_function = 'RMSE',
                             iterations = 1200,
                             l2_leaf_reg = 2,
                             verbose = False,
                             random_strength = 5)
```

If we compare the results of the default model with those of these hyper parameters:

Default params:

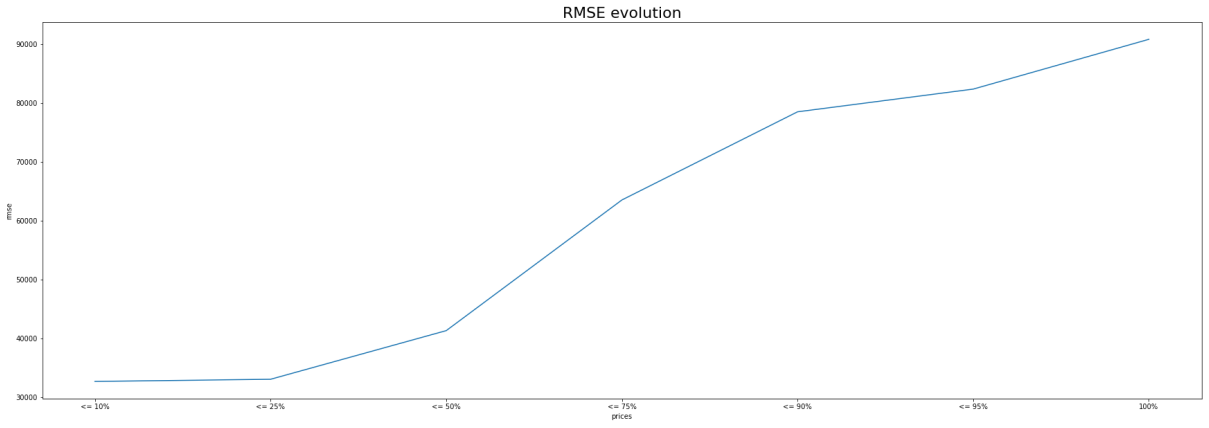
	model	r2	mse	rmse	mae	type
0	CatBoost Regression	0.9086	5.994176e+09	77422.0662	51439.0468	train
1	CatBoost Regression	0.8720	8.285730e+09	91025.9829	59756.0979	test

Tuned params:

	model	r2	mse	rmse	mae	type
0	CatBoost Regression (all data)	0.9121	5.767241e+09	75942.3560	50541.3586	train
1	CatBoost Regression (all data)	0.8723	8.265689e+09	90915.8369	59676.1767	test

So we have managed to increase the R2 slightly in the test, from 0.8720 to 0.8723 and reduce the RMSE from €91,025 to €90,915. The difference is very small, so it would be necessary to continue investigating new combinations.

Since it is difficult to reduce the RMSE in a more relevant way, we wondered how does the price of the property affect the prediction? That is, how the volume of the price is related to the error. To do this, we divided the y_test prices into different percentiles and extracted the RMSE as we incorporated higher ones, which resulted in the following graph:



With it, we can conclude that the higher the price, the harder it will be for the model to predict it precisely.

LightGBM

Again, we run GridSearchCV, in this case on the LightGBM model.

The LightGBM hyperparameters [10] on which we have worked have been the following:

- max_depth: sets the depth of each tree, a tree with more depth can lead to overfitting. A value between 3 and 12 is recommended.
- learning_rate: Just like for the CatBoost algorithm, it's the ratio to reduce each gradient step.
- n_estimators: It's the number of decision trees, similar to the "iterations" in CatBoost.

The parameters set for the test are the following:

```
%%time
LGBM_model = LGBMRegressor()
cv = RepeatedKFold(n_splits=3, n_repeats=2, random_state=42)
parameters = {'max_depth': [5,10],
              'learning_rate': [0.05,0.03],
              'n_estimators': [300,500,1000],
              'objective': ['regression']}

grid_LGBM = mf.search_cv(X_train, y_train, LGBM_model, parameters, cv)
```

Here are the best performing results:

```
LGBM_model = LGBMRegressor(max_depth = 10,
                           n_estimators = 500,
                           learning_rate = 0.03,
                           objective = 'regression'
                           )
```

If we compare the results of the default model with those of these hyperparameters:

Default parameters:

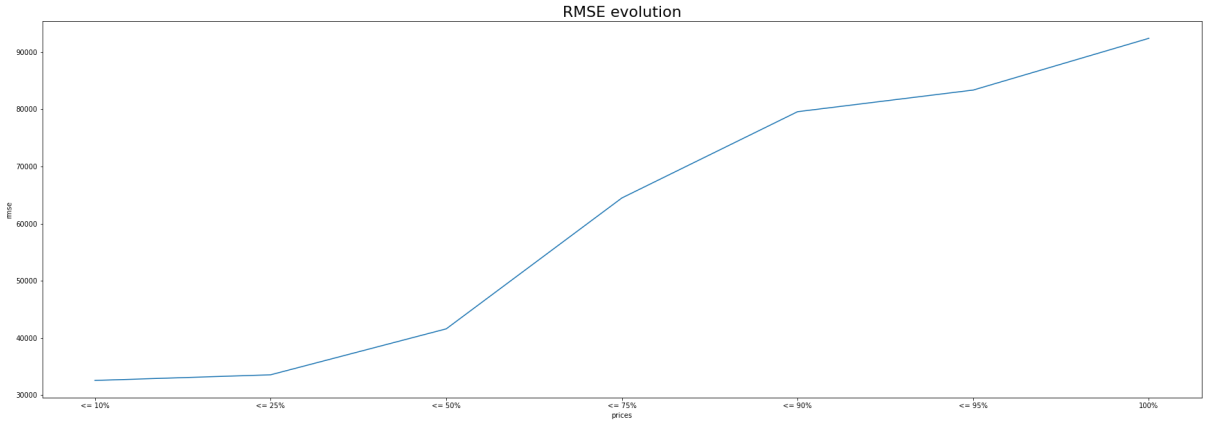
model	r2	mse	rmse	mae	type
LightBM Regression	0.8938	6.964769e+09	83455.1889	55341.9985	train
LightBM Regression	0.8668	8.623379e+09	92862.1523	61213.5767	test

Tuned parameters:

	model	r2	mse	rmse	mae	type
0	LGBM Regression (all data)	0.8992	6.608753e+09	81294.2396	53681.7944	train
1	LGBM Regression (all data)	0.8687	8.497048e+09	92179.4317	60505.1118	test

Por lo que hemos conseguido aumentar el R2 ligeramente en test, de 0.8668 a 0.8687 y reducir el RMSE de 92.862€ a 92.179€.

Just like we did with the CatBoost model, we want to see how prices impact the RMSE as they increase:



De modo que parece algo propio del dataset que, cuando nos enfrentamos a predicciones con un precio más elevado, el modelo tiene mayores problemas para acercarse al resultado correcto.

Content based recommender

To finish the modeling section, although it's not a model per se, we worked on a simple property recommendation system based on cosine similarity.

The goal of this development is to be able to offer a list of existing properties in the dataset (with properties from Idealista and Fotocasa) similar in characteristics to the ones the user inputs in order to predict the price, on the frontend.

Evaluation

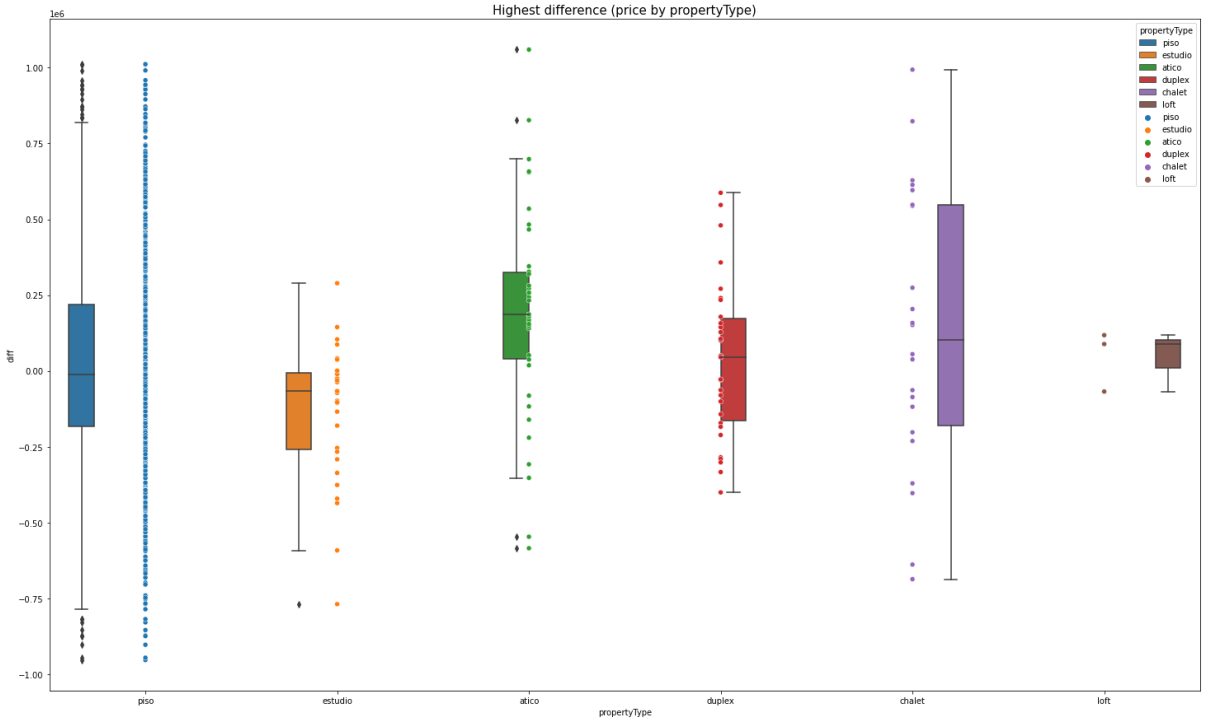
We have reached this point with CatBoost as the clear winner and the model to be used in the project. Although at this point we have already been reviewing evaluation metrics, the ones chosen to analyze the performance of the models were the following:

- R2: It represents the proportion of the variance in the dependent variable that is explained by the model.
- MSE: the mean of the squared difference between the original value and the predicted value.
- RMSE: the square root of the MSE, given in the same unit as the dependent variable and represents the standard deviation of the prediction errors, that is, the dispersion of the data around the best fit line.
- MAE: the mean absolute error is the average of the absolute difference between the actual and predicted values by the model.

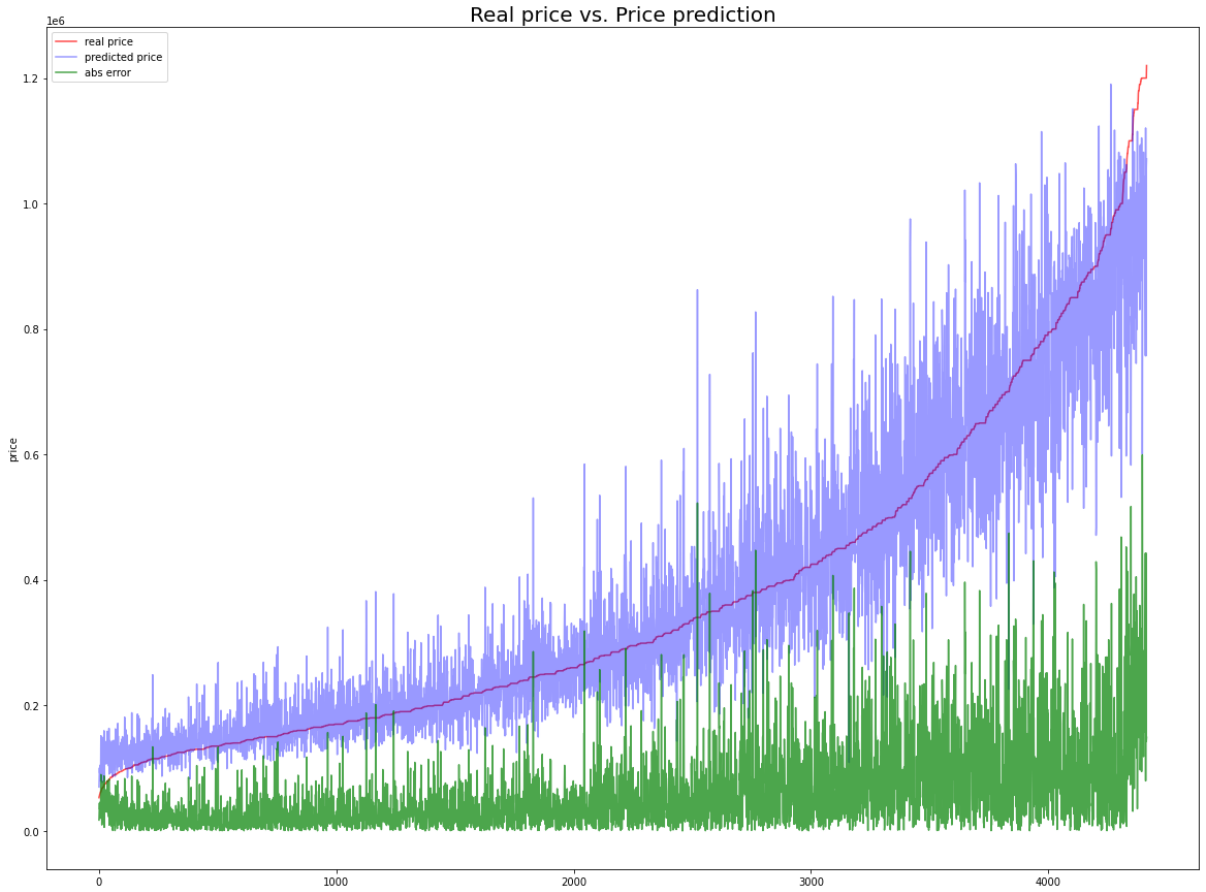
The most important metrics we have been focusing on are the R2 and the RMSE, always aiming for a high value of the first and a low value of the second. As indicated, the best performance achieved in the project was using CatBoost:

	model	r2	mse	rmse	mae	type
0	CatBoost Regression (all data)	0.9121	5.767241e+09	75942.3560	50541.3586	train
1	CatBoost Regression (all data)	0.8723	8.265689e+09	90915.8369	59676.1767	test

To later analyze the model, we made some custom visualizations that allowed us to understand how the predictions behaved according to the dependent variable of the test datasets.



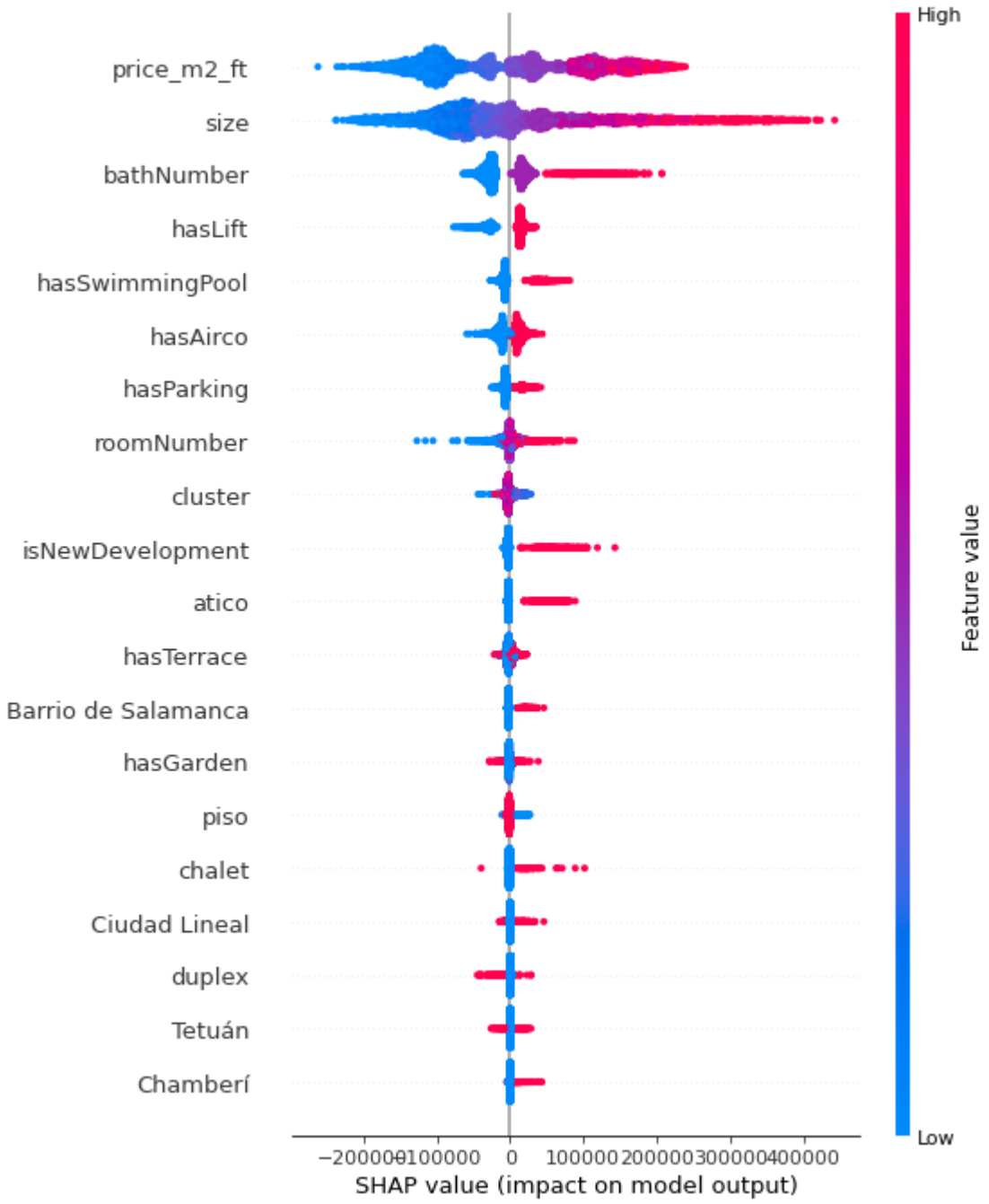
In this graph we can see the distribution of the absolute differences between the actual price and the price predicted by the model for the test datasets. We can observe how we find the greatest difficulty in predicting the prices of chalets and attics, however, the apartments concentrate their errors between 250,000 and -250,000, although much higher absolute errors are reached. Where there seems to be less possibility of error is in the lofts.



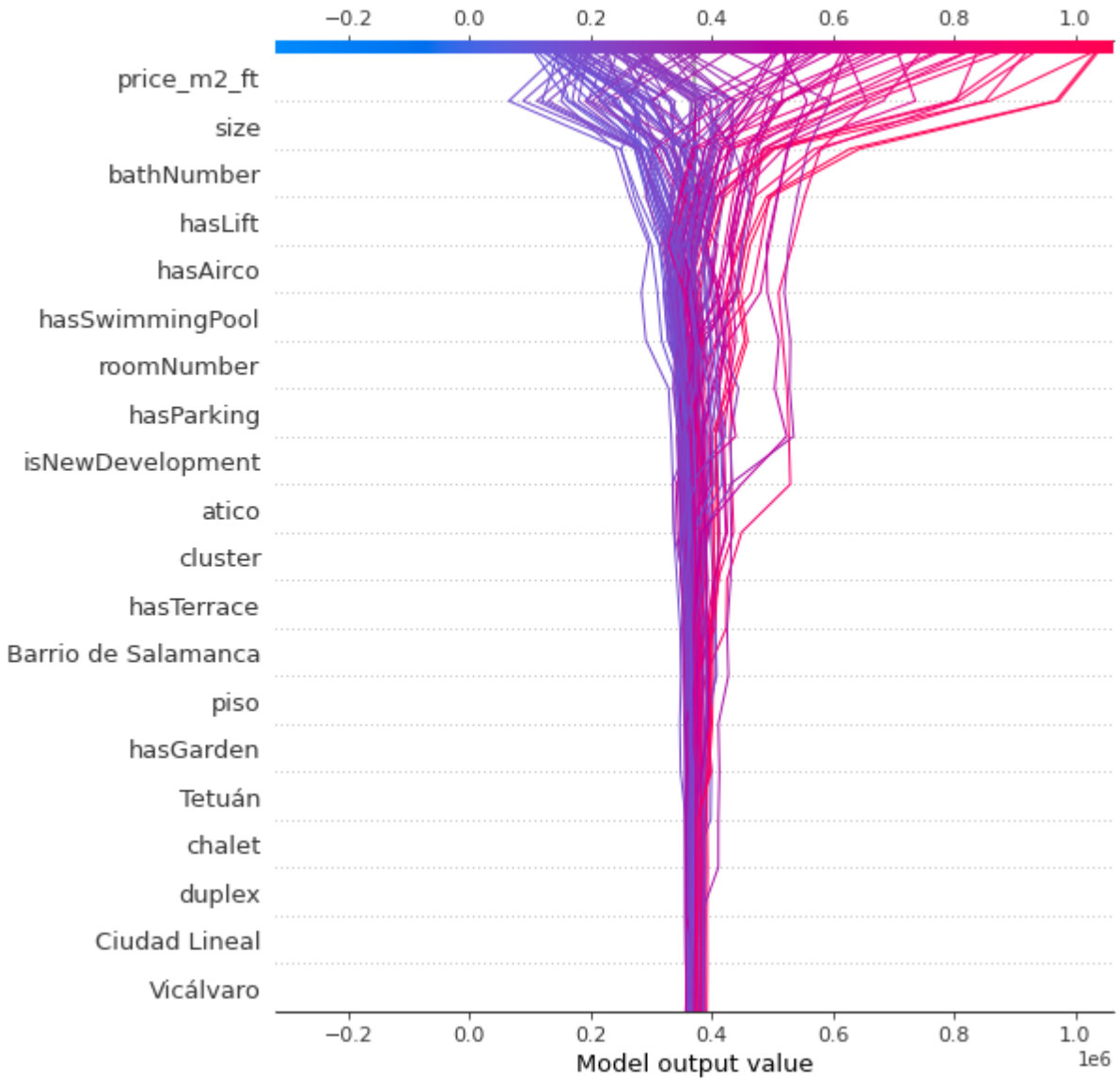
In this graph, we observe the evolution of the actual price and the predicted price against the absolute error, which, as we previously concluded, increases as the model faces higher prices to predict.

Finally, we face the real price with the predicted price again. We clearly see how from 250,000€ the distance between the dots and the red tendency line increases, due to the difficulty of the model to predict the highest prices.

To discover how the features have performed, we have used SHAP, which allows us to understand the impact of these on the predictions.

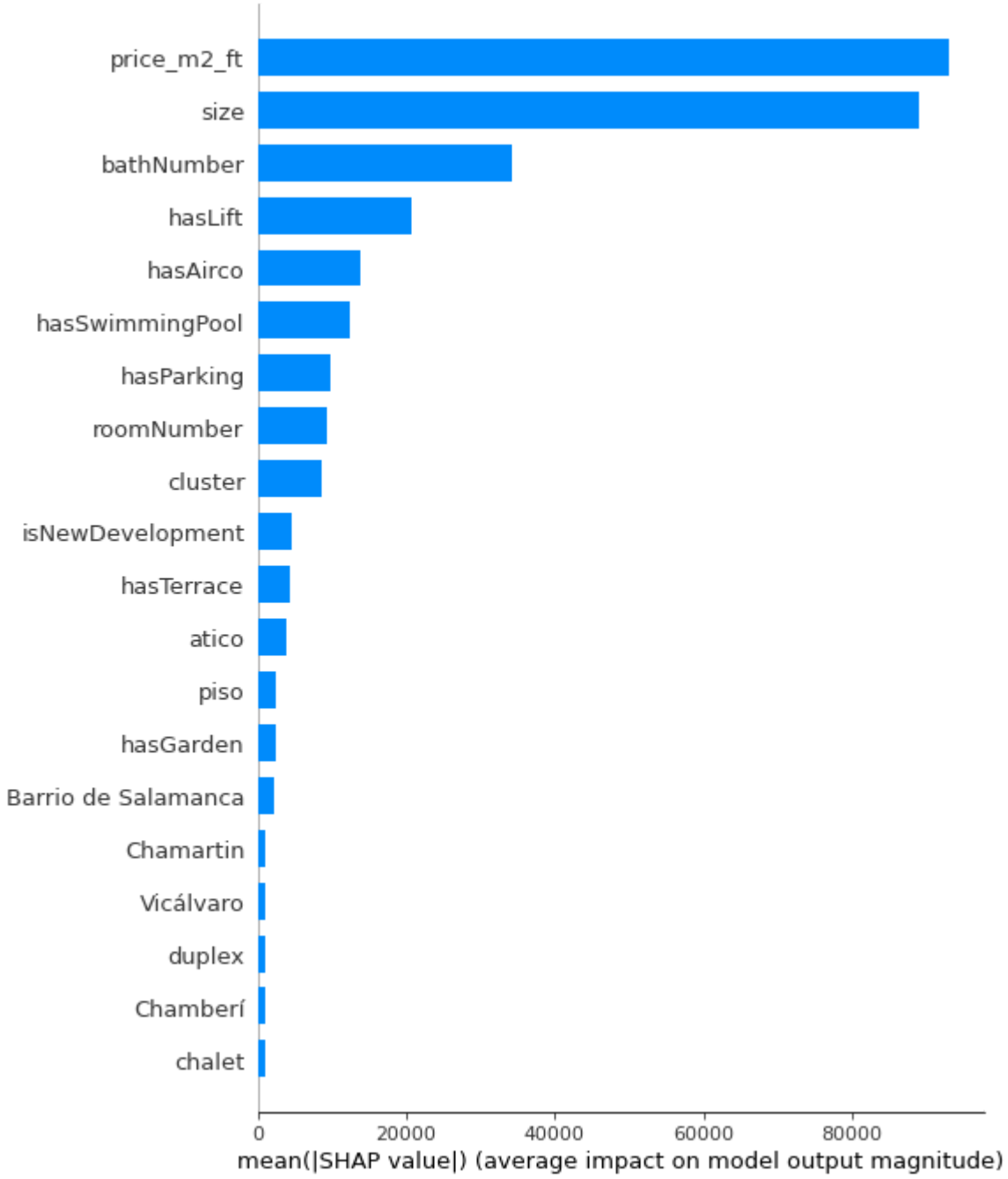


In this Beeswarm graph we can see how the features price_m2_ft, size and bathNumber have a great impact on the prediction. In the first one we can see how, at higher values, the prediction will be greater and vice versa; however, in the case of size, we see that the larger values influence the prediction more than the smaller ones, something that also happens with bathNumber.



In this decision graph we see the first 100 predictions to get an idea of which feature deviates from the expected value. We can confirm that the first three features are the ones that have the most influence.

In this graph we have taken an example of a concrete prediction, where it is seen how each feature "pushes" the prediction to one side or the other of the expected value.



Finally, as we have already indicated, the first three features seem to have a greater impact on the prediction. Out of curiosity, we tried launching the model without “price_m2_ft”, but it worsened the CatBoost performance.

Deployment

The frontend used to make the project work is Streamlit on Heroku. Efforts have been made to implement the simplest interface possible to achieve two goals:

1. Predict prices based on the characteristics introduced by the user.
2. Compare Fotocasa.es URLs to see if the price is above or below the prediction.

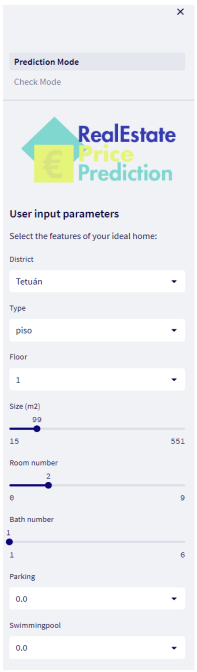
The first mode simply works using the functions developed throughout the project to obtain the dataset (composed of a single record according to the features introduced by the user) with which the model predicts.

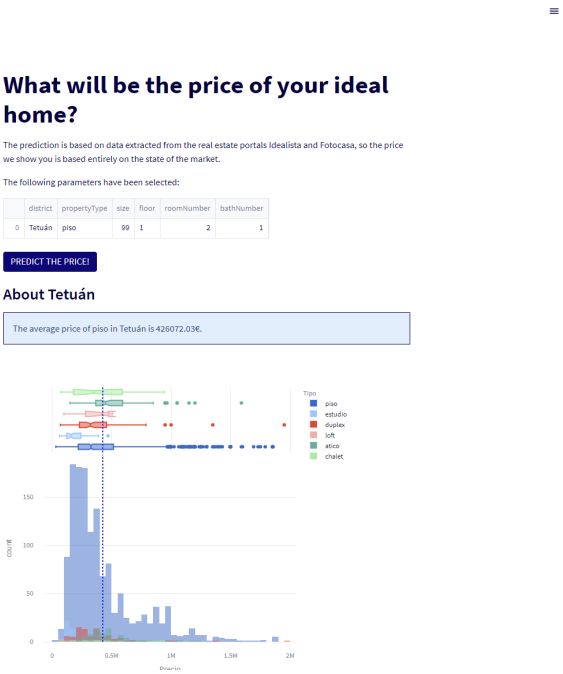
The second mode is slightly more complicated, since we combine the above with the scraping used to extract information from Fotocasa.es to obtain the dataset that the model can predict.

You can access the interface at the following link:
https://realestate-prediction.herokuapp.com/

User manual

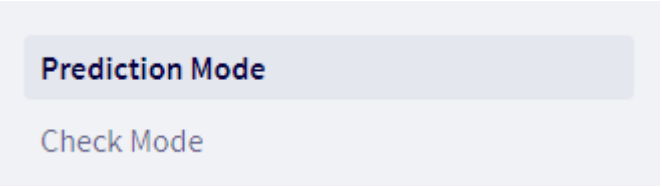
The first thing to keep in mind is that the interface is made of two parts. On the left is the sidebar, where the user can enter the desired values. On the right is the main screen where the results will be displayed. In this location, some average values will be displayed according to the chosen features by default.





How to use it

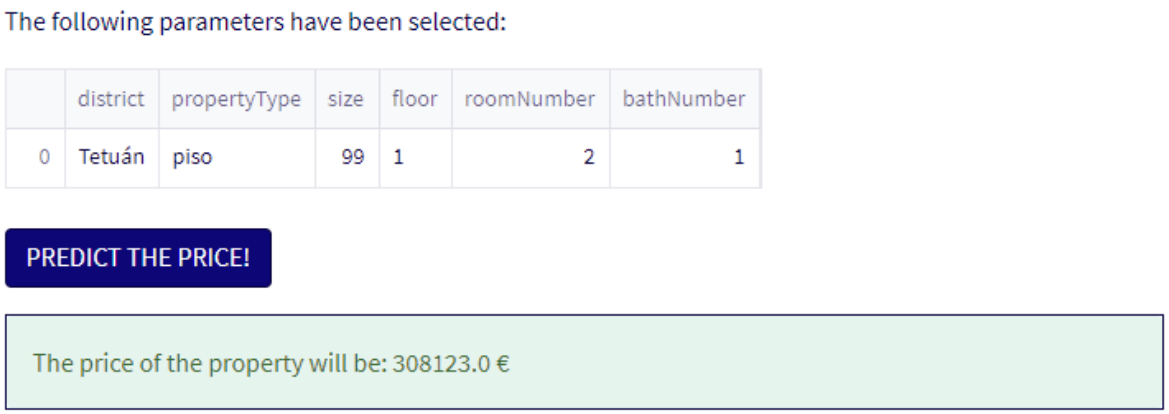
As previously indicated, the interface has two modes of use, below we explain each one. Select each of them by clicking on the desired option in the upper left corner:



1. **Prediction Mode**

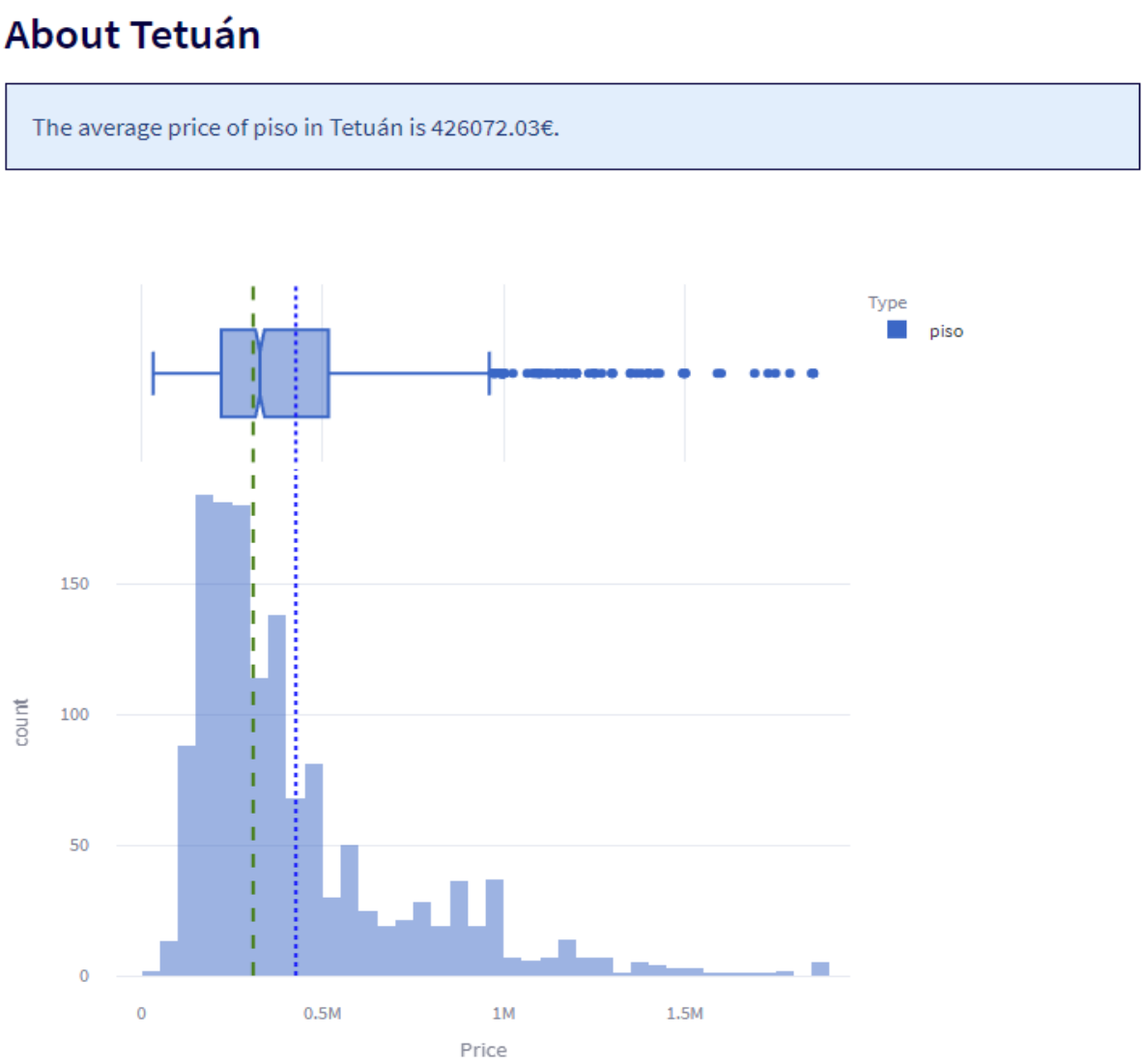
It is the functionality in which the user must fill the options found in the left-side sidebar to enter the features they want the property they are looking for to have. Once these values have been entered, click the blue button **PREDICT PRICE!**.

After clicking the button, the price prediction will be displayed according to the previously established characteristics:



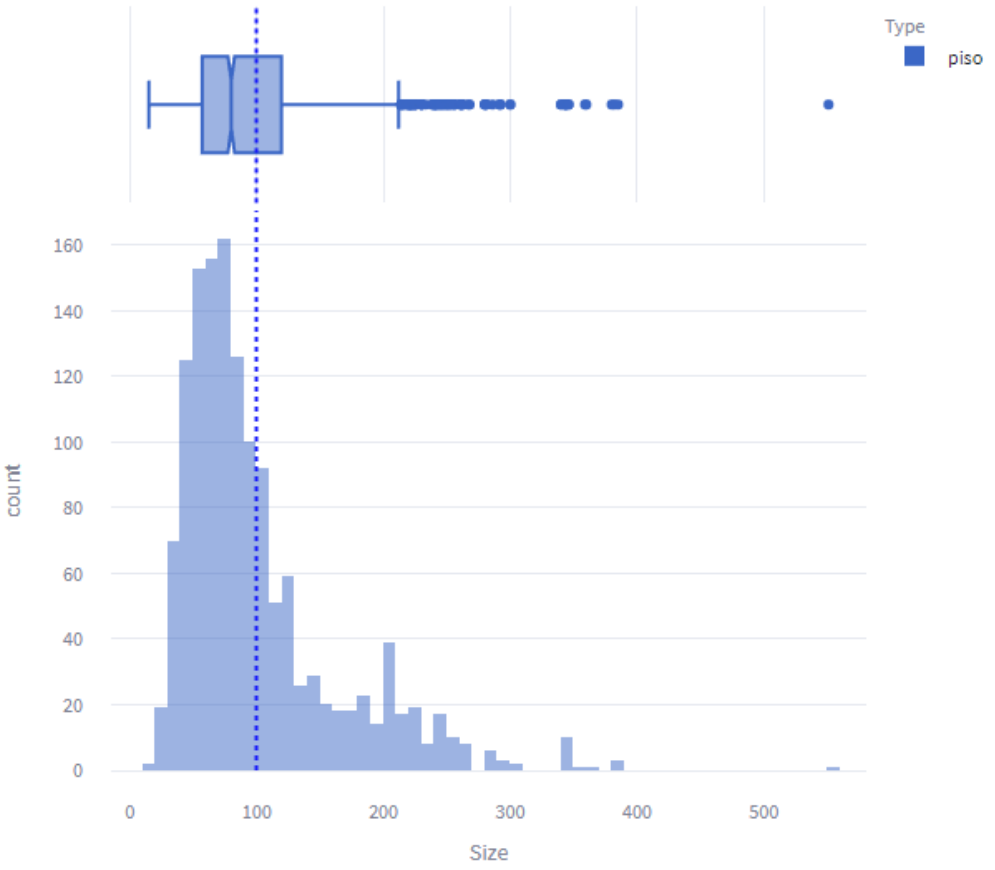
In addition, the rest of the screen will update to show data related to the new price. For example, the price distribution of the selected property type in the selected

neighborhood will be displayed and you will be able to compare the mean (blue line) with the predicted price (green line):



Right below that, you can see the average size of the chosen property type for the selected district:

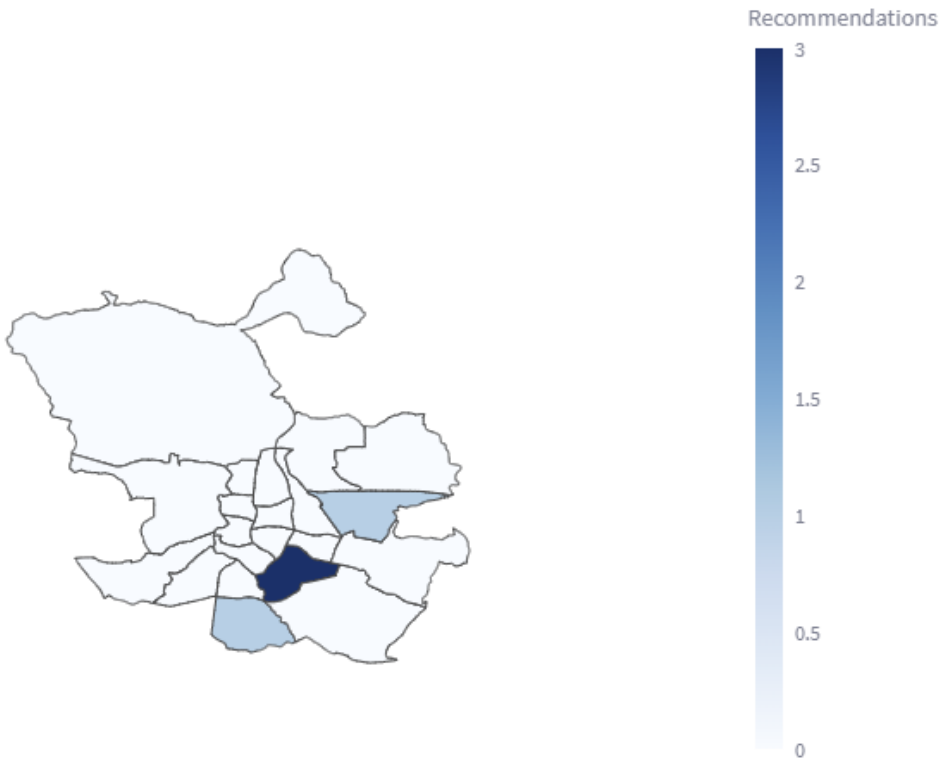
The average size of piso in Tetuán is 99.79m2.



And finally, as a recommender, there will be displayed five properties extracted from the database, having similar values to those established by the user and their location in Madrid:

Similar Real Estates

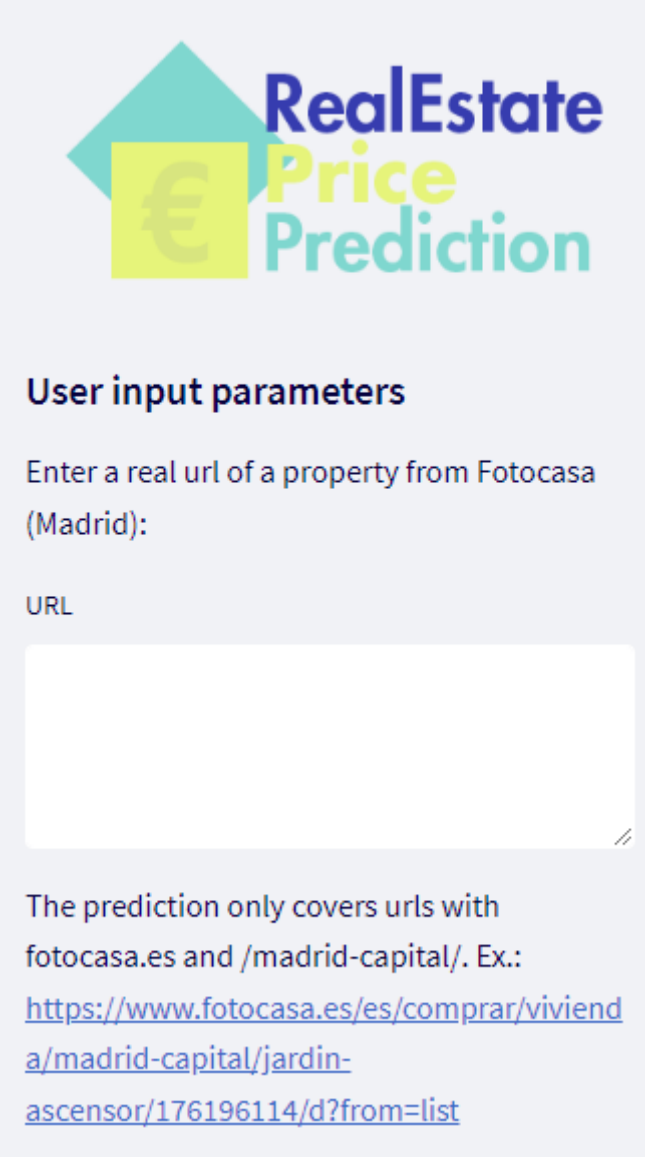
	district	propertyType	size	floor	roomNumber	bathNumber
15214	Puente de Vallecas	piso	701.0000	3	3.0000	1.0000
19343	Puente de Vallecas	piso	116.0000	0	1.0000	1.0000
20839	Puente de Vallecas	piso	167.0000	6	3.0000	1.0000
22310	Villaverde	piso	137.0000	6	4.0000	1.0000
22923	San Blas	piso	214.0000	3	3.0000	1.0000



2. Check Mode

The second mode of use works in a similar way to the previous one, the only difference is that, instead of entering the characteristics of the home that interests

you, you simply have to enter the url. The only condition is that the url is from Fotocasa.es and from Madrid municipality in this case.



When entering the url, the blue “Check url” button will appear, which you will have to click to make the prediction.

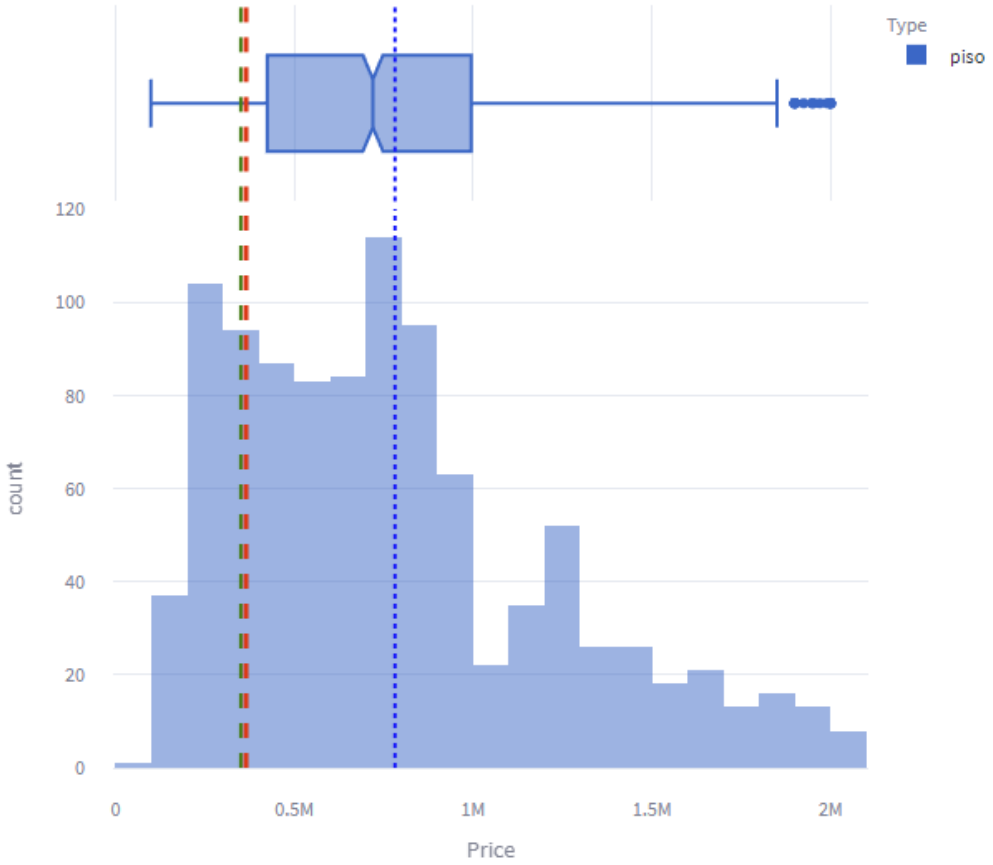
Unlike the previous method, now it will be indicated if the real price is above or below the prediction, and we will be able to see it represented in the graph, where the blue line is still the average, the green is the prediction and the red is the real price:

The price of the property will be: 350398.0 €. The real price is 365000 €.

The difference is 14602.0 €. The price is higher than in the sector.

About Chamberí

The average price of piso in Chamberí is 781843.26€.



Recommendations for similar homes will also be displayed.

Conclusions and next steps

Looking at the final version of the model, it is evident that the RMSE is still quite high for the €30,000 target that we pursued at the beginning of the project. Therefore, the R2 around 0,87 and the RMSE over €90.000 are not the results we expected.

Despite having thoroughly cleaned the dataset, it is inevitable to have an error like this, since the data has been extracted through scraping, which means that it is subject to numerous errors carried from the real estate platforms from which it was extracted, since that it is impossible to ensure that all advertisers correctly fill in the available fields that we have used as features.

Additionally, during the analysis, we uncovered valuable insights that have substantiated previously held beliefs, such as the correlation between certain neighborhoods and the attributes of homes, including pricing. We also discovered new relationships, such as the strong correlation between the number of bathrooms and home prices.

In conclusion, the project has yielded meaningful insights, but the results tell us that there is still room for improvement. To enhance the project's outcomes, it would be interesting to increase the number of records, particularly by broadening the diversity of property types. Additionally, a valuable direction to pursue could be the addition of new features that could bring relevant information to the dataset. Furthermore, exploring different combinations of hyperparameters in the CatBoost and LightGBM models may lead to even more optimal results.

About the correct filling of the properties characteristics by the users, another means of enhancing the efficacy of this project could have been the implementation of Natural Language Processing (NLP) techniques on each property description, to facilitate additional data validation checks. This approach was not pursued due to concerns over scope expansion, as it would have resulted in an overly ambitious undertaking given the limited time available. Additionally, while property descriptions are a user-generated feature, they can still provide valuable information for cross-referencing purposes. However, it must be noted that relying solely on such descriptions may not be sufficient to address all data-related issues, hence there was a risk associated with this approach.

References

[1] https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/madrid-comunidad/madrid-provincia/madrid/
https://www.idealista.com/news/inmobiliario/vivienda/2022/11/17/800194-el-grafico-que-muestra-la-evolucion-del-precio-de-la-vivienda-hasta-2032

[2] https://medium.com/analytics-vidhya/understanding-crisp-dm-and-its-importance-in-data-science-projects-91c8742c9f9b

[3] https://fhernanb.github.io/libro_mod_pred/adaboost.html#explicaci%C3%B3n-sencilla-de-adaboost

[4] https://www.numpyninja.com/post/gradient-boost-for-regression-explained

[5] https://www.kdnuggets.com/2020/12/xgboost-what-when.html

[6] https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm

[7] https://towardsdatascience.com/dealing-with-categorical-variables-by-using-target-encoder-a0f1733a4c69

[8] https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html
https://rubialesalberto.medium.com/clustering-con-dbscan-y-hdbscan-con-python-y-sus-hiperpar%C3%A1metros-en-sklearn-8728283b96ac

[9] https://medium.com/analytics-vidhya/catboost-101-fb2fdc3398f3

[10] https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html