Machine Learning Engineer Nanodegree

Capstone Project

Adrian Ciubotaru
March 31st, 2022

I. Definition

Project Overview

One of the most challenging aspects when it comes to real estate is figuring out the right price of a property. This is true for both the seller and the buyer. One has to either put a lot of effort into researching the market or put their trust in someone to properly represent their interests, both of which come with various costs and risks.

What if there was a better way? What if, by simply pressing a button, one could get a reasonable and unbiased price prediction for their property? In this project I will turn this desire into a reality through the power of Machine Learning.

Problem Statement

The problem is figuring out the right price for properties. The fundamental output of this project will a model that predicts price based apartment characteristics. From the point of view of the end user, the answer will be obtained by simply submitting their characteristics in a form.

The scope is limited to apartments in a few neighborhoods in Bucharest. To achieve this the following steps will be taken:

- Scraping data from the largest real estate portal in Romania (imobiliare.ro)
- Cleaning the data
- Trying a comprehensive set of models & hyper parameters
- Comparing the model against the benchmark
- Deploying the model and exposing it through a Lambda Function & API Gateway
- Creating a simple form with React
- Calling the previous API and returning the result in the front-end

Metrics

This problem is a regression problem, which means that the target variable is a continuous one. Because of this, I will use Root mean squared error (RMSE) as a metric to judge the performance of my model.

I chose this one because it is one of the most common and robust metrics for regression problems.

II. Analysis

Data Exploration

The data used to train the model was scraped in November 2021. It is a list of 2665 dictionaries, each containing the following properties:

- Number of rooms
- Usable surface area
- Total surface area (usable + walls)
- Partitioning Type. This can take the following values:
    - Decomandat – every room is accessible without going through another room
    - Semidecomandat – at least one room is accessible only by passing through the living room
    - Nedecomandat – all rooms rooms are accessible only by passing through the living room
    - Circular
- Comfort type. This can take the following values:
    - 1, 2, 3 – from most to least comfortable
    - Lux – luxury apartments
- Floor
- Number of kitchens
- Number of bathrooms
- Building year
- Building structure. This can take the following values:
    - Beton , Romanian for concrete structure
    - Caramida, Romanian for brick structure
    - Lemn, Romanian for wooden structure
    - Aletele, Romanian for other
- Type of building
    - Apartment building
    - House/villa
- Highest floor
- Listing url
- Number of balconies
- Number of parking spots
- Number of garages
- Specifications. This field is a list of various properties that are not standardized in the same way as the rest. Examples include: available utilities, types of floors, available public transit routes, materials used for doors, furniture etc.
- Price. The price is expressed in three currencies: RON, EUR, USD. It includes VAT where specified.

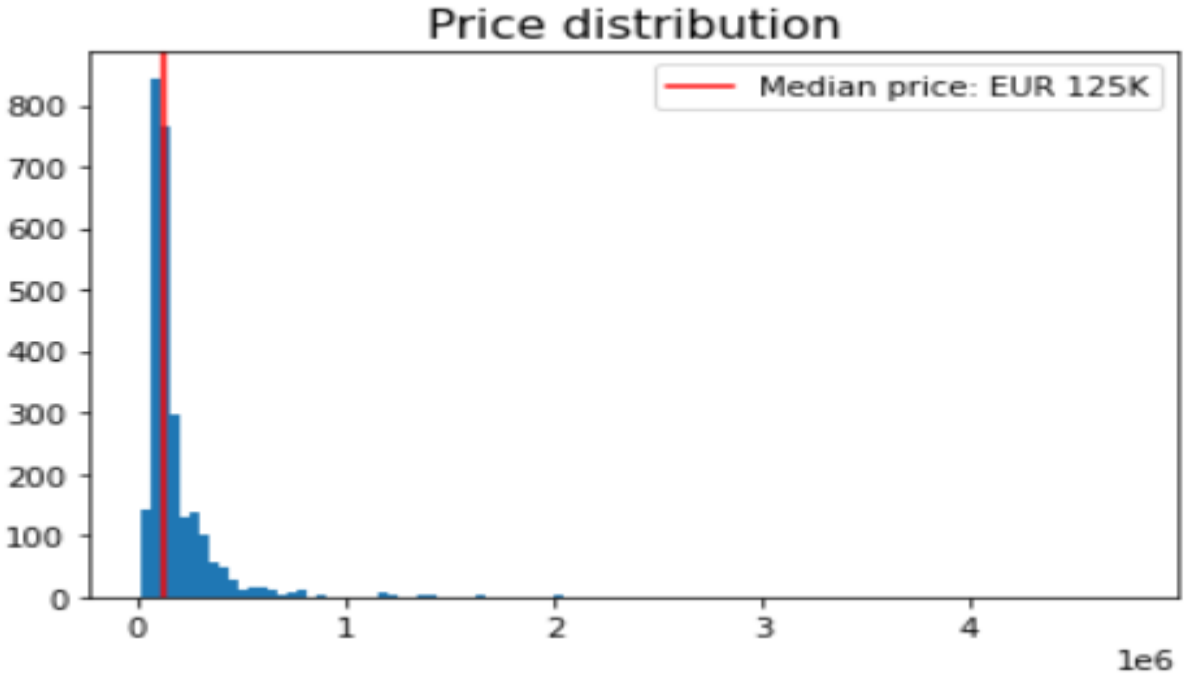
Based on the fields above, the following variables were created to be used in the final model:

- Eur_price. Only the price in EUR was kept as a target variable. Missing values were dropped, as well as values that meant for rent prices. In the end **2659** valid observations remain.
- Partitioning type was turned into partition_decomandat & partition_other. This is due to the fact that decomandat is the most numerous category, with over 60% of the observations. Nulls were assimilated into partition_other.
- Comfort Type was turned into comfort_other, comfort_1 and comfort_lux. Nulls were assimilated with comfort_other, since sellers of either comfort 1 or lux would have a strong incentive to display the proper comfort type
- Highest floor as the column max_floor
- Floor type has the following values: middle_floor, first_floor, last_floor. This distinction is important because the first and the last floors of any building are perceived to be inherently flawed in the Romanian market.
- Bathrooms. All nulls are filled with 1, the most common value
- Bathrooms_ratio. This column is created by dividing bathrooms by rooms.
- Building Year. The following categories were created:
    - before_1941
    - between_1941_1977
    - between_1977_1990
    - between_1990_2000
    - between_2000_2010
    - after_2010
    - not_finished
    - not_started
- Building structure has the following values: unknown_building_structure, concrete_building_structure, other_building_structure. This is because null values are extremely common, while everything else besides concrete is quite rare.
- Number of balconies transformed into a binary variable has_balconies. The previous missing values are counted as no balconies because sellers would have an incentive to specify any number of balconies
- Number of garages and number of parking spots converted into one binary variable, has_parking_spots_or_garages
- Neighborhood. Based on the url, a neighborhood can be assigned to each apartment. Dummies were created for each neighborhood.
    - 1_mai_area
    - Agronomie_area
    - Aviatiei_area
    - Aviatorilor_area
    - Banu_manta_area
    - Chibrit_area
    - Domenii_area
    - Dristor_area
    - Stefan_cel_mare_area

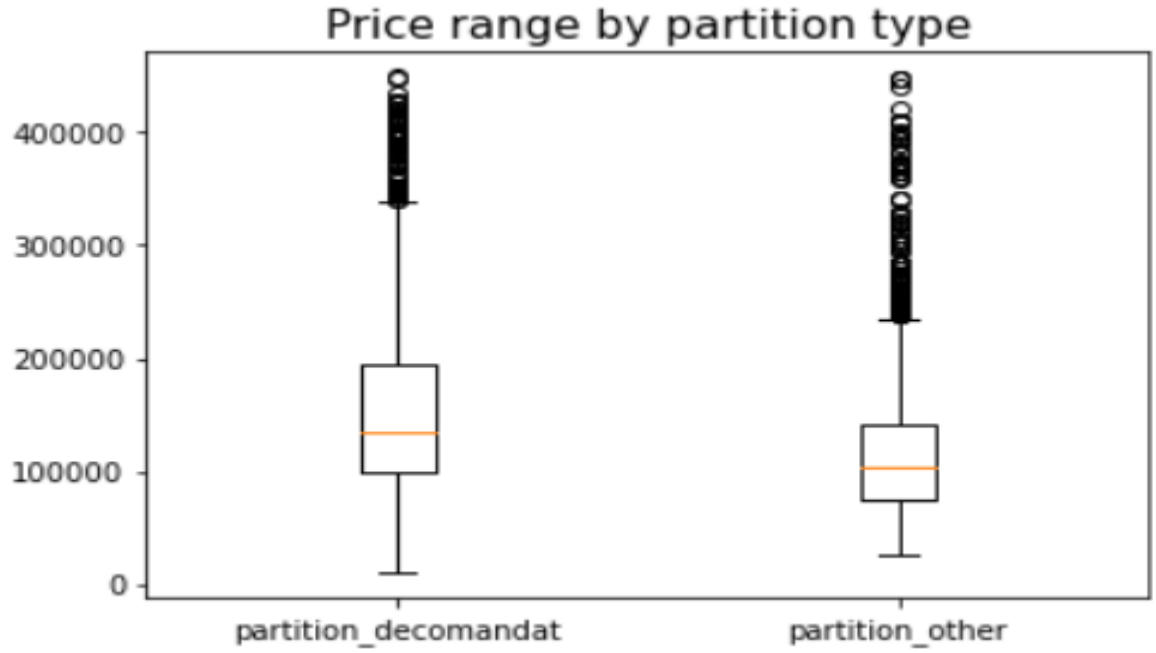
    - Titulescu_area
    - Turda_area
- The following variables were created from the specifications field:
    - has_floor_heating
    - only_district_heating
    - building_with_video_surveillance

Exploratory Visualization

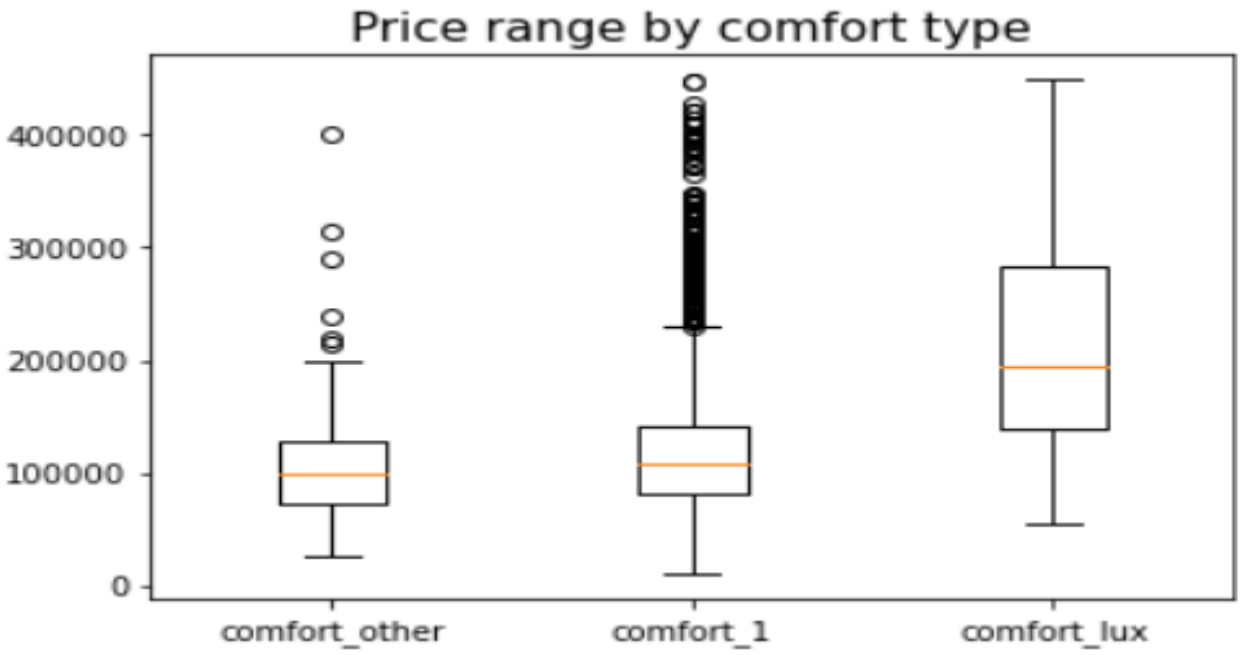
A major feature to explore is obviously price. As can be seen bellow, it follows a lognormal distribution.



When it comes to partition type, a subtle change in price distribution can be observed by looking at prices bellow the 95th percentile



Unsurprisingly, comfort_lux has the most expensive apartments, followed by comfort_1, as can be seen bellow.



Algorithms and Techniques

This project will use Sagemaker's AutoML. This means that the models will be automatically trained, built and tuned. AutoML is ideal for tabular data, which is the format I will be using. Since the problem I am solving is a regression problem, the following algorithms will be tried by the AutoML job:

- Deep Learning powered by MXnet
- Linear Learner Algorithm
- XGBoost Algorithm

The reason for choosing this approach is that it frees up a lot of time for the feature selection, one the most important and time-consuming aspects of machine learning. Since the algorithms used are proven to be robust, it is up to me to create the best features to help solve this problem.

Benchmark

The benchmark being used is inspired directly from imobiliare.ro. Currently the website offers a range for all properties within a neighborhood. The range goes from the minimum listing price of any property in the area all the way to the max listing price of any property in the area. A reasonable benchmark model would be *(Min Price + Max Price)/2*.

This would be a naïve model, which always predicts the same price for each neighborhood. Although it is an extremely simplistic benchmark, it is widely used and therefore any improvements upon it will surely have a serious impact.

III. Methodology

Data Preprocessing

The preprocessing involved:

- Removing invalid observations (missing essential properties such as surface, price, rooms)
- Creating new variables (such as bathroom_ratio, has_parking_spots_or_garages)
- Creating dummy variables for relevant categorical variables (for example neighborhood)

Due to the hands-off nature of AutoML, this is pretty much the only active preprocessing effort for this first iteration of this project.

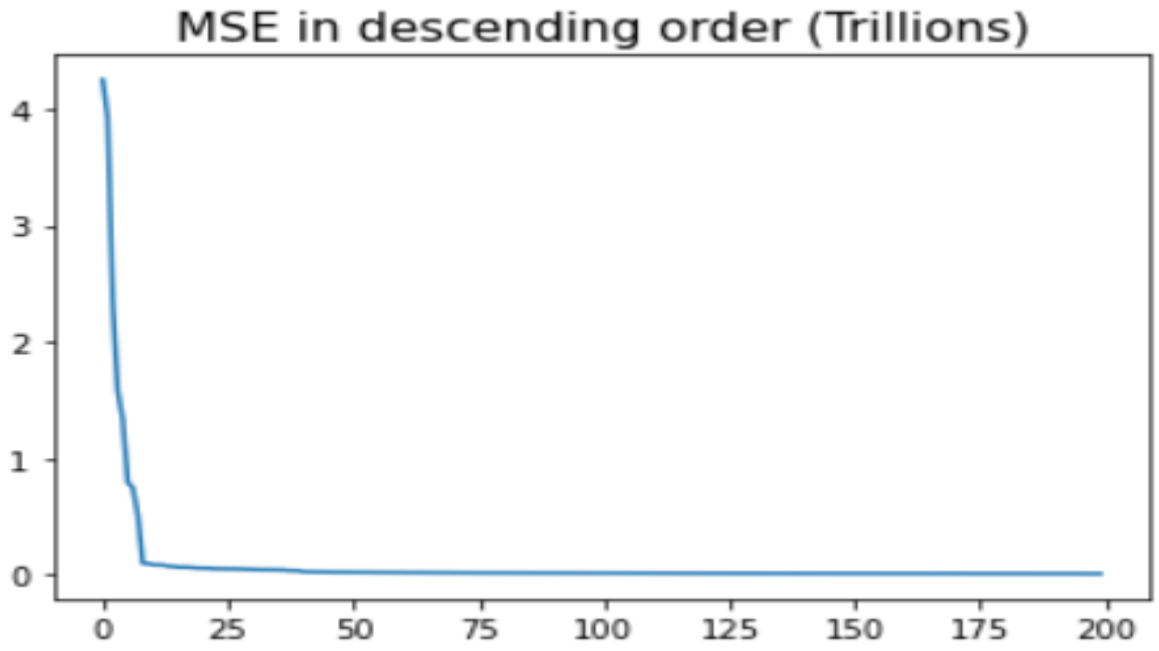
Implementation

As stated before, an AutoML job was created to determine the best model for this problem. It worked great, out of the box, given the classic nature of this problem. Tabular data for a regression problem, using MSE as a metric, which is different in a non-essential way from RMSE, the final metric for judging the performance of out model.

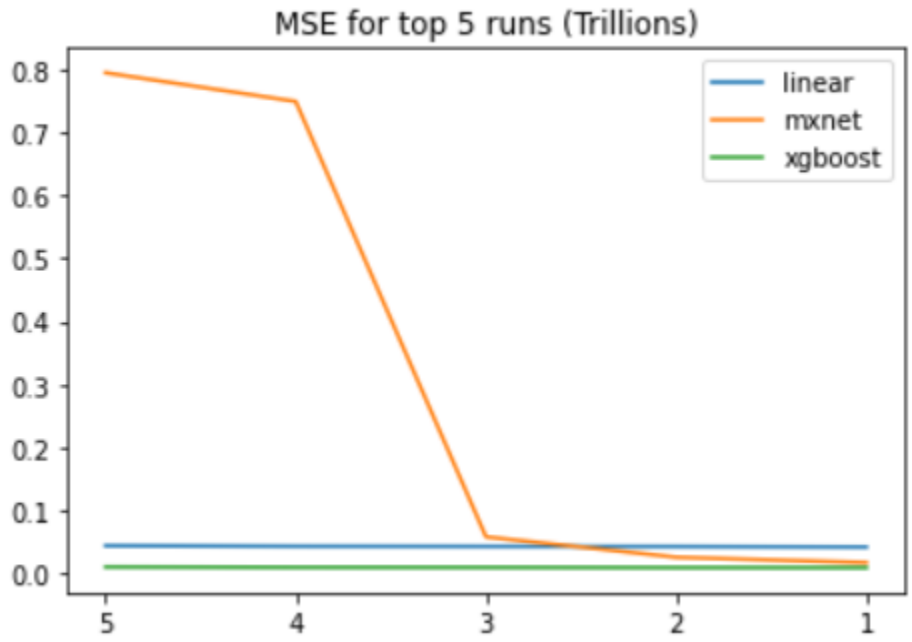
The code can be found in the file 03_train_test_automl.ipynb of this repository.

Refinement

AutoML tried multiple models, with varying MSE.



In the best candidate was chosen based on the desired metric, MSE. XGBoost has consistently proven to be the best.



IV. Results

Model Evaluation and Validation

The best candidate has proven itself to be the best among competing candidates. It should be noted however that the final test is for it to be tested completely new data.

The model has a RMSE of roughly 83969. The new data went through the same pipeline of preprocessing as the training data, which validates that it is great to use for the moment. Naturally, with time, as the real estate market undergoes changes, the current model will score poorer and poorer.

Justification

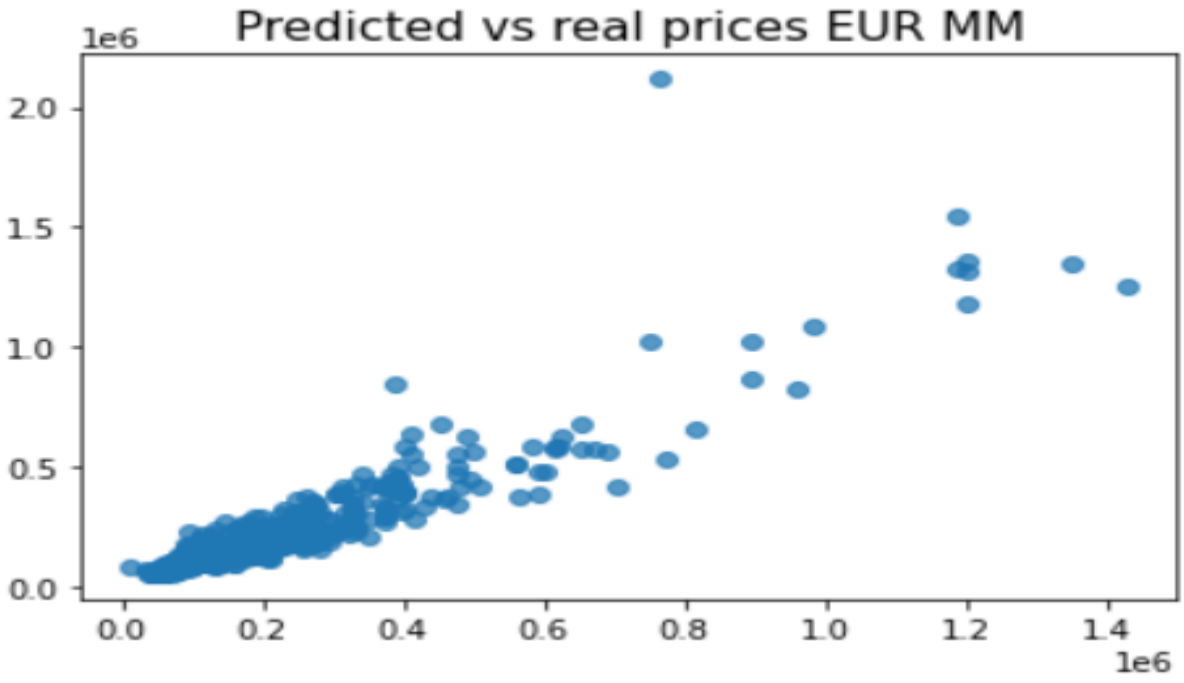
The model has a RMSE of 83969, while the benchmark only obtains a RMSE of 212331, calculated on new data, never before seen by the best candidate from the AutoML job.

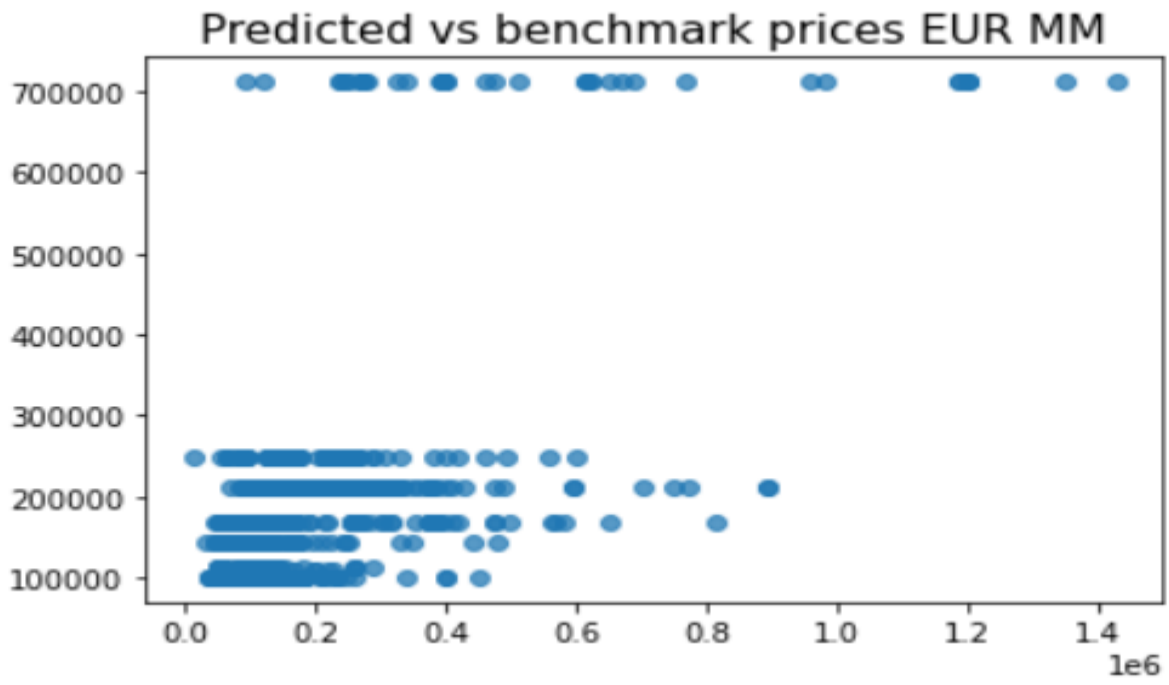
This is absolutely better, and therefore can constitute a serious improvement.

V. Conclusion

Free-Form Visualization

The quality of the resulting model vs the benchmark is obvious in the following plots:





Reflection

This entire journey has not been without its obstacles. To summarize, all the steps that were taken for this project were roughly:

- Scraping data
- Exploring & cleaning data
- Running an AutoML Job
- Exposing the best candidate via a Lambda function & API Gateway
- Building a simple form to allow for the model's consumption

To begin with, data quality was an important first issue, as it was a bit surprising for me that a listing can be displayed on imobiliare.ro even if it lacks something as basic as number of rooms. Luckily, this was the exception rather than the rule, and the same goes for every other important field that has missing data.

Another major step was exposing my model to the outside world, via Lambda functions & API Gateway. In this sense, I had some difficulty in figuring out how to properly install pandas in my Lambda environment.

Nevertheless, at the end of the day, it has been a great experience, and I believe the current solution can be the start of a legitimate project for imobiliare.ro to improve their service.

Improvement

There are multiple steps that can be taken in order to improve this project, such as:

- Getting more data. This can be solved through a partnership with imobiliare.ro. As it stands, the project has a limited pool of listings
- Getting a better target variable. Right now I used the asking price. This can be quite different from the selling price. A database with the selling prices is much more valuable.
- Getting better independent variables. No doubt, accurate geographic locations for the apartments will help the model immensely. Not only can this information be transformed into features in the model, but external data can be used to enhance the database. For example restaurants within an x mile radius can be used as a feature.
- Various other algorithms can be used, besides the default ones in the AutoML class.
- This whole process can be automated with the retraining of models being done after certain triggers