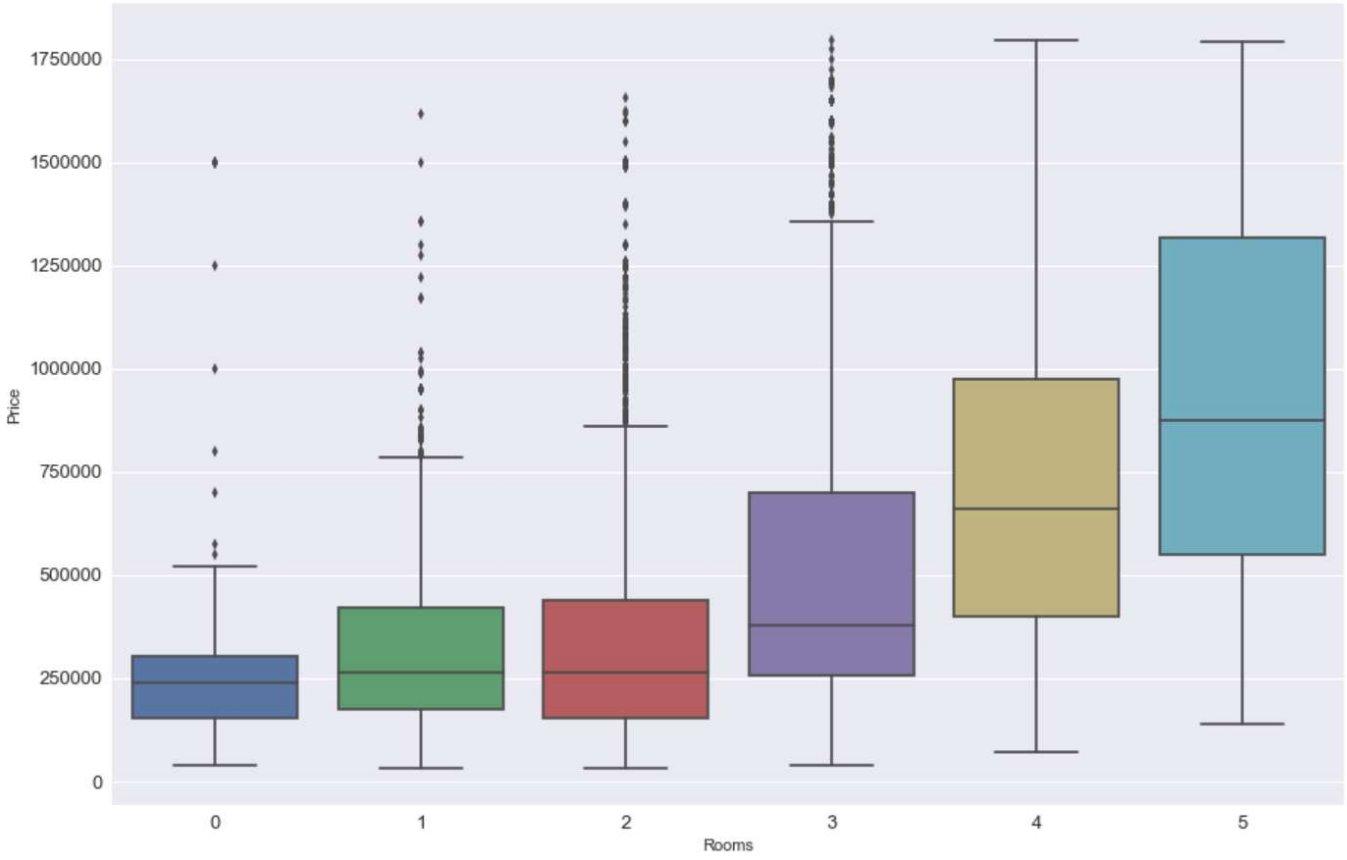
I was looking for a house, so I built a web scraper in Python! — part II (EDA)

Fábio Neves Nov 16, 2018 · 8 min read ★



the dataset we gathered before. I will then attempt to perform some exploratory data analysis (EDA) on the dataset. My goal is to give you some examples of what can be done, and not to provide a super complex analysis. But do feel free to comment and make suggestions! Some of the stuff we will see in this article:

- get the number of rooms for each house
- get the price per square meter
- remove outliers and "weird" observations
- using Seaborn to visualize our data

Let's start by having a look at what we got from scraping Sapo website before.

```
lisboa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22097 entries, 0 to 22096
Data columns (total 10 columns):
Title           22097 non-null object
Zone            14295 non-null object
Price           22097 non-null int64
Size (m²)       22097 non-null object
Status          22097 non-null object
Date            22097 non-null datetime64[ns]
Municipality    22097 non-null object
Description     21975 non-null object
URL             22097 non-null object
Image           22097 non-null object
dtypes: datetime64[ns](1), int64(1), object(8)
memory usage: 1.9+ MB
```

There is a date in the dataset, which stands for the date the ad was published. I already know there are some ads previous to 2018, and I think it's safe to assume these houses are not for sale anymore. To have a quick way to filter that, we can add two columns with the year and the month.

```
lisboa_clean = lisboa

years = []
months = []

for x in lisboa_clean['Date']:
    months.append(str(x.year) + '-' + str(x.month))
```

```
lisboa_clean['Month'] = months
```

Done! I'll also convert the "Size (m^2)" to numeric values, and for records that do not contain numeric values (like "-"), I will convert them to NaN (with *coerce*) and then I'll drop them since their value for this exercise will be none.

```
lisboa_clean['Size (m²)'] = pd.to_numeric(lisboa_clean['Size (m²)'],
errors='coerce')

lisboa_clean.dropna(subset=['Size (m²)'], inplace=True)
```

Although I also got the columns "URL" and "Image", we will not need them for this article, so we can simply remove them.

```
lisboa.drop(['URL', 'Image'], axis=1, inplace=True)
```

If you have done any kind of analysis with real estate data, you probably noticed that we still don't have the number of rooms for each property… That information was not available in the search pages (at least directly), so we will need to get a bit creative.

In every Title, there is a string like "T2", which tells us the house has 2 rooms. A T3 is 3 rooms and so on. My objective is to get the number, which can have one or two digits. Enter the (dreaded) regular expressions. The next lines of code will not look pretty, and I'm almost sure there must be an easier or more efficient way to do it, but after some testing, this seems to do the work fine!

The code goes through every title in the dataset, and retrieves to a new list (*rooms*) the strings that match the regular expression below, i.e. "any string that starts with a 'T' and has 1 or 2 digits after". We end up with these unique values:

```
rooms = []
for h in lisboa_clean.Title:
    n = [(s) for s in re.findall(r'T\d{1,2}' , h)]
    rooms.append(n)
```

```
array([ 'T3',  'T2',  'T5',  'T4',  'T1',  'T6',  'T0', None,  'T8',  'T7',  'T10',
       'T9', 'T12', 'T13', 'T15', 'T20'], dtype=object)
```

It seems that some houses have “None” rooms. We should be fine dropping them and proceeding, but just in case, this bit of code tells us how many houses did we lose by doing this.

```
lisboa_clean['Rooms'] = roomsT
print(len(lisboa_clean['Rooms']))
lisboa_clean.dropna(subset=['Rooms'], inplace=True)
lisboa_clean.reset_index(inplace=True, drop=True)
print(len(lisboa_clean['Rooms']))
```

```
21806
19796
```

A total of 2010 houses were dismissed. Not a serious issue for us though!

After removing the “None”, we can proceed with getting just the numbers. One more cell with a *for* loop will take care of that. This loop will convert to integer the characters after the letter T.

```
r=[]
for x in lisboa_clean['Rooms']:
    r.append(int(x[1:]))

lisboa_clean['Rooms'] = r
```

Done! After this, adding the price per square meter should be piece of cake. And then we can check what *describe* tells us about what we did to our dataset.

```
lisboa_clean['Price/m²'] = lisboa_clean['Price'] / lisboa_clean['Size (m²)']
```

```
pd.set_option('display.float_format', lambda x: '%.2f' % x) #this will make the dfs not show scientific notation for price
lisboa_clean.describe()
```

	Price	Size (m²)	Rooms	Price/m²
count	19796.00	19796.00	19796.00	19796.00
mean	582182.49	149.45	2.51	4430.36
std	1980298.62	1149.17	1.16	16677.21
min	32000.00	1.00	0.00	2.48

```
max  84316872.00  93000.00  20.00  695772.26
```

```
lisboa_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19796 entries, 0 to 19795
Data columns (total 12 columns):
Title           19796 non-null object
Zone            12355 non-null object
Price           19796 non-null int64
Size (m²)       19796 non-null float64
Status          19796 non-null object
Date            19796 non-null datetime64[ns]
Municipality    19796 non-null object
Description     19706 non-null object
Year            19796 non-null object
Month           19796 non-null object
Rooms           19796 non-null int64
Price/m²        19796 non-null float64
dtypes: datetime64[ns](1), float64(2), int64(2), object(7)
memory usage: 1.8+ MB
```

with all this process, we are down to 19796 properties

Time to clean up!

This is probably the most important part of the whole process. Ultimately, it's the quality of your work in this stage that will define the quality of your analysis or model. The fact that there is no exact rule for this, makes it difficult to decide how much data should we keep/drop, and that's where a bit of common sense comes in handy. A good rule of thumb is to get rid of less than 3% of your data. If you throw away more and more data, you will be limiting the amount of information you have to work with.

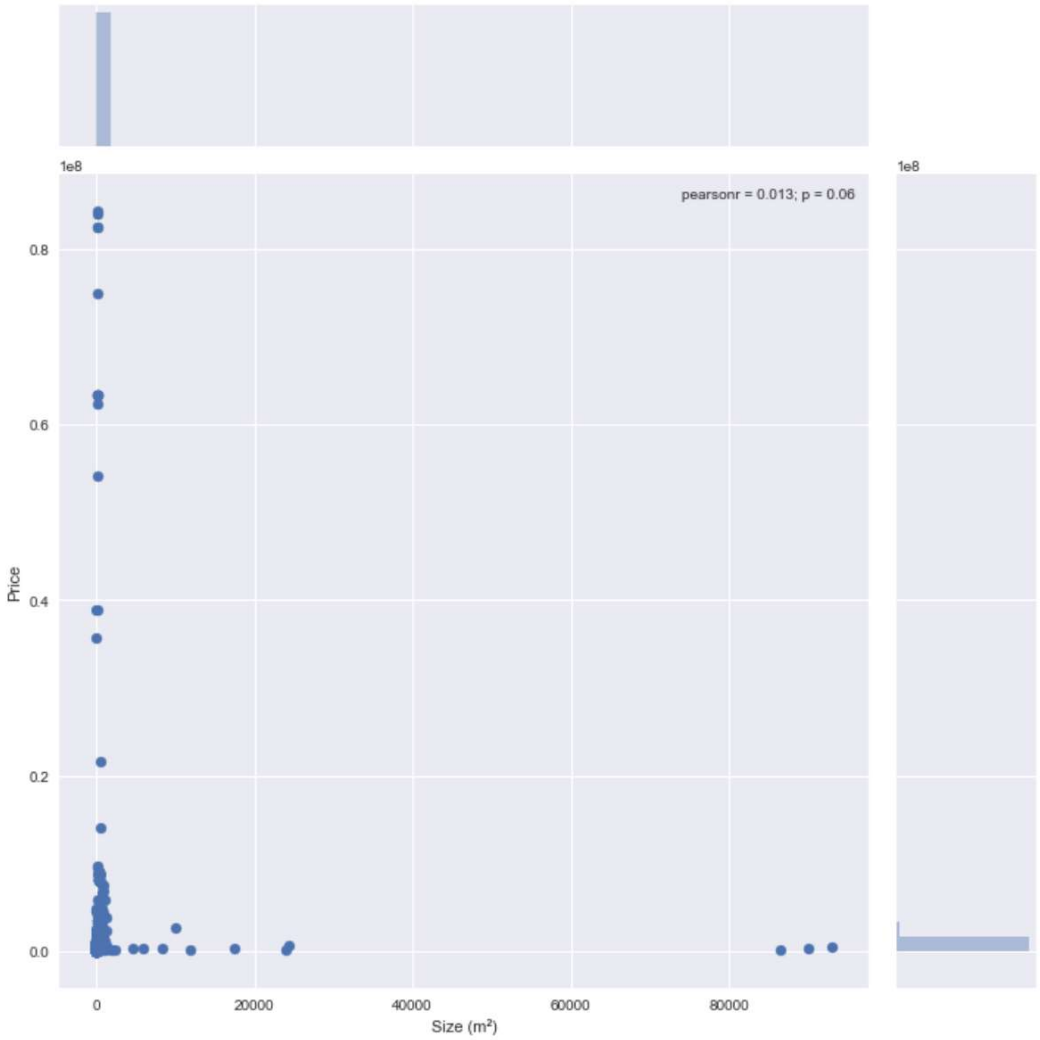
> *The goal here is to always have some kind of reasoning for the choices you make while pre-processing your data.*

And my *choice* for this article is to not care much about the 3%! Right away we can see that there are some strange values in our data, more precisely a house that costs more than 80 million euros, or a house with 93 square kilometers. These extreme values (outliers) will have a huge impact in measures like the mean. The "proof" is that although the median price is about 350k, the mean price is around 582k.

> ***The median is less sensitive to outliers**, which makes it a nice tool in situations like these.*

outliers. Notice the pearson correlation between Price and Size is very low, as well as the p-value (which is an indicator for statistical significance — basically, if it is lower than 0.05 it means the correlation coefficient is significant).

```
sns.jointplot(x='Size (m²)', y='Price', data=lisboa_clean, size=(10))
plt.show()
```



"not looking good" is an understatement...

After some testing, I came up with the following filters for the dataset:

- I'm not looking to buy a castle yet, so I'll remove any house with more than 600 square meters
- Same for houses with more than 5 rooms

On the other hand, I do find it weird that there might be millionaires shopping for houses in such a generic website like this. It just doesn't seem the right type of platform for buying or selling luxury homes, so we could actually argue that these properties are not representative of the market we're trying to analyze here.

In a nutshell, that's how I would describe what Data Science is all about. Coming up with these stories about your data and putting your theories (or the *hypothesis*) to the test.

Back to our data, let's remove the observations I mentioned above. With *len* we can see how many records we are throwing away.

```
len(lisboa_clean[lisboa_clean['Size (m²)'] > 600])
```

```
99
```

```
len(lisboa_clean[lisboa_clean['Price'] > 1800000])
```

```
472
```

```
len(lisboa_clean[lisboa_clean['Rooms'] > 6])
```

```
77
```

```
lisboa_clean = lisboa_clean[lisboa_clean['Size (m²)'] < 600]

lisboa_clean = lisboa_clean[lisboa_clean['Price'] < 1800000]

lisboa_clean = lisboa_clean[lisboa_clean['Rooms'] < 6]
```

What about the distribution for the newly created Price/Size? We have some cleaning to do here too. The average price per square meter is around 5k (see the *describe* method before) so I'll remove houses with this value higher than 10k. In the opposite extreme, I'll also remove observations lower than 300 euros per square meter.

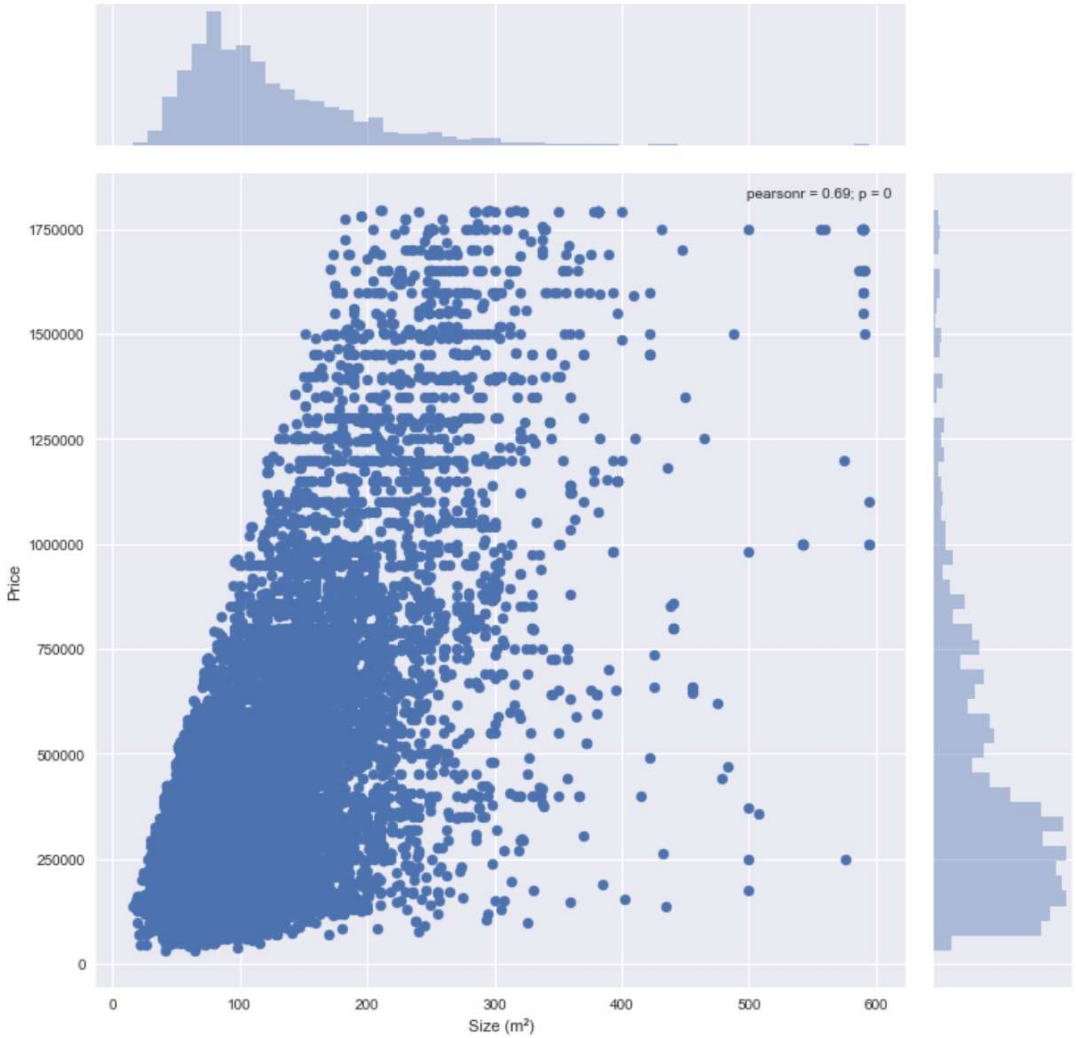
```
len(lisboa_clean[(lisboa_clean['Price/m²'] > 10000) | (lisboa_clean['Price/m²'] < 300)])
```

```
167
```

```
lisboa_clean.reset_index(inplace=True, drop=True)
```

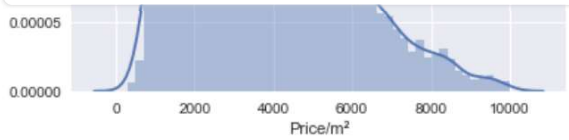
Time to check that *jointplot* again. We're basically smoothing the distributions by removing the extreme values from our data. This will allow for better models, if we were to build one for predicting house prices, for instance.

```
sns.jointplot(x='Size (m²)', y='Price', data=lisboa_clean, size=(10))
plt.show()
```



notice how the pearson correlation coefficient suddenly improved to 0.69, just by cleaning noise from the dataset

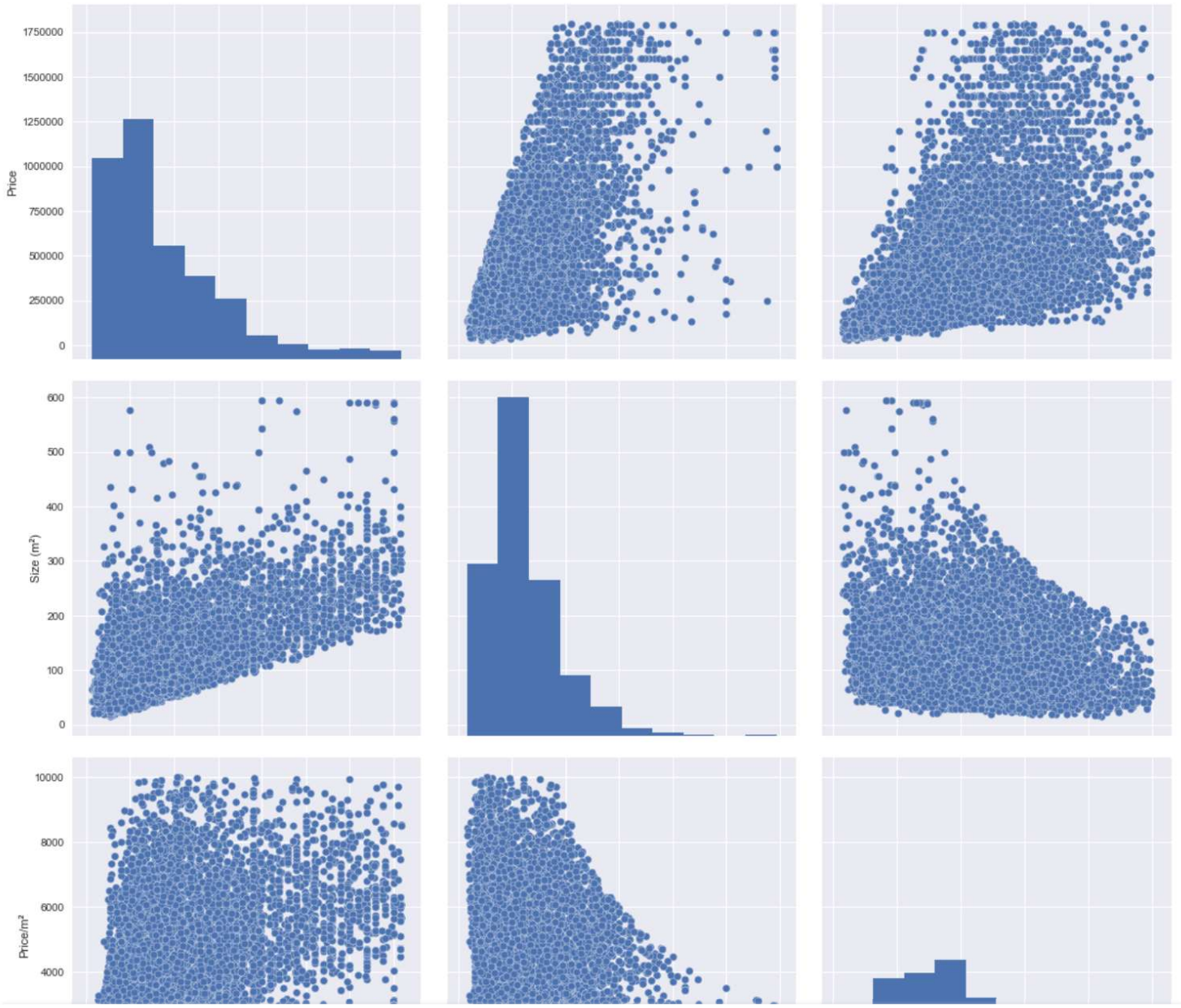
```
sns.distplot(lisboa_clean['Price/m²'])
plt.show()
```

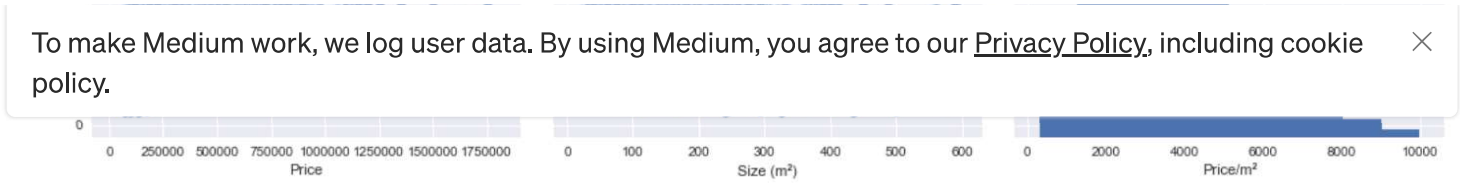


the price per square meter is also more balanced

And in the spirit of diversity, I'll share just another type of chart, the *pairplot*. This one is really cool, but can get unreadable if you add too many variables. There isn't a lot of valuable insights here, but it does look fancy!

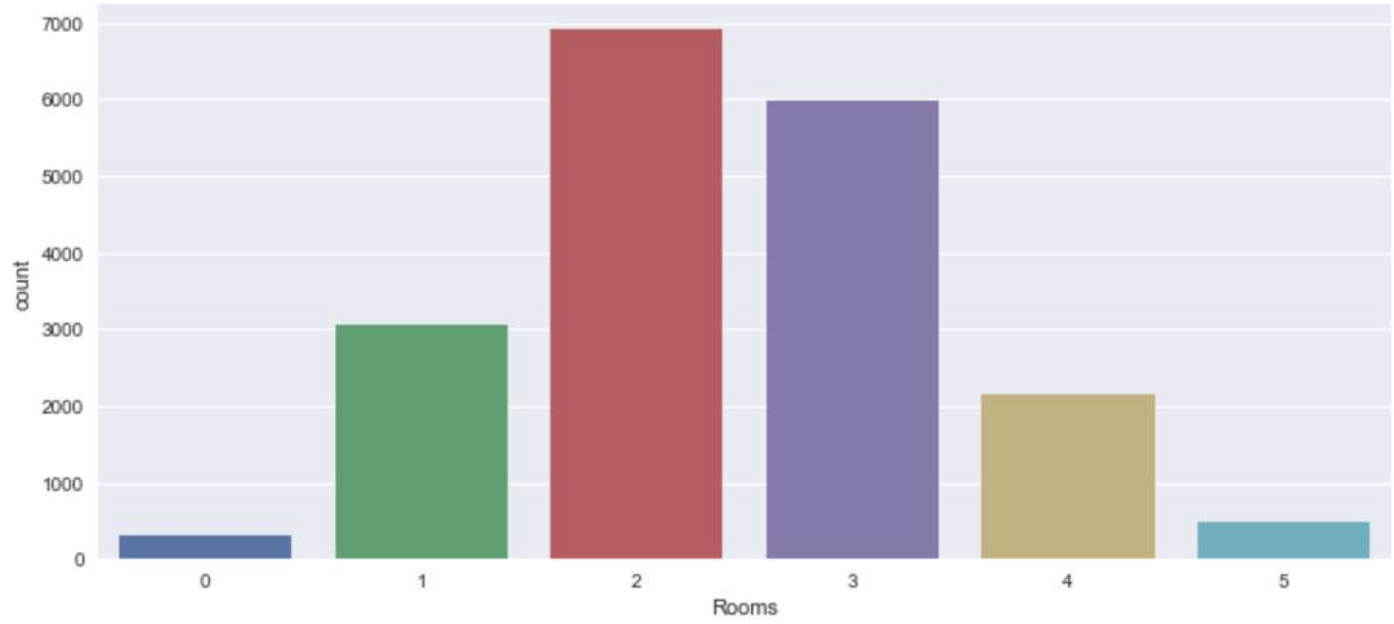
```
cols = ['Price', 'Size (m²)', 'Price/m²']
sns.pairplot(lisboa_clean[cols], size = 5)
plt.show()
```

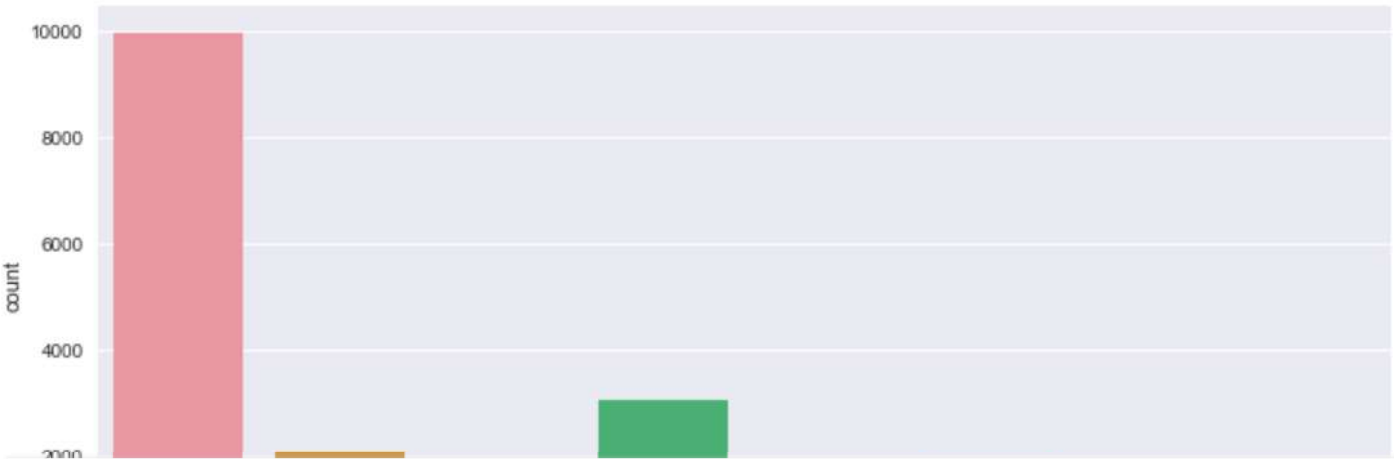


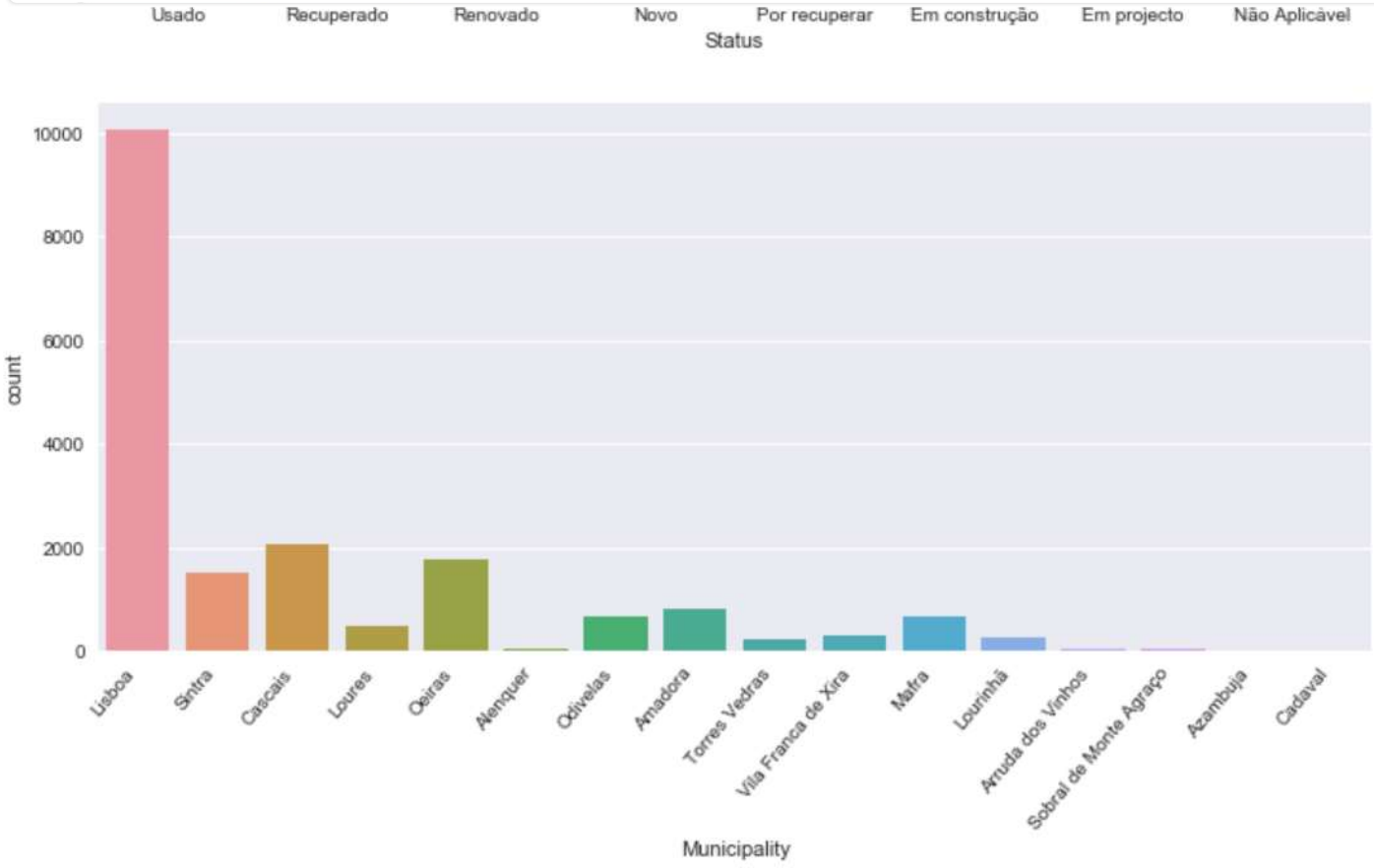


We can also explore the remaining categorical variables like “Status”, “Municipality” and “Rooms”.

```
fig, ax =plt.subplots(3,1, figsize=(12,20))
sns.countplot(lisboa_clean['Rooms'], ax=ax[0])
sns.countplot(lisboa_clean['Status'], ax=ax[1])
sns.countplot(lisboa_clean['Municipality'], ax=ax[2])
ax[2].set_xticklabels(ax[2].get_xticklabels(), rotation=50,
ha="right")
plt.show()
```





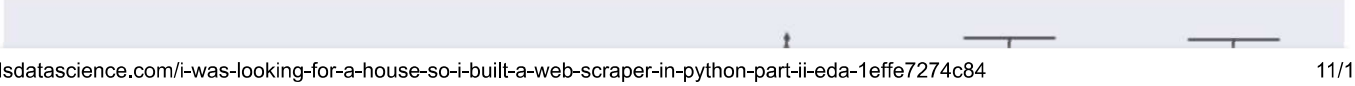


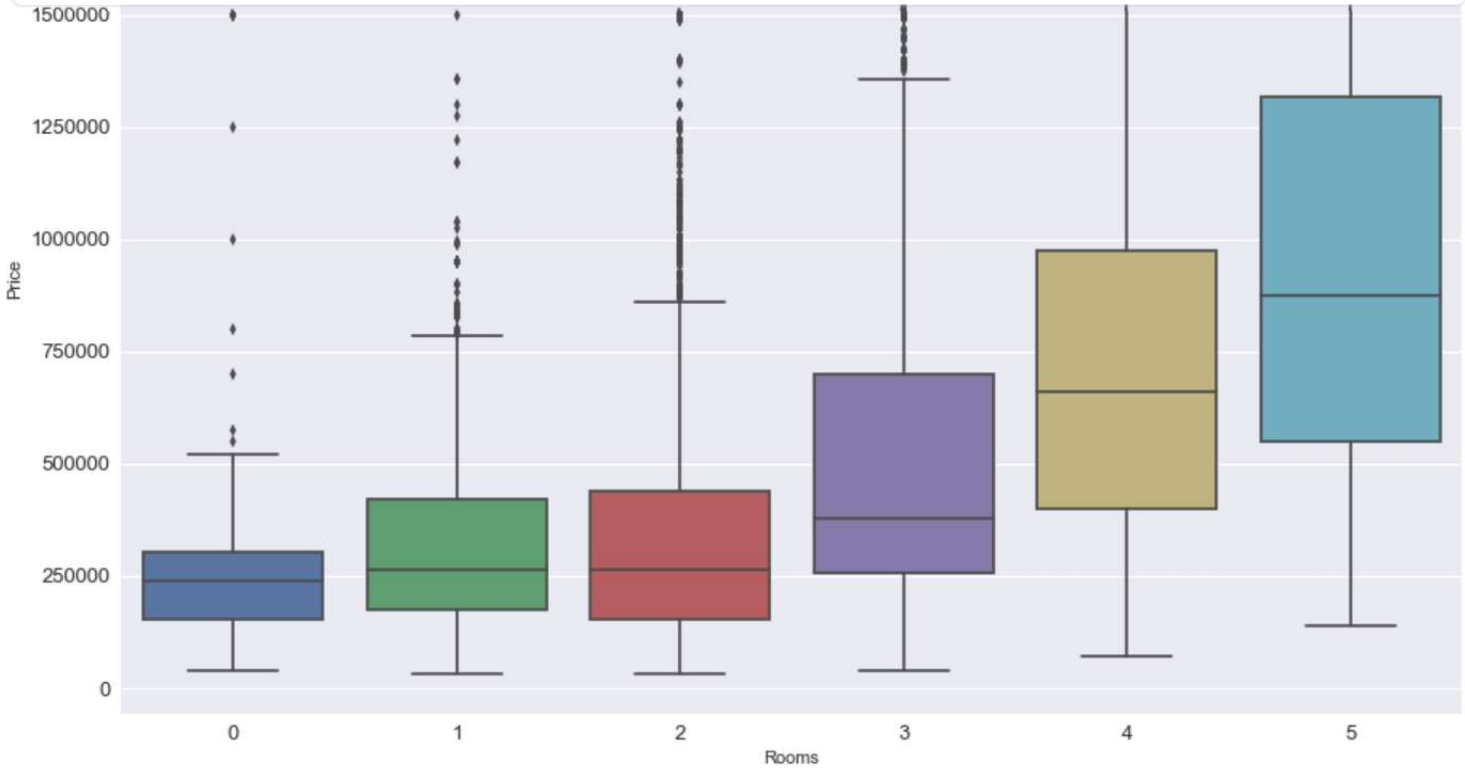
Unsurprisingly, the main municipalities in Lisboa district are Lisboa, Cascais and Oeiras. This may not mean much to someone who is not familiar with the city, but these are basically the prime locations in the district.

The number of houses with 2 rooms is more than double the number of houses with 1 room. I guess it does makes sense that the majority of houses has more than 1 room, I just never gave it too much thought before!

Let's wrap things up with some some *boxplots* for these variables.

```
plt.figure(figsize=(15,10))
ax = sns.boxplot(x='Rooms', y='Price', data = lisboa_clean)
ax.tick_params(labelsize=13)
plt.show()
```



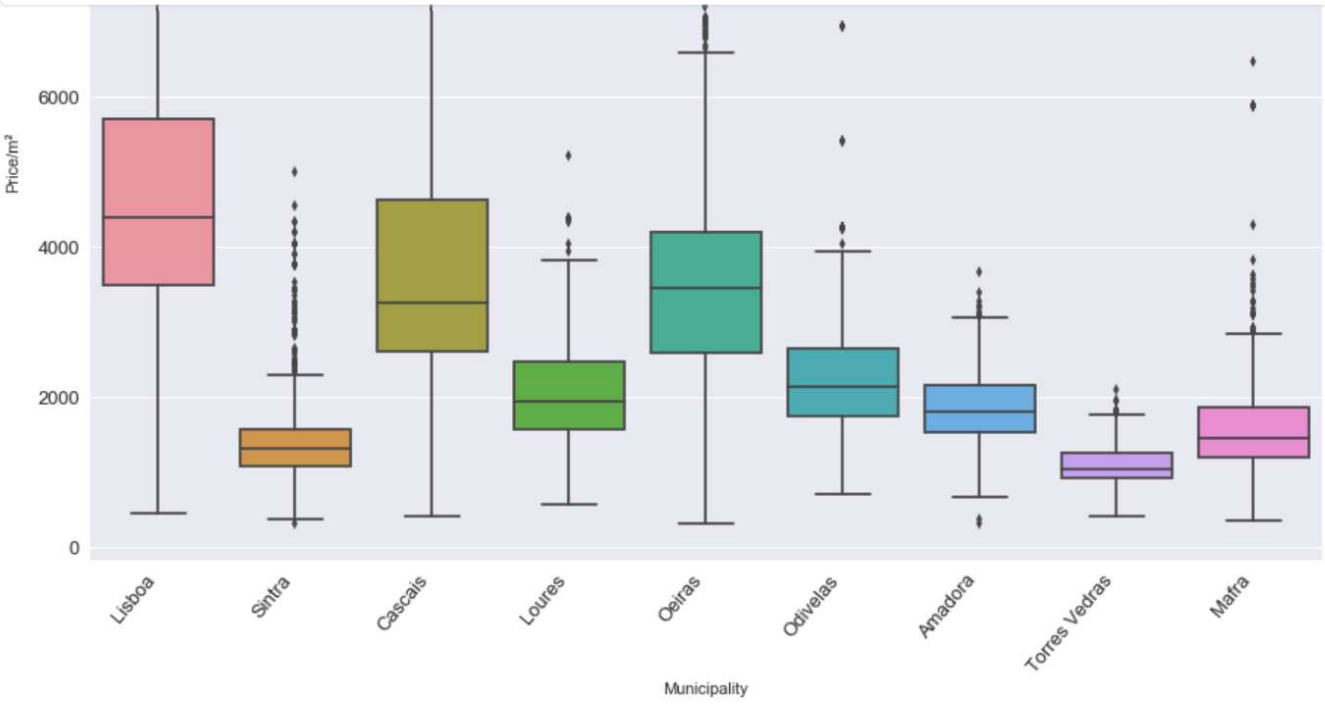


Again, pretty obvious conclusions since we expect the value of the properties to go up as the number of rooms increases. The interesting aspect in these chart may be the fact that 1 and 2 rooms houses are competing in the same price range. There are virtually infinite ways we can further explore this information.

Final *boxplot*, I promise! Let's check the differences in price per square meter in a specific set of municipalities

```
areas = lisboa_clean.loc[lisboa_clean['Municipality'].isin(['Lisboa',
'Amadora', 'Cascais', 'Oeiras', 'Sintra', 'Loures', 'Odivelas',
'Mafra', 'Torres Vedras'])]
```

```
plt.figure(figsize=(15,10))
ax = sns.boxplot(x='Municipality', y='Price/m²', data = areas)
ax.set_xticklabels(ax.get_xticklabels(), rotation=50, ha="right")
ax.tick_params(labelsize=13)
plt.show()
```



There is a huge amount of possibilities when it comes to exploring and representing data. I hope this article gives you some useful tips on how to better understand your data! Even though we didn't find any mind-blowing fact about the real estate market in Lisbon, I think we were able to see that a few simple steps like removing extreme values, can improve the considerably the quality of our analysis.

Thanks for reading! If you liked this article, I invite you to check my other stories. I'm mainly interested in Data Science, Python, Blockchain and digital currencies, technology, and a few other things like ***photography****! If you'd like to get in touch, you can contact me* ***here*** *or simply reply to the article below.*

Data Science Python Exploratory Data Analysis Towards Data Science Programming