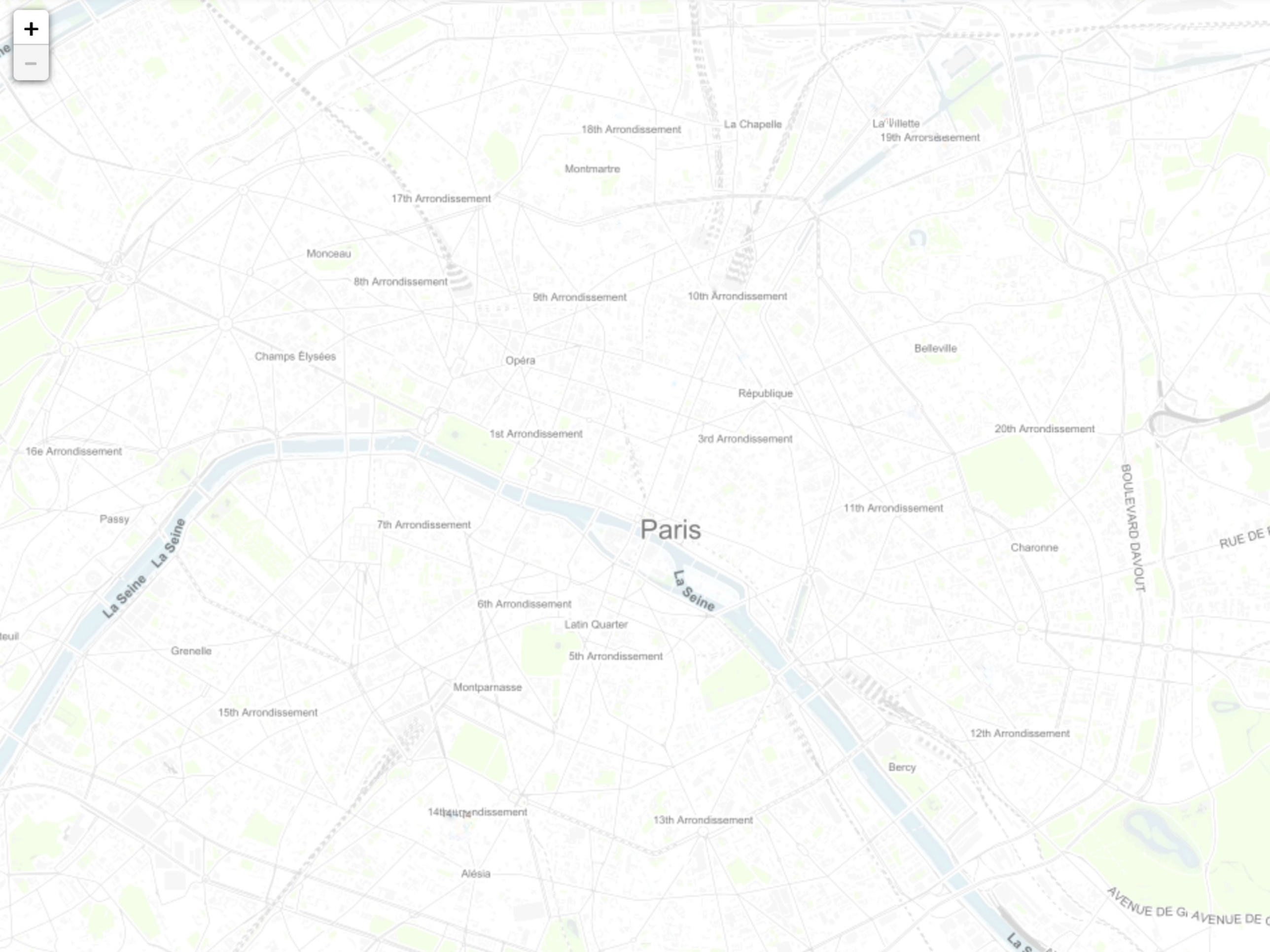
KAGGLE COMPETITION

PREDICTING THE RIGHT PRICE OF AN AIRBNB ACCOMMODATION

Toulouse School of Economics


18th Arrondissement
La Chapelle
La Villette
19th Arrondissement
Montmartre
17th Arrondissement
Monceau
8th Arrondissement
9th Arrondissement
10th Arrondissement
Champs Élysées
Opéra
Belleville
République
1st Arrondissement
3rd Arrondissement
20th Arrondissement
16e Arrondissement
Passy
7th Arrondissement
Paris
11th Arrondissement
Charonne
La Seine
La Seine
La Seine
6th Arrondissement
Latin Quarter
5th Arrondissement
Grenelle
Montparnasse
15th Arrondissement
12th Arrondissement
Bercy
13th Arrondissement
Alésia
BOULEVARD DAVOUT
RUE DE
AVENUE DE GI
AVENUE DE

7th December 2018

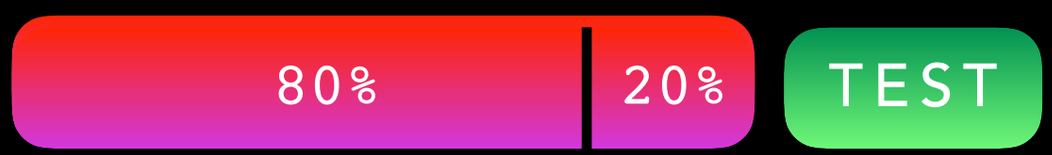
PROBLEM STATEMENT

In view of the number of housing units in Paris, how to define a fair price according to the location, the equipment, or the capacity of a unit?

Regression problem

DATA PRESENTATION

- Two files: - train $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
  -test $\{x_{n+1}, \ldots, x_N\}$



- 57 predictors
- Goal: learn on the train set to predict $\{\hat{y}_{n+1}, \ldots, \hat{y}_N\}$

DATA PREPROCESSING

Dropping everything that seems irrelevant at first view

- No NLP, so we drop text columns

```
name
neighborhood_overview
notes
transit
access interaction
house_rules
```

- Columns that don't seem to be helpful for price prediction

```
host_id
calendar_updated
availability_30
availability_60
availability_90
availability_365
```

- Redundant columns

```
host_since
reviews_per_month
```

DATA PREPROCESSING

Keeping some variables related to location

country	∈ {'France', nan}	⇒ useless
department	∈ {'Hauts-de-Seine','Paris', 'Seine-Saint-Denis,'Val-de-Marne'}	⇒ useless
geolocation + geopoint	redundant with latitude and longitude	⇒ useless
neighbourhood_cleansed	20 unique values + no missing values	⇒ we keep it
city	redundant with neighbourhood_cleansed	⇒ useless
zipcode	55 unique values + 12 missing values	⇒ we keep it

DATA PREPROCESSING

Variables with too many missing values

VARIABLE	% OF MISSING VALUES
host_response_time	33.37
host_response_rate	33.37
first_review	25.50
last_review	25.50
review_scores_rating	26.96
review_scores_accuracy	27.15
review_scores_cleanliness	27.09
review_scores_checkin	27.02
review_scores_communication	27.11
review_scores_location	27.18
review_scores_location	27.18
review_scores_value	27.19

- review scores are important when choosing an Airbnb

- dropping all rows where a value is missing would result in loosing a lot of information

⇓

we keep them for now

CLEANING VARIABLES, ONE-BY-ONE

`host_response_time (categorical)`

Among the listings with a missing `host_response_time`, `47.21%` have `nb_reviews=0`!

=> Case of posted properties that hadn't yet received any review at the moment when the web-scraping was made

=> So in fact it's not really a "missing" value

=> For `host_response_time` we replace `nan` by `"undetermined"`

CLEANING VARIABLES, ONE-BY-ONE

`host_response_rate (numerical)`

We want to use same treatment, so we transform it into categorical

=> Proportion that respond all the time is `70.37%`

=> We split and replace `nan` by `"undetermined"`

CLASS	COUNTS
100%	20724
undetermined	14749
51-95%	5862
0-50%	1862
96-99%	1000

CLEANING VARIABLES, ONE-BY-ONE

`host_verifications (categorical)`

=> We replaced `nan` by median

=> We simply keep the number of verifications `(numerical)`

`latitude, longitude (numerical)`

=> Only one missing in the train database so we dropped it

`security_deposit, cleaning_fee (categorical)`

=> Both are well-balanced, no treatment needed

`first_review => days_since_first_review`

CLEANING VARIABLES, ONE-BY-ONE

property_type (categorical)

Some categories with too few observations

CLASS	% VALUE_COUNTS
Apartment	95.45
Loft	1.05
House	0.74
B&B	0.53
Condominium	0.30
Other	0.30
…	…

=> We group them

CLASS	VALUE_COUNTS
Apartment	43137
House	609
Hotel	450

CLEANING VARIABLES, ONE-BY-ONE

first_review (date)

Some categories with too few observations

CLASS	% VALUE_COUNTS
Apartment	95.45
Loft	1.05
House	0.74
B&B	0.53
Condominium	0.30
Other	0.30
…	…

=> We group them

CLASS	VALUE_COUNTS
Apartment	43137
House	609
Hotel	450

CLEANING VARIABLES, ONE-BY-ONE

room_type (categorical)

CLASS	% VALUE_COUNTS
Entire	85.74
Home / Appt	13.2
Private room	1.05
Shared room	

Few instances of shared room but we keep it unchanged

Shared room

+

−

18th Arrondissement

La Chapelle

La Villette

19th Arrondissement

Montmartre

17th Arrondissement

Monceau

8th Arrondissement

9th Arrondissement

10th Arrondissement

Champs Élysées

Opéra

Belleville

République

1st Arrondissement

3rd Arrondissement

20th Arrondissement

16e Arrondissement

Passy

La Seine

La Seine

7th Arrondissement

Paris

La Seine

11th Arrondissement

Charonne

6th Arrondissement

Latin Quarter

Grenelle

5th Arrondissement

Montparnasse

15th Arrondissement

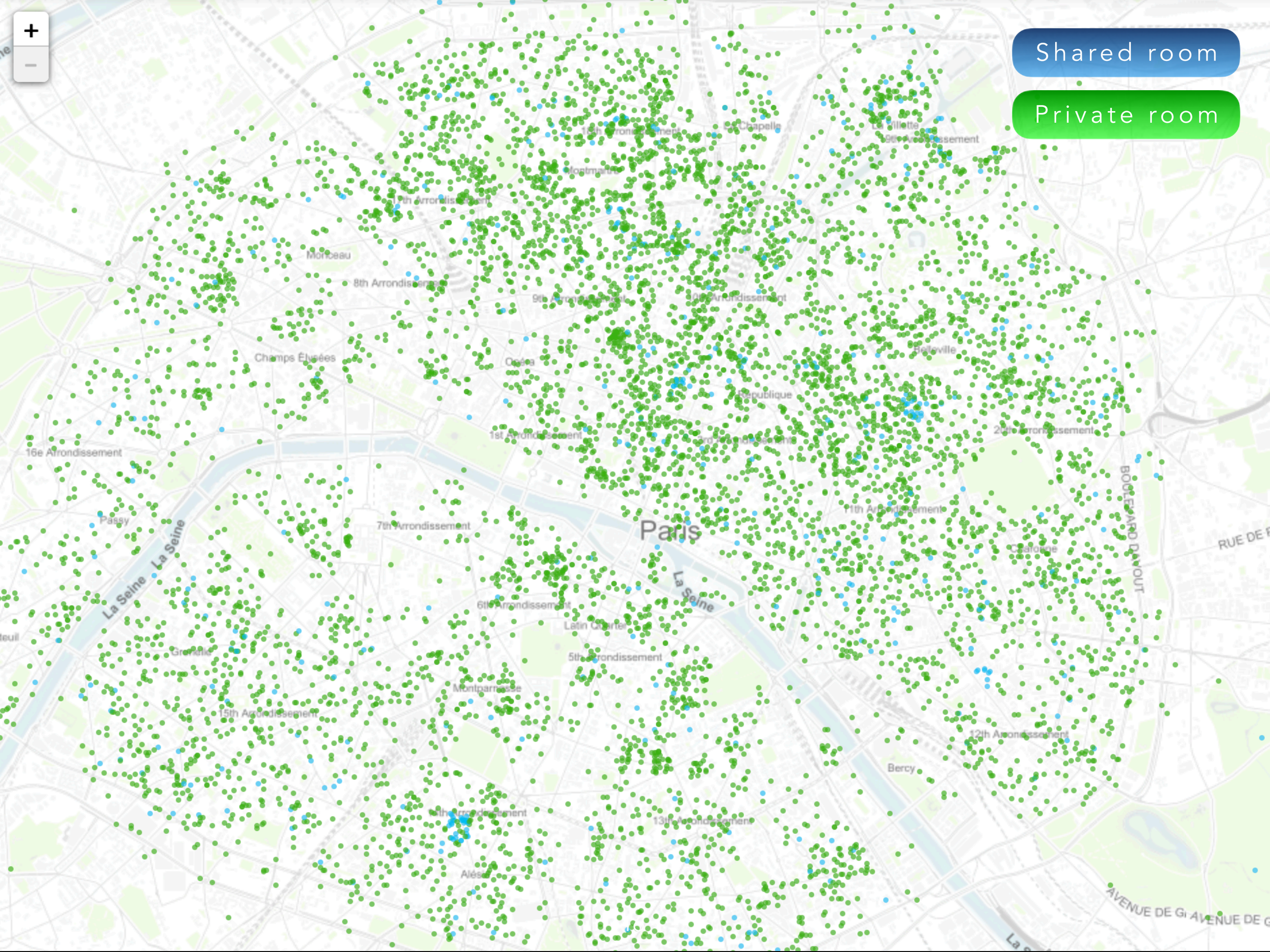
12th Arrondissement

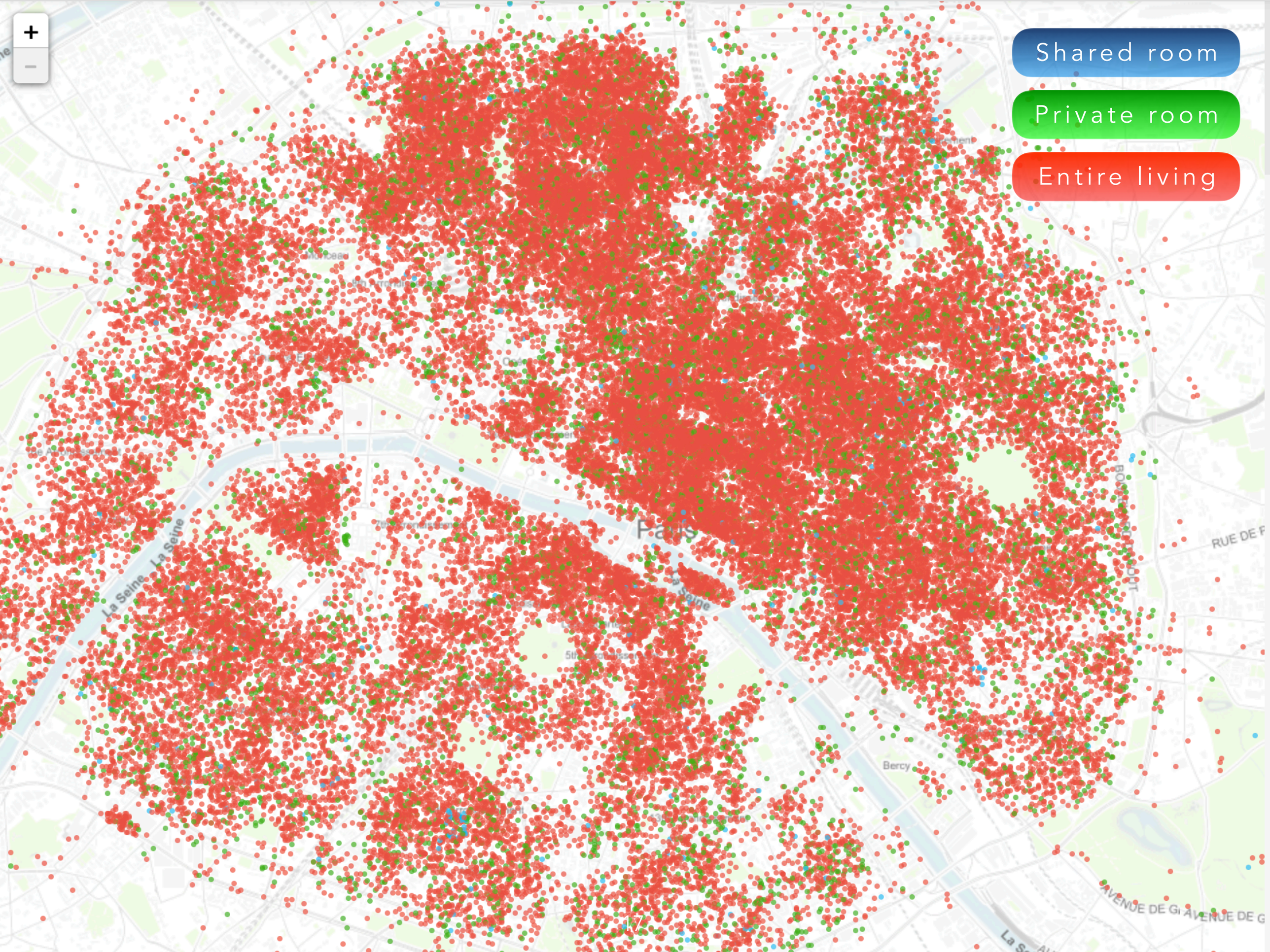
Bercy

14th Arrondissement

13th Arrondissement

Alésia


Shared room
Private room
Paris
La Seine
Montmartre
9th Arrondissement
8th Arrondissement
Champs Élysées
Opéra
1st Arrondissement
16e Arrondissement
Passy
7th Arrondissement
6th Arrondissement
Latin Quarter
5th Arrondissement
Montparnasse
15th Arrondissement
13th Arrondissement
Bercy
12th Arrondissement
11th Arrondissement
Belleville
République
20th Arrondissement
BOULEVARD DAVOUT
RUE DE
AVENUE DE


Shared room
Private room
Entire living
Paris
La Seine
La Seine
Bercy

CLEANING VARIABLES, ONE-BY-ONE

`amenities (categorical)`

One line looks like this

```
TV, Internet, Wireless Internet,
Kitchen, Buzzer/wireless intercom,
Heating, Family/kid friendly,
Hair dryer, Iron
```

- `nans` were replaced by the mode
- One dummy for every amenity
- Only keep the ones which appear in no more than `90%` nor less than `10%` of the listings

CLEANING VARIABLES, ONE-BY-ONE

`features (categorical)`

One line looks like this

```
Host Has Profile Pic,
Host Identity Verified
```

- `nans` were replaced by the mode
- One dummy for every feature
- Only keep the ones which appear in no more than `90%` nor less than `10%` of the listings

CLEANING VARIABLES, ONE-BY-ONE

`review_scores (numerical)`

- Many reviews are 10% (or 100%) so we convert them into categorical variables

  `'0-9%', '10%'`

  `'0-79%', '80-94%', ’95-100%'`

- We replace `nan` by `"undetermined"`

CLEANING VARIABLES, ONE-BY-ONE

Numerical variables left

`host_seniority`
`host_listings_count`
`accommodates`
`bathrooms`
`bedrooms`
`beds`
`guests_included`
`extra_people`
`minimum_nights`
`maximum_nights`
`number_of_reviews`
`price`

Descriptive statistics

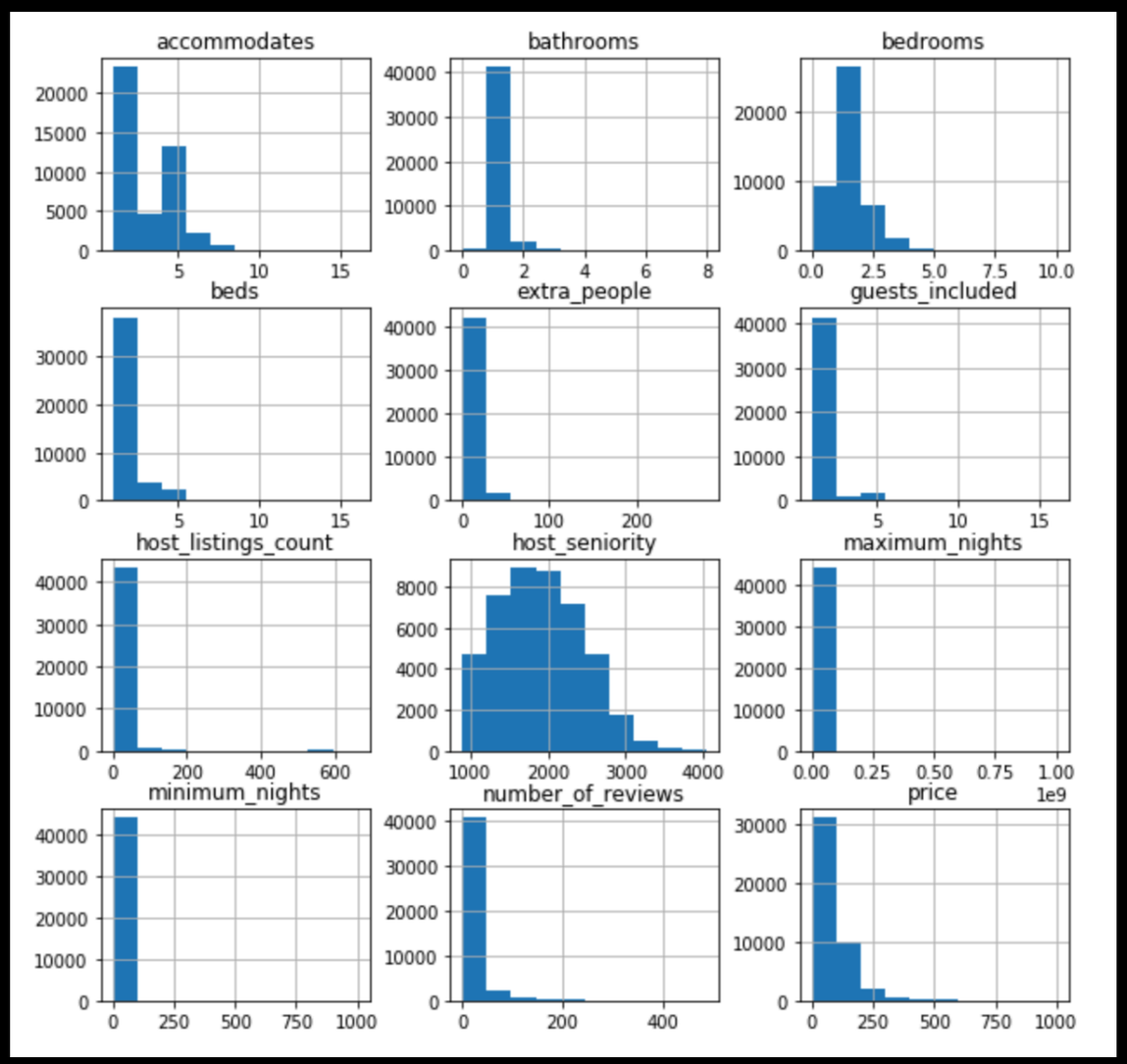
- `host_listings` has max at 661 (is it even possible?!)
- `extra_people` has max at 281 (?!)
- `maximum_nights` has max at 1e+9. It may be a value by default
- minimum `price` is 0

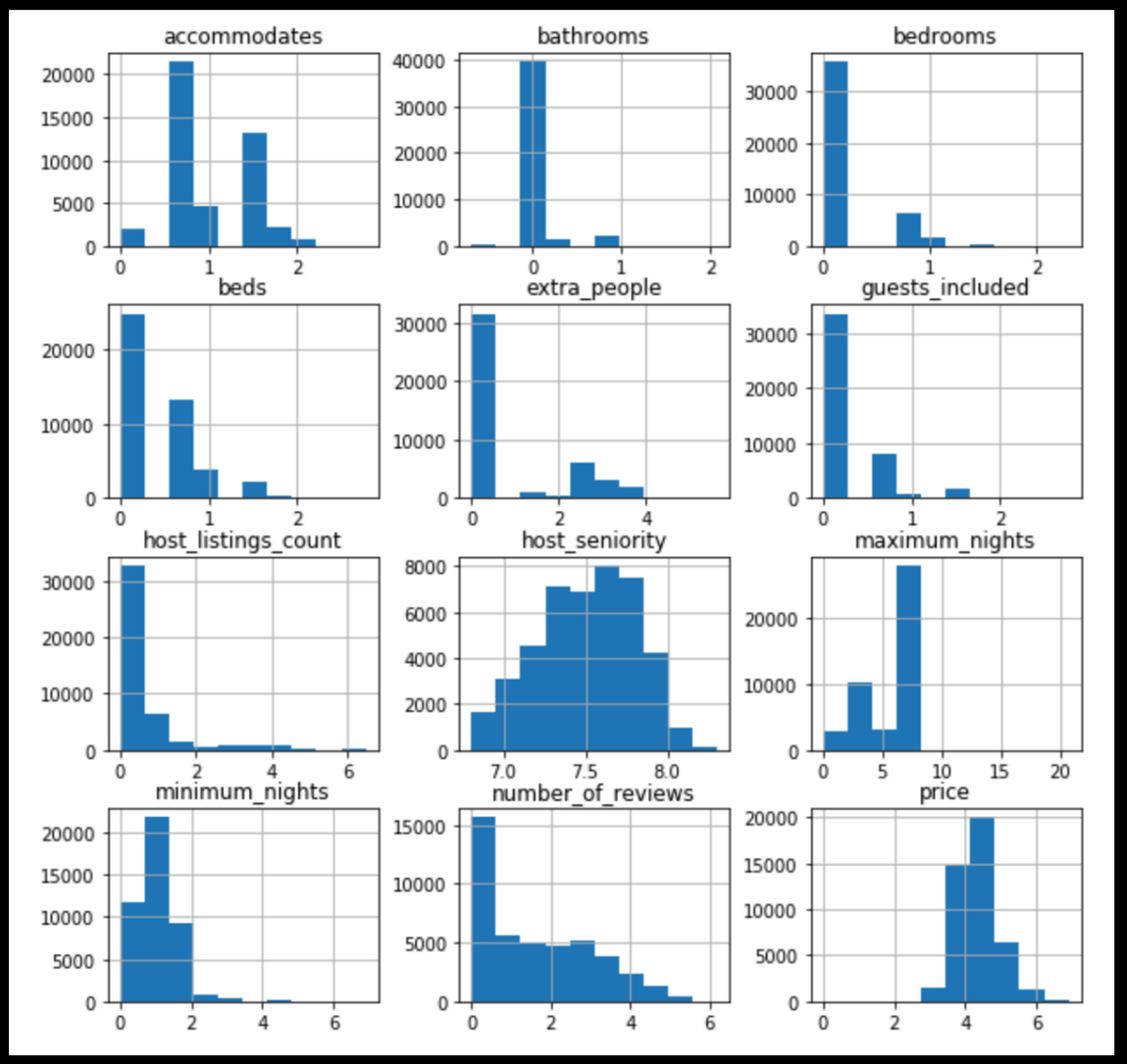
CLEANING VARIABLES, ONE-BY-ONE

Numerical variables left

Nearly all our variables are right-skewed (`mean >> median`)

=> We do a `log(x+1)`-transformation (brings closer to normality)





CREATING NEW FEATURES

VARIABLE	DESCRIPTION
descrip_mentions_pt	description mentions public transport
monument_distances	distance to Paris most visited monuments
distance_to_center	/
subway_distances	distance to the nearest subway station

- Monuments taken: Eiffel Tower, Louvre Musuem, Notre Dame, Arc de Triomphe, Sacré Coeur
- `lat_center = 48.856600, lon_center = 2.352200`

DATA MODELING

- Evaluation Metric

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2}$$

- Standard scaling of the data
  Caution: same scaler on train and test sets!

$$x' = \frac{x - \mu}{\sigma} \Rightarrow \mathbb{E}(X') = 0 \text{ and } \mathbb{V}(X') = 1$$

1ST APPROACH

XGBoost

THE MODEL

- Decision-**tree** based
- Uses **bagging**: combining predictions from several trees
- Uses **boosting**: we build trees sequentially
  - by minimizing the errors from previous models
  - by increasing influence of high performing models
- Uses **gradient descent** technique to minimize the expected loss

$$\hat{F} = \underset{F}{\text{argmin}}\ \mathbb{E}\left(L(y, F(x))\right)$$

- Also uses: parallel processing, tree-pruning, handles missing values, handles regularization, and more…

PARAMETERS

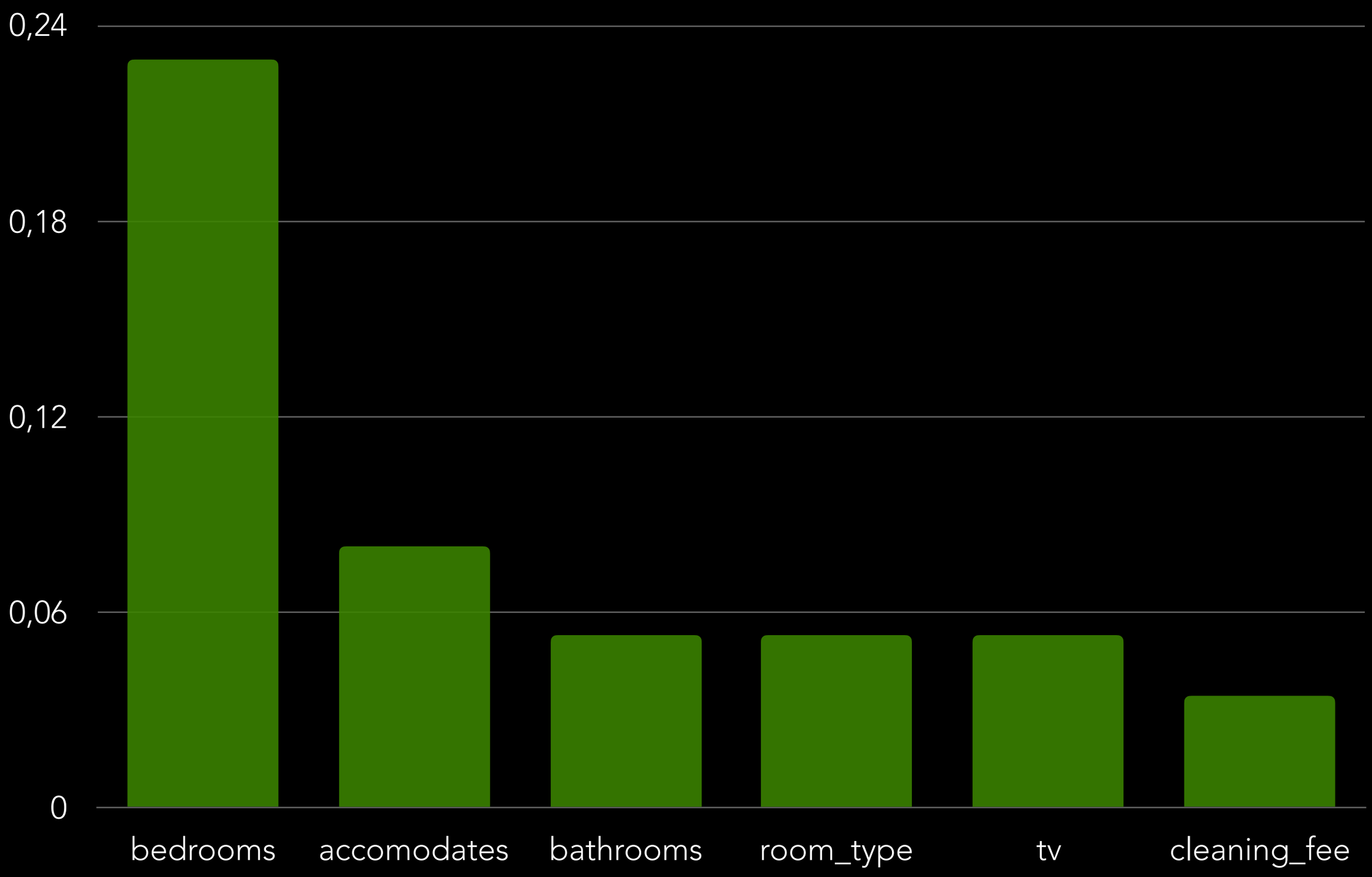
- `n_estimators`: number of trees to fit
- `max_depth`: maximum depth of a tree
  => Useful to control over-fitting
- `learning_rate`: how much do we retain from weights of previous step at every new step
- `min_child_weight`: min sum of weights required in a child
  => Useful to control over-fitting
- `subsample`: fraction of obs to be randomly sampled for each tree
  => Useful to control over-fitting.
- `colsample_bytree:` fraction of cols to be randomly sampled for each tree.

RESULTS

STAGE	RMSE	Observations
Before parameter tuning	train_RMSE=44.91 validation_RMSE=44.95	Nearly no overfitting
After parameter tuning	train_RMSE=32.94 validation_RMSE=42.47	We overfit much more but test_RMSE is much better!

- Question arises
  Should we privilege a model that has a worse **`test_RMSE`** but does not overfit over one that is more capable to generalize but that gives less performance on the **`test`** set?

FEATURE IMPORTANCE



FIRST COMMENTS

We didn’t talk about how we encoded qualitative variables

- There are **many ways to encode**, adapted to different kinds of issues such as high cardinality, ordered or nominal, etc.

- In our case we don't have any variable with a lot of modalities (more than 1000) so we **don't need to worry about high cardinality issues.**

- But we have some **ordered categorical variables**

- Many articles in the literature have shown that the type of **encoding that we use should depend on the type of model** that we intend to implement:
  - we ordinally encoded multinomial hierarchical categorical features
  - we one-hot encoded simple ones (0 or 1)

- We will implement **CatBoost** for a new type of encoding

FIRST COMMENTS

- We didn’t do any feature selection because tree-based models are not very much affected by multicollinearity

2ND APPROACH

CatBoost

THE MODEL

- Was launched in 2017 (**later than XGBoost**) by the Russian tech company Yandex

- Similar parameters as XGBoost

- **Treats qualitative variables differently** + combines them => more efficient in some situations

- We don’t need to scale any feature

THE MODEL

- one_hot_max_size encoding

feature 1	y
A	35
B	40
A	21
C	75
A	83.2
C	50
A	51

feature 1	y
A	35
A	40
B	21
C	75
A	42.75
C	
A	

- categorical features combination

RESULTS

for learning_rate=0.02

depth	l2_leaf_reg	one_hot_max_size	rsm	RMSE
6	3	2	1	train_RMSE=39.5588 validation_RMSE=42.4739
5	3	2	1	train_RMSE=40.6128 validation_RMSE=42.6497
7	3	2	1	train_RMSE=38.2708 validation_RMSE=42.3555
8	3	2	1	train_RMSE=38.2653 validation_RMSE=42.4516
9	3	2	1	train_RMSE=34.9649 validation_RMSE=42.4501
7	3	4	1	train_RMSE=37.4872 validation_RMSE=42.1436
7	3	6	1	train_RMSE=37.2428 validation_RMSE=42.0587

RESULTS

- We also tuned `l2_leaf_reg` and `rsm` but performance decreased

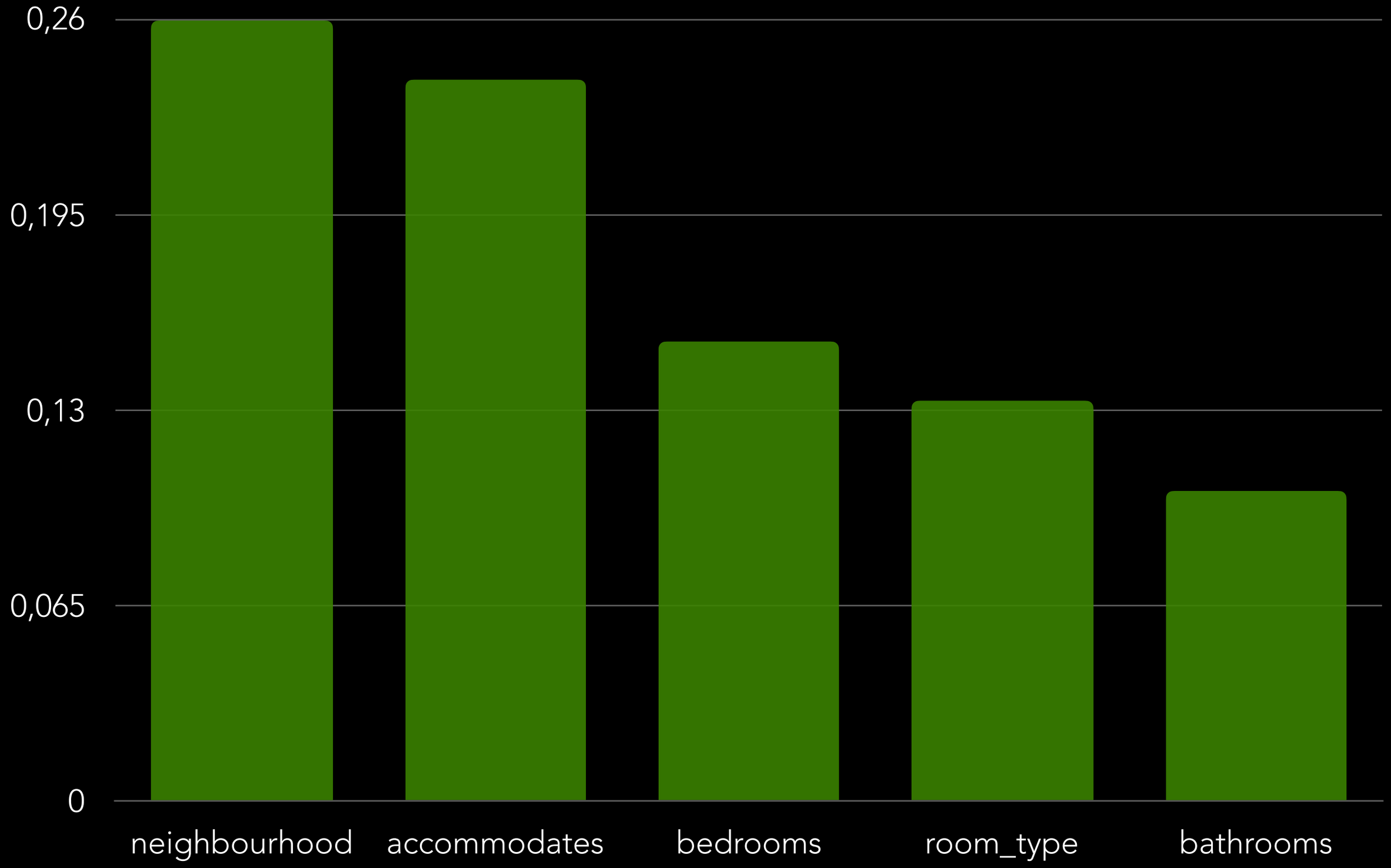
- Here we overfit more than with XGBoost…

=> but given that **in the specific case of this competition** the test set will be a subset of the data on which we trained, it's not a big issue!

=> we increase n_estimators (e.g. 2000)

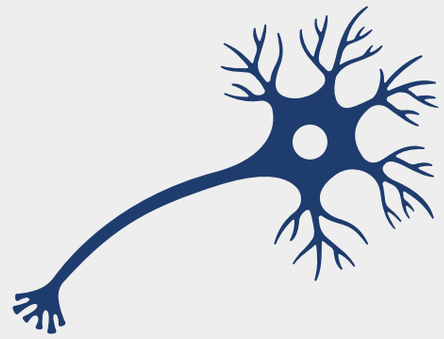
Best submission
train_RMSE=19.5855
validation_RMSE=41.8872

FEATURE IMPORTANCE



3RD APPROACH

Neural Networks



THE MODEL

- Sequential model with Dense Layer

- Creation:
  1 Input Layer
  m Hidden layers
  1 Output Layer with only 1 neuron
  +
  Activation functions

- Loss: mean squared error

- Different regularization techniques

RESULTS

- 4 layers (128, 256, 256, 1) / no regularization / no batchnormalization / default adam / batch_size=256 => (train_RMSE=0.2203, validation_RMSE=0.3672)
- 4 layers (128, 256, 256, 1) / dropout=0.2 / no batchnormalization / default adam / batch_size=256 => (train_RMSE=0.2933, validation_RMSE=0.3618)
- 4 layers (128, 256, 256, 1) / dropout=0.2 / batchnormalization / default adam / batch_size=256 => (train_RMSE=0.2795, validation_RMSE=0.3266)
- 4 layers (128, 256, 256, 1) / dropout=0.3 / batchnormalization / default adam / batch_size=256 => (train_RMSE=0.2762, validation_RMSE=0.3177)
- 4 layers (128, 256, 256, 1) / dropout=0.4 / batchnormalization / default adam / batch_size=256 => (train_RMSE=0.2686, validation_RMSE=0.314)
- **4 layers (128, 256, 256, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.2716, validation_RMSE=0.3123)**
- 4 layers (128, 256, 256, 1) / dropout=0.45 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.2815, validation_RMSE=0.3154)
- 4 layers (128, 256, 256, 1) / l1=0.005 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.3167, validation_RMSE=0.3231)
- 4 layers (128, 256, 256, 1) / l2=0.01 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.6249, validation_RMSE=0.6274)
- 4 layers (128, 256, 256, 1) / l2=0.02 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.3296, validation_RMSE=0.3391)
- 4 layers (128, 256, 256, 1) / l2=0.03 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.462, validation_RMSE=0.4765)
- 5 layers (128, 256, 256, 256, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.2795, validation_RMSE=0.3182)
- 5 layers (128, 256, 256, 512, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=256 => (train_RMSE=0.2918, validation_RMSE=0.3253)
- 4 layers (128, 256, 256, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=128 => (train_RMSE=0.265, validation_RMSE=0.3135)
- 4 layers (128, 256, 256, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=512 => (train_RMSE=0.3047, validation_RMSE=0.3227)
- 4 layers (64, 128, 256, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=512 => (train_RMSE=0.3047, validation_RMSE=0.3227)
- 3 layers (128, 256, 1) / dropout=0.4 / batchnormalization / custom adam / batch_size=512 => (train_RMSE=0.2885, validation_RMSE=0.3214)

CONCLUSION

- When it comes to unstructured data, NN are generally better, but here it's not the case and XGBoost outperforms
- CatBoost is better than XGBoost because of the way it treats categorical variables

BIBLIOGRAPHY

MISSING VALUES

- https://towardsdatascience.com/handling-missing-values-in-machine-learning-part-1-dda69d4f88ca
- https://towardsdatascience.com/data-cleaning-with-python-and-pandas-detecting-missing-values-3e9c6ebcf78b

CATEGORICAL FEATURES

- https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d
- https://medium.com/data-design/visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931
- http://contrib.scikit-learn.org/categorical-encoding/basen.html
- https://towardsdatascience.com/smarter-ways-to-encode-categorical-data-for-machine-learning-part-1-of-3-6dca2f71b159

XGBOOST TUNNING

- https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

BIBLIOGRAPHY

FEATURE SELECTION

- https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html

SIMILAR PROJECT

- https://towardsdatascience.com/predicting-airbnb-prices-with-deep-learning-part-1-how-to-clean-up-airbnb-data-a5d58e299f6c
- https://towardsdatascience.com/predicting-airbnb-prices-with-machine-learning-and-deep-learning-f46d44afb8a6
- https://github.com/L-Lewis/Airbnb-neural-network-price-prediction