trivago Case Study
"Predict a click" Challenge

Carlo Contaldi

contaldicarlo@gmail.com

www.linkedin.com/in/carlocontaldi/

github.com/carlocontaldi/trivago_case_study

---

1. Introduction

This report presents my case study for trivago's "Predict a click" challenge. Such study constitutes my submission to trivago for a step of my recruitment process for the position "Junior Data Scientist – Market Dynamics" (Düsseldorf, Germany).

1.1. Background

trivago is a German transnational technology company specializing in internet-related services and products in the hotel, lodging and meta search fields. Its mission is to be the traveler's first and independent source of information for finding the ideal hotel at the lowest rate.

Marketplace at trivago is a cross-functional team which delivers data-driven decision making solutions addressing the connection between customer demand and advertiser supply. The goal is to provide the right accommodations for users and help advertisers to run well-performing campaigns on the trivago website.

The balance between the interests of users and advertisers is established by the Exposure Algorithms, which determine what hotels and advertisers trivago presents to the user and what position they will be shown at. One important objective of the algorithm is to anticipate how likely users are to click on a specific hotel.

1.2. Specification & Preliminary Analysis

In this section all the relevant information included in the challenge description is reported and analyzed. All given assumptions will be evaluated in my workflow.

1.2.1. Statement of Work

The goal of this challenge is "to predict how often a hotel will be clicked based on certain characteristics of the hotel".

The deliverable for this challenge is given by:

- submission file with predictions;

- modelling code;
- explanations on the thought process behind each step.

Despite bonus questions included in the challenge are optional, I will attempt to integrate them in my data science workflow because they might support it.

The predictions are evaluated based on a given control metric, i.e. a "normalized weighted mean square error" defined as in the following:

$$wmse := \frac{1}{n}\frac{\sum_{i=0}^{n} w_i \cdot (predictedClicks_i - observedClicks_i)^2}{\sum_{i=0}^{n} w_i} \tag{1}$$

$$w_i := log(observedClicks_i + 1) + 1 \tag{2}$$

This metric defines a Mean Square Error where the error of each prediction is weighted by a quantity which is proportional to the logarithm of the true target value; this loss score can be read as the mean of all weighted square errors, with the weights summing up to 1. This custom MSE is emphasized with hotel instances having logarithmically large numbers of clicks. Given that it is a composition of differentiable functions, it is differentiable as well.

Such metric defines an error: it should be minimized in order to maximize the performance.

The task is formulated in an open and simple way: it is unconstrained by a minimum target error over the given control metric or any other satisficing metric. I consequently assume that my Statement of Work consists of:

1. identifying and leveraging as many as possible data patterns;
2. modelling and evaluating a representative set of pipelines with the purpose of performance maximization over the given control metric.

1.2.2. Data

I am provided with 3 datasets encoded in .csv format. Each entry in the data contains information about a specific hotel with respect to a specific time frame, and is uniquely identified by a `hotel_id`.

- `train_set.csv` – training dataset. Each entry is characterized by a set of features representing hotel properties, as well as the target variable `n_clicks`.
- `test_set.csv` – test dataset. Each entry contains the same information included in training entries, excepting for the target variable.
- `sample_submission.csv` – example submission file. Each entry is associated with each entry in the test set via the `hotel_id`, and contains the target value prediction.

Below is reported the given description of all data columns:

- `hotel_id`: unique hotel ID.
- `city_id`: ID of city which the hotel is located in.

- `content_score`: score describing the quality of the content provided for the hotel; expected variable domain: number in $[0, 100]$.
- `n_images`: number of images that are available for the hotel; expected variable domain: non-negative integer number.
- `distance_to_center`: distance in meters of the hotel to the nearest city center; expected variable domain: non-negative number.
- `avg_rating`: average rating of the hotel; expected variable domain: number in $[0, 100]$.
- `n_reviews`: number of reviews available for the hotel; expected variable domain: non-negative integer number.
- `avg_rank`: average position the hotel had in the list; expected variable domain: number $n \geq 0$ if 0-indexed or $n \geq 1$ if 1-indexed.
- `avg_price`: average price in Euro of the hotel; expected variable domain: non-negative number (I assume the possibility of a 0 value, which makes sense for instance with a newly opened hotel which launches a free rooms promotion).
- `avg_saving_percent`: "average saving users achieve on the hotel by using trivago", i.e. the relative difference between the cheapest and most expensive deal for the hotel; given that the "relative difference" between two quantities can be defined in several ways, further analysis is needed to better qualify this data source.
- `n_clicks`: the number of clicks the hotel has received in a specific time frame; expected variable domain: non-negative integer number.

Each entry should be characterized by a unique `hotel_id` over both training and test sets.

The training set is not clean and may contain entries with missing or nonsensical values, as well as hashed and/or modified values. I assume that the above-mentioned aspects potentially characterize the test set as well.

2. Frame the Problem

Based on the given specification, the task can be formulated as a regression problem in the domain of the target variable, i.e. the set of non-negative integers. I assumed that the control metric defined in Equation 1 is reliable enough for the case at hand. Given that it is a differentiable function, it can be used as a custom loss function for models which need it. Further analyses about evaluation metrics are reported in Section 2.1.

I have been provided with labeled training data and unlabeled test data. I could leverage both training and test data during exploratory data analysis so as to assess data representativeness or sampling bias over a larger population sample, but I can't ignore the possibility that I may better demonstrate my data science skills by choosing to not leverage test data.

It may be that the test dataset has been designed to represent enough the phenomenon and patterns underlying the training data so as to get significant performance by using the training data only, as well as to embed patterns which can be captured only by means of a model having relevant generalization capability over the training data. Thus, I assumed that the test set is "not available at production time", i.e. that it can't be leveraged during the execution of my workflow. I'm aware that by doing so I have higher chances of making bad modeling assumptions with respect to the test data; on the other hand, such assumption promotes a more realistic case study and gives me the possibility to better demonstrate my skills (if we assume the truth of my hypotheses about test dataset design).

I ensured a fair generalization capability in my modelling pipelines so as to address potential issues in new data (such as assuming that each field may have missing values). With regard to new data, I assumed that it is characterized by the same features in terms of names, formats and the processes which generated them. I consequently leveraged the `test` set just to ensure:

- its compatibility with the `train` set;
- `train` and `test` set disjointness with respect to `hotel_id`, as assumed in the challenge specification.

The best model resulted from my pipeline has been trained with `train` data and then applied on the `test` set. Resulting predictions have been included in the attached submission file.

I noticed that the challenge specification does not rule out the usage of external data.

2.1. Bonus Question #1

Can you describe in your own words what the purpose of the evaluation metric above is? What alternative metrics would make sense in this context?

In general, an evaluation metric is needed to quantitatively evaluate and compare estimators performance, as well as to drive other procedures of the data science workflow, e.g., model tuning. A proper metric should be able to convert a relevant set of criteria describing the business case performance into a quantitative value.

As said in Section 1.2.1, this metric defines a Mean Square Error where the error of each prediction is weighted by a quantity which is proportional to the logarithm of the true target value. As we take the square of the error, the effect of larger errors become more pronounced than smaller error, hence the model can focus more on the larger errors. At the same time, predictions are weighted by a normalized quantity proportional to the logarithm of the actual `n_clicks`. Basically, what is needed to achieve here is an estimator especially good in predicting high-valued regression instances, with the first criterion being the maximum possible precision.

I would always recommend the usage of a single control metric, so as to deal with a straightforward and unambiguous evaluation and comparison.

3. Programming Environment and Execution Procedure

I used Conda 4.5.11 with Python 3.6.5 (latest versions to date) on a machine implementing Windows 10 64-bit to set up my PyData stack. I worked within a new environment defined and activated via terminal as in the following:

```
> conda create --name trivago python=3.6.5 numpy pandas matplotlib seaborn
  scikit-learn py-xgboost
> conda activate trivago
```

The `main.py` script executes all the code I wrote and saves the output in a logfile.

```
> cd project_folder
> python main.py
```

The attached code is commented in some sections, especially those which are computationally expensive. However, completing its execution provides the user with the required submission file with predictions.

4. Data Science Workflow: Exploration Phase

A generic data science life cycle comprises hypothesis formulation, data retrieval, cleaning, exploration and visualization, feature engineering, modeling, evaluation. I generally recommend to approach the cycle with a Lean Startup spirit: the sooner we iterate, the sooner we reach our goal.

Any applied algorithm characterized by random components has been executed with a predefined `random_state=0`, so as to ensure experiment repeatability. In general it would be better to assess any non-deterministic action over a repeated number of trials so as to ensure a fair amount of repeatability while also prevent inferences driven and biased by randomness. On the other hand, given the purpose of this project, I prefer to possibly sacrifice fairness in favor of a lower need of time and resources.

4.1. Data Retrieval

I imported each .csv file as a Pandas DataFrame, and checked the import for correctness with `head()`, as coded in `import_data.py`.

4.2. Exploratory Data Analysis

All the work of this step is coded in `eda.py` and `visualize.py`.

`train_set` and `test_set`, from now on defined as the `train` and `test` sets, respectively contain approximately 400 thousand and 132 thousand entries (specifically 396487 and 132162 rows, as also indicated in the challenge description).

By observing `train` and `test` columns, I assessed that `test` has the same column names of `train` excepting for `n_clicks`. Moreover, I noticed the presence of an additional feature not described in the challenge specification, named `stars`. Such column has been taken into account in my workflow despite it was not reported in the challenge description.

All column types in the three sets are numerical, as expected: I converted all of them into the smallest float type possible (which resulted to be `float32`) so as to ensure overall

compatibility and optimize data handling. The applied procedure also addresses conversion issues in new data: any non-numerical value is converted into a NaN.

As part of feature analysis, I decided to consider an in-depth evaluation of residuals independently for each numerical feature distribution, since a similar approach may help in identifying extremely skewed distributions and properly addressing outliers involving the joint action of all features. I define an "extreme outlier" as a data entry having extreme values for at least one of its numerical features, i.e. values which would impact in a bad way the rescaling process needed for most of regression models; its presence may wash out values assumed by the inlier population. A "strong outlier" can be defined as a entry which is tagged as an outlier by outlier detectors, based on the joint totality of its features; I would generally recommend to not drop related entries and to tag them as outliers, since identifying them provides our model with an additional useful piece of information. The main purpose of this feature-independent residuals evaluation is to uncover patterns associated with strong outliers or, even better, extreme outliers.

Each feature is analyzed in-depth below, then a global analysis of patterns over missing data and an high-level excursus on outliers in this business case are reported; finally, an in-depth univariate and bivariate feature analysis concludes this section.

4.2.1. Feature Analysis: hotel_id

This column expectedly constitutes the unique identifier of an hotel entry. Given that it can be consequently defined as the primary key of the sets, any entry with a null `hotel_id` should be dropped. No entry in the given data has null `hotel_id`.

With regard to variable domain, I checked that all IDs in the sets are whole numbers.

ID uniqueness in train and test sets has been assessed: all values are unique. Possible duplicate entries with respect to this field should be dropped.

train and test sets resulted to be disjoint with respect to this field. Allowing redundant entries across training and test sets would entail making predictions over known and/or conflicting data: this is one of the reasons why they should be dropped.

As a further check over the expected submission file content, it has been verified whether test and sample sets involve the same entries.

Since this feature does not have any informative power for the task at hand nor is needed for feature engineering purposes, it has been dropped.

4.2.2. Feature Analysis: city_id

This feature describes the ID of the city which the hotel is located in.

I observed that 508 train entries (around 1/1000 of the total) are characterized by missing values for this field. By visually inspecting a subset of them, I noticed the presence of a potential pattern: all entries having a missing `city_id` have also missing `content_score`, `n_images`, `distance_to_center`, `avg_rating`, `stars` and `n_reviews`; the presence of this pattern has then been confirmed. Such consistent and defined pattern makes me wonder on the process which generated data, in particular the causes over multiple missing value assignment. Data characterized by such property may embody exclusive information about the underlying phenomenon, so it would be reasonable to investigate about it, possibly

impute missing values and leverage related entries. On the other hand, given the relatively low number of entries characterized by this pattern and that it is still the first iteration of my workflow, I decided to drop related entries and possibly address the problem later.

With regard to variable domain, I verified that all city IDs in the train set are whole numbers.

This feature represents a categorical variable: analyzing the distributions of its values in the population may help in better understanding available data as well as assessing its representativeness. I computed general statistics over value counts of this feature, and observed that:

- 33213 distinct city IDs are present in the training set, i.e. there are approximately 12 hotels per city on average;
- minimum and maximum frequencies are respectively 1 and 2489;
- the median frequency is 2;
- the frequency at the third quartile is 7.

Given the resulting heterogeneous and variate distribution, I furthered this analysis by plotting frequency twentiles (which helps to evaluate data skewness as well). I observed that half of training entries involve cities with 1 or 2 hotels only, 45% of the entries belong to cities having between 3 and 50 hotels, the remaining 5% of the entries involve cities having more than 50 hotels.

This feature presents a reasonable distribution and probably has predictive power for the task at hand, just based on the hypothesis that the hotel city impacts on customer demand and interest. In order to leverage it, we need to take into account its large number of categorical values (expected to be around 1/12 of data size), and that many of them occur too rarely to be relevant for training. We also need to consider that new data may include a consistent set of new values. I planned the application and evaluation of two preprocessing strategies later in my workflow so as to effectively leverage this feature, even with new data:

- apply one-hot encoding followed by a dimensionality reduction technique over all features, so as to automatically rule out poorly informative features;
- redefine feature values by means of a clusterization technique before one-hot encoding.

4.2.3. Feature Analysis: content_score

This feature represents a score which describes the quality of the content provided for the hotel.

I assessed that there are no missing values for this variable, excepting for those of the entries dropped during `city_id` feature analysis, as explained in Section 4.2.2.

With regard to variable domain, I verified that all scores in the training set are whole numbers in the range $[0, 100]$. Based on this assumption, this variable can't be affected by extreme outliers and will be rescaled in a predefined way by dividing it by 100 during the

preprocessing step: such rescaling approach is independent of data and would always result in a correct handling of new data.

By evaluating feature distribution with Kernel Density Estimation, it can be observed that most of scores are in the range [40,65] (87% of train entries), with two peaks around 42 and 58; another relevant portion of data peaks at 20 and collects approximately 6% of entries in its surroundings. It can be concluded that this feature presents a reasonable distribution.

4.2.4. Feature Analysis: n_images

This feature represents the number of images that are available for the hotel.

Besides the entries dropped during `city_id` feature analysis (as explained in Section 4.2.2), I assessed the presence of a single entry having a missing value over this feature, and dropped it without further consideration.

Below are reported the observations I made by observing general statistics, value counts and twentiles of this series.

- Negative values are included in the series; specifically, the value -1 occurs approximately in 1/120 of data (3361 entries). This anomaly may be due to many reasons, such as the fact that it indicates a special value with a special meaning for this feature, or a systematic error due to some bias source in the data collection process.

- Extreme outliers are present in this series, given that the median and the maximum values–respectively 1 and approximately $2.4 * 10^6$–are characterized by consistently different orders of magnitude. This hypothesis has been confirmed by the fact that the minimum value and all twentiles are in the same order of magnitude: the series includes a relatively small set (less than 5% of data) of high values. These values are significantly high, if we consider that even sample median and mean (respectively 1 and 259) are characterized by different orders of magnitude.

- The absence of images is the most frequent event: it occurs in almost half of data (approximately $1.9 * 10^5$) entries).

With regard to negative values, I planned to consider a feature engineering step which defines a further boolean feature indicating whether original values were negative or not.

Let us consider the task of building a detector of strong outliers for this feature. By observing KDE representation of this series, it can be concluded that an acceptable detector should tag as outliers all values lower than -1 or at least higher than 10. Upper limits larger than 10 would be more conservative and thus better, if they are approximately characterized by the same magnitude.

In order to conservatively qualify a larger limit and minimize the possibility of considering inliers as outliers, I also leveraged a box representation of the series. The box plot is generally used to graphically depict groups of numerical data through their quartiles, as well as to visualize the action of an outlier detector implementing the interquartile range technique. As a first observation, the inlier frontier yielded by such method is $[-1, 6]$: the upper limit of this range can be considered a safe minimum value for the discriminating threshold. By

observing the global box plot, it can be observed an irregular outlier distribution, excepting for the surroundings of the box, where a denser (supposedly outlier) cluster is depicted.

So as to possibly identify a larger upper limit for our frontier, let us now assume that the baseline range is too restricted: in particular, this would mean that a significantly sized sample subset having a complementing distribution with inlier data is erroneously tagged as outlier data; in other words, the hypothesis to prove here is that the supposedly too limited frontier significantly cuts out the inlier distribution tail, and erroneously promotes it as outlier data. By zooming the box plot in the $[0, 200]$ feature range, it is possible to identify a well defined data cluster up to 75 (depicted as outlier data), which appears to decrease in density as the distance from the upper whisker increases. This data cluster resembles the above-mentioned distribution tail, cut out by our too strict outlier detector, and the relatively large gap between the last point of the cluster (75) and the next data point (around 190) endorses such hypothesis. Moreover, given that the cluster has a significant relative size with respect to training data size (around 4.5%), it would be even less reasonable to consider it as part of outlier data. On the basis of previous reasonings, it can be concluded that $[-1, 75]$ can be assumed as a target inlier frontier.

An optimal outlier detector in this case would be able to infer the above-mentioned strategy without any tuning procedure. A set of relevant outlier detectors have been evaluated.

Outlier detectors based on Z-Score depend on the sample mean. Given that the sample mean of this series is not representative of inlier data, such technique is not appropriate for this case.

A more robust measure of "central tendency" is the median as the presence of extreme values have a reduced effect on calculation of the median compared with the mean. The median value is used in the Modified Z-Score outlier detection method (Boris Iglewicz and David Hoaglin (1993), 'Volume 16: How to Detect and Handle Outliers', The ASQC Basic References in Quality Control: Statistical Techniques). Its default implementation yielded $[-1, 249]$ as inlier frontier.

The Elliptic Envelope is another technique able to detect outliers in a Gaussian distributed dataset, based on a robust covariance estimate to the data. It identified outliers outside of the range $[-1, 4]$. It should be pointed out the presence of a `contamination` parameter, which indicates the outlier proportion and which is set by default to 0.1.

A drawback of Z-Score-based and Elliptic Envelope methods is that they assume that inlier data is Gaussian distributed, which is not exactly our case (in particular, from the density graph I can infer a half-normal shape existing over non-negative values only). The Isolation Forest is a tree-based outlier detector which does not make any assumption on the shape of the distribution. Its default implementation yielded $[0, 4]$ as outlier frontier. Even this model involves a `contamination` parameter, once again set to 0.1 by default.

An alternative `sklearn` implementation of the Isolation Forest involves an approach which does not depend on a given `contamination` ratio: given its appealing insensitiveness to this parameter, this version has been evaluated as well. It identified outliers outside of the range $[0, 3]$.

I did not take into account DBSCAN or similarly expensive approaches due to high space and time complexity.

The Z-Score detector provided us with the most conservative inlier frontier, in accordance with our visually inferred target outlier frontier.

It can be concluded that, by excluding outliers, this feature presents a reasonable distribution.

4.2.5. Feature Analysis: distance_to_center

This feature represents the distance between the hotel and the nearest city center, in meters.

I assessed the presence of 20 entries with missing values for this variable, besides those dropped during `city_id` feature analysis, as explained in Section 4.2.2. I also noticed that such entries are the only ones in which also `n_reviews` values are missing. Given the relatively low number of entries with missing values for these variables, I decided to drop them.

With regard to the variable domain, I verified that all values in the training set are whole non-negative numbers.

From the KDE density graph I can infer a half-normal shape existing over non-negative values only.

By observing general statistics and twentiles of this series, I noticed that extreme outliers are present in this series, given that the median and the maximum values–respectively $1.5 * 10^3$ and $1.8 * 10^7$ (approximated)–are characterized by consistently different orders of magnitude. This hypothesis has been confirmed by the fact that the minimum value and all twentiles present a regular, polynomial trend, whereas a relatively small set of entries (less than 5% of data) involves relatively very high values which detach from such trend. These values are significantly high, if we consider that even sample median and mean (respectively $1.5 * 10^3$ and $1.6 * 10^4$) are characterized by different orders of magnitude.

By observing KDE representation of this series, it can be concluded that an acceptable outlier detector for the task at hand should tag as outliers all values lower than 0 or at least higher than 10^4. Upper limits larger than 10^4 would be more conservative and thus better, if they are approximately characterized by the same magnitude.

The baseline outlier detection based on the interquartile range approach identifies the range $[0, 9.6 * 10^3]$ as inlier frontier. Based on the box plots of this feature, a more regular distribution with respect to the `n_images` case can be observed. In particular, in this case it is harder to identify a discriminating outlier threshold, because of the relatively long and homogeneous distribution involving inlier tail and outliers. This aspect is more emphasized in the second plot, where a zoom in the $[0, 10^5]$ range has been applied. As a consequence, no better estimate for the upper limit of our frontier can be inferred by visual inspection this time.

The baseline detector based on Z-Score depends on the sample mean. Given that the sample mean of this series is not representative of inlier data, such technique is not appropriate for this case.

The default implementation of the Modified Z-Score strategy yielded approximately $[0, 1.4 * 10^5]$ as inlier frontier: since one order of magnitude definitely separates the upper limit of this frontier from the hypothesized expected value, I excluded this method from

further consideration for this feature.

The Elliptic Envelope technique identified outliers outside of the range $[0, 9848]$ in its default implementation.

The default implementation of the Isolation Forest and its new version respectively yielded $[0, 9958]$ and $[0, 11736]$ as outlier frontier.

I finally chose the new version of the Isolation Forest detector for use in my pipeline for this feature, because it yields the largest conservative inlier frontier without even depending on the `contamination` parameter.

It can be concluded that, by excluding outliers, this feature presents a reasonable distribution.

4.2.6. Feature Analysis: avg_rating

This feature indicates the average rating of the hotel.

I assessed the presence of approximately $1.1 * 10^5$ entries with missing values for this variable, besides those dropped during `city_id` feature analysis, as explained in Section 4.2.2. Given that they constitute more than one quarter of training data, an imputation strategy is needed. I also planned to tag all imputed values by providing the dataframe with an additional boolean column.

With regard to the variable domain, I verified that all non-missing values in the training set are whole numbers comprised in the range $[0, 100]$, as expected. Based on this assumption, this variable can't be affected by extreme outliers and will be rescaled in a predefined way by dividing it by 100 during the preprocessing step: such rescaling approach is independent of data and would always result in a correct handling of new data.

By comparing statistics summaries of the data subsets respectively with assigned `avg_rating` and missing `avg_rating`, I noticed that:

- all entries with missing values have `n_reviews` $= 0$, which happens in no entry in the complementing subset;
- entries with missing values have a relatively lower `n_images` and `stars` overall, as also shown in the twentile comparison plots.

Based on the perfect correlation indicated at the first bullet, I am confident to assume that missing entries for this field are due to the fact that no customer reviewed related hotels. Such fact would also involve the Cold Start problem: what rating should be assigned to a new hotel?

I plotted twentiles computed on the set of non-missing values, and observed that 90% of data presents an `avg_rating` in the range $[70, 90]$. Besides being a relatively restricted range with respect to the $[0, 100]$ domain, it also shows a considerable systematic bias with respect to the mid-range value (50). Investigating about the reasons of such bias may provide us with further insights about this variable. It can be concluded that, besides the unexpectedly restricted and biased distribution range, this feature is reasonably distributed within such range.

I planned to address the imputation problem with a simple, baseline approach in the first place, given by replacing NaNs with a single, reasonable value. Given that this transformation would probably introduce modelling bias, I chose the imputation strategy able to desirably reduce it as much as possible, i.e. the strategy which has the minimum impact on the distribution. As a consequence, I did not consider static values such as 0 or 50 (mid-range value). Average and median values of the series are more suitable to our case; in particular, given the absence of high skewness (induced by the limited domain), if the variable is normally distributed then the mean would be a good fit. The KDE graph computed over non missing `avg_rating` distribution can be approximated to a normal distribution with a spike on 80. Still, considering that such spike is relatively close to the distribution peak (around 85), its impact on the final result would probably be negligible.

I consequently chose the mean as imputation strategy in this case. So as to adapt the imputed value to the variable domain, it has been rounded. The (bad) impact of such simple imputation strategy can be observed in the KDE representation of the variable over all data after imputation: I would indeed recommend the application of more robust imputation techniques, such as regression imputation. I planned to address the problem at a later iteration of my workflow.

4.2.7. Feature Analysis: stars

This feature is the only one which was not introduced in the challenge description; its name suggests that it indicates the hotel star rating or a more generic star-based score.

I assessed the presence of 33 entries with missing values for this variable, besides those dropped during `city_id` feature analysis, as explained in Section 4.2.2.

Based on general statistics of this variable, it can be observed that its training series exists on the $[0, 5]$ range; I also assessed that such series is constituted of whole numbers only. These considerations endorse that such feature represents some sort of rating.

Let us assume that it represents the hotel stars rating. If this is true, there should be a relevant correlation between this variable and the `avg_price`, i.e. the most likely feature among the available ones to be correlated with the hotel stars rating, based on common sense. In order to test such assumption, I planned to leverage violin plots, i.e. a combination of KDE and box representations: it shows the distribution of quantitative data across several levels of a set of categorical variables. By using this representation on the ordered set of values assumed by `stars`, it is possible to evaluate trend correlations between our control feature and the distributions of other features grouped by the values the control feature assumes.

As a preliminary check, I evaluated whether data subsets grouped by `stars` contain enough samples to be representative of the underlying populations. It resulted that the 1-star subset is the smallest one, containing approximately 1% of data: this sample set is relatively small, but it has enough data points–4642–to assume that the resulting distribution can be approximated to the true one at an acceptable degree for our task. I also noticed that the 0-star subset includes 63% of data.

I consequently plotted a set of violin plots for all numerical features in the dataset, indexed by the 6 possible `stars` values. Below are reported my observations.

- The ordered $[1, 5]$ `stars` range exhibits a clear correlation with other features; in particular `content_score`, `avg_rating`, `avg_price` and `avg_saving_percent` increases with the `stars` value, whereas `avg_rank` decreases as the `stars` value increases.
- The 0-star subset looks like a group on its own as compared to the other groups for many aspects:
  - it has comparatively lower aggregate values in terms of `content_score` and `n_reviews`;
  - the distribution of its `content_score` and `avg_rating` has a different shape with respect to the other groups;
  - as opposed to other groups, most of its entries have a 0 value for `n_images` and `avg_saving_percent`.

  On the other hand, this group seems to complete the correlation sequence for what concerns `avg_rank`.

What I can derive from such observations is first of all an endorsement to my initial hypothesis, i.e. that `stars` represents the hotel stars rating; I am confident enough about the truth of this assumption especially thanks to the clear correlation between this variable and `avg_price`.

I could also hypothesize that the 0 value actually indicates a missing value for this feature based on the following reasonings:

- 0-star hotels do not exist, or at least do not constitute more than half of the hotel population;
- the absence of stars may also indicate any structure which is generally less characterized by a typical star rating, such as hostels or aparthotels (serviced apartment complexes which use a hotel-style booking system).

I checked on `trivago.com` whether my assumption on hostels and aparthotels makes sense with respect to the way data is handled in the website. I assessed that it is possible to search over two distinct categories for what concerns the "Property type", which are "Hotel" and "House/Apartment", and that the star rating is missing for a relevant set of entries of the second category I visually inspected. This empirical evaluation further endorsed my hypotheses on the 0 value.

At this point I have been able to "connect the dots" between my convincing (but not proven) assumptions on the 0 value and the observations previously made on violin plots, but I was also aware that my consequent inferences (reported below) were colored by confirmation and orientation bias and that they should be taken with a grain of salt:

- `content_score`, `n_images` and `avg_rank` have relatively worse values for aparthotels because advertising and marketing efforts of hotels are generally more professional and expensive, which result in higher content scores.

- `avg_saving_percent` values equal to 0 entail the presence of a single room available, which is a frequent case for aparthotels, and the only possible case for single-apartment aparthotels; on the other hand hotels are more likely to have more rooms available at any time.

- Aparthotels present a comparatively lower `n_reviews` for the same reasons explained in the previous bullets.

Regardless of the true meaning of the 0 value, I finally decided to proceed by considering 0 as a special value, detached from the $[1, 5]$ ordered series and deserving a separate boolean column in the dataframe. The fact it does not fit in the sequence before 1 entails the need to replace it with a proper value in the $[1, 5]$ range: leaving it as it is would be misleading for any learning system which considers this feature as numerical, because it would comparatively consider 0-star hotels "worse" than the others.

In conclusion, I decided to treat 0 as a missing value. By following the same reasonings I provided in Section 4.2.6 for `avg_rating` missing values, based on the assessed normality of the distribution by visual inspection on the related KDE plot, I consequently chose a rounded mean imputation strategy for this feature.

Given the limited domain in which this variable exists, it will be rescaled in a predefined way by dividing it by 5 during the preprocessing step: such rescaling approach is independent of data and would always result in a correct handling of new data (unless trivago will not consider at some point a 6- or 7-star hotel category).

4.2.8. Feature Analysis: n_reviews

This variable represents the number of available hotel reviews.

I assessed the presence of 20 entries with missing values for this feature, besides those dropped during `city_id` feature analysis, as explained in Section 4.2.2. I also noticed that such entries are the only ones in which also `distance_to_center` values are missing. Given the relatively low number of entries with missing values for these variables, I decided to drop them.

With regard to the variable domain, I verified that all values this feature assumes in the training set are whole non-negative numbers.

From the KDE density graph I can observe a half-normal shape existing over non-negative values only.

Based on general statistics and twentiles of this series, I noticed that extreme outliers may be present in this series, given that the median and the maximum values–respectively 189 and approximately $2.8 * 10^5$–are characterized by consistently different orders of magnitude. This hypothesis has been confirmed by the fact that the minimum value and all twentiles present a regular, polynomial trend, whereas a relatively small set of entries (less than 5% of data) involves relatively very high values which detach from such trend. These values are significantly high, if we consider that even sample median and mean (respectively 189 and 914) are basically separated by an order of magnitude.

By observing twentile and KDE representation of this series, it can be concluded that an acceptable detector for the task at hand should tag as outliers all values lower than 0 or

at least higher than 4000. Upper limits larger than 4000 would be more conservative and thus better, if they are approximately characterized by the same magnitude.

The baseline outlier detection based on the interquartile range approach identifies the range $[0, 777]$ as inlier frontier. Based on the box plots of this feature, a more regular distribution with respect to the `n_images` case can be observed. In particular, in this case it is harder to identify a discriminating outlier threshold, because of the relatively long and homogeneous distribution involving inlier tail and outliers. This aspect is more emphasized in the second plot, where a zoom in the $[0, 10^4]$ range has been applied. As a consequence, no better estimate for the upper limit of our frontier can be inferred by visual inspection this time.

The baseline detector based on Z-Score depends on the sample mean. Given that the sample mean of this series is not representative of inlier data, such technique is not appropriate for this case.

The default implementation of the Modified Z-Score strategy yielded $[0, 6006]$ as inlier frontier.

The Elliptic Envelope technique identified outliers outside of the range $[0, 2448]$ in its default implementation.

The default implementation of the Isolation Forest and its new version respectively yielded $[0, 2622]$ and $[0, 1599]$ as outlier frontier.

Despite the new version of the Isolation Forest is less sensitive to parameter setting, it considers as outlier approximately 15% of the training data, which I consider too much with respect to my generic common sense threshold for outlier proportion (10%). I would choose the Modified Z-Score detector for use in my pipeline for this feature as it is, because it yields the largest conservative inlier frontier with a resulting outlier proportion of around 3%.

I also noticed that there is a value gap in available data: no sample assumes any value in $(0, 50)$ in terms of this feature.

It can be concluded that, by excluding outliers and the above-mentioned peculiarity, this feature presents a reasonable distribution.

4.2.9. Feature Analysis: avg_rank

This feature represents "the average position the hotel had in the list". I can assume that such list is the list of results the trivago search engine returns on a query.

It has been assessed the absence of missing values, as well as that all values this feature assumes in the training set are floating-point non-negative numbers. In particular, this feature exists within the range $[1, 100]$ in the training set. The variable domain complies with the expected meaning of the feature: it represents an average rank, apparently a 1-indexed one.

By observing the KDE plot of this feature over the training data, it seems that 1 is the true minimum of the overall population. I assessed this assumption by observing that approximately 1% of the data (3608 entries) is characterized by this extreme value–which seems a reasonable proportion for first-rank hotels.

Initially, I hypothesized that the maximum at 100 involves some implementation limit in the number of results returned by the trivago search engine. I empirically inspected on

the trivago website the number of entries returned by the engine and observed that at the time of writing the number of returned results for a generic query is 500. The fact that both 100 and 500 are nice round numbers (based on human common sense) endorses the implementation limit hypothesis. On the other hand, based on last percentiles and KDE plots, I observed that 99% of entries are in the range $[1, 25]$. I leveraged the box plot so as to better understand how extreme upper values distribute, and observed a very singular pattern: the $[77, 100]$ range contains data given by whole numbers only. I also observed that data is more or less homogeneously distributed in this range, and totals to around 100 samples. Regardless of the reasons of this particular trend, the related data subset is too negligible in size (around 1/4000 of training data) to be considered part of the inlier population a priori and without further information. I finally decided to proceed with outlier detection for this feature as well.

The baseline detector based on Z-Score and Modified Z-Score identified strong outliers respectively outside of ranges $[1, 34]$ and $[1, 37]$, with an outlier proportion approximately of 1/1000. They provide us with similar results because both mean (approximately 14.7) and median (15) are near, which makes both of them a representative measure of "central tendency" of this series. Based on previous considerations, I do not expect the possibility of extreme outliers in data.

All tested implementations of Elliptic Envelope and Isolation Forest yielded comparatively more restricted frontiers on both sides.

It can be concluded that this feature presents a bit uncommon but reasonable distribution.

4.2.10. Feature Analysis: avg_price

This feature represents the average price of the hotel, expressed in Euro.

I assessed the presence of 165 entries with missing values for this feature. I also noticed that such entries are the only ones in which also `avg_saving_percent` values are missing. Given the relatively low number of entries with missing values for these variables, I decided to drop them.

Based on the minimum value of this feature in the training set (4), I can assume that only positive prices make sense in this context, unless trivago does not allow the possibility of supplying free rooms.

Based on twentile and KDE plots, I can observe a gaussian distribution mostly spread between 20 and 300.

Based on the box plots of this feature, a more regular distribution similar to that of `distance_to_center` can be observed: a relatively long and homogeneous distribution involving inlier tail and outliers. This aspect is more emphasized in the second plot, where a zoom in the $[0, 2000]$ range has been applied. As a consequence, no better estimate for the upper limit of our frontier can be inferred by visual inspection this time.

The baseline detector based on Z-Score depends on the sample mean. Given that the sample mean of this series (around 109) seems to be skewed by upper outliers when compared with the median (77), such technique is not appropriate for this case.

The default implementation of the Modified Z-Score strategy yielded $[4, 427]$ as inlier frontier.

The Elliptic Envelope technique identified outliers outside of the range $[4, 199]$ in its default implementation.

The default implementation of the Isolation Forest and its new version respectively yielded $[16, 226]$ and $[19, 205]$ as outlier frontier.

It can be concluded that this feature presents a steep but reasonable distribution.

4.2.11. Feature Analysis: avg_saving_percent

As indicated in the challenge description, this feature represents the "average saving users achieve on the hotel by using trivago", which is defined as "the relative difference between the cheapest and most expensive deal for the hotel". As the name indicates, it seems to be expressed as a percentage.

I assessed the presence of 165 entries with missing values for this feature. I also noticed that such entries are the only ones in which also `avg_price` values are missing. Given the relatively low number of entries with missing values for these variables, I decided to drop them.

Based on general statistics computed over this feature values, I observed that it maxes at 99: given that this number is lower than 100 and that the feature does not assume negative values it results that the most probable definition of "the relative difference between the cheapest and most expensive deal for the hotel" corresponds to:

$$\frac{maxPrice - minPrice}{maxPrice} \tag{3}$$

where $maxPrice$ and $minPrice$ respectively represent the prices of the cheapest and most expensive deals of the hotel. As a consequence, it can be emphasized that:

- the absence of deals would entail a missing value for this feature (and this would explain why missing values in `avg_price` and `avg_saving_percent` characterize either none or both of these features on any entry in the training set);
- a 100% value for this feature is not possible unless trivago does not allow the possibility of supplying free rooms;
- if a hotel has a single deal available or the same price for all its deals, then the value of this feature would be 0.

I observed that the values this feature assumes are all whole numbers in the train set; this is surprising, given that this feature should represent an average value. I suppose that original values have been rounded or truncated during the data engineering process. As a consequence, by considering also that a 100 value is not possible based on previous considerations, I planned to rescale a priori this variable by dividing it by a predefined factor of 99.

By observing twentiles of this feature, it can be concluded that around 60% of data has a `avg_saving_percent` $= 0$. This may be due to my above-mentioned hypothesis on the 0

value or other reasons. As also mentioned in Section 4.2.7, the twentile plot shows that a relatively higher number of this entries have 0 stars with respect to the data portion having `avg_saving_percent` > 0.

In absence of further information useful to better qualify this feature, I have doubts about the reasonability of its distribution in the training data.

4.2.12. Feature Analysis: n_clicks

This feature represents the number of clicks the hotel has received in a specific time frame. It is the target variable of our regression task.

I assessed the absence of missing values, as well as that all values in the series are non-negative whole numbers.

Based on twentile, KDE and box plots I observed that the series is characterized by an half-normal distribution, with an high proportion of 0s (around 65% of entries).

By observing more in detail the box plot, a particular pattern can be identified: all `n_clicks` seem to be even numbers. This aspect has been verified.

This variable seems to assume a reasonable normal distribution. In particular, based on quantile plots, it can be observed how residuals spread along a polynomial or even exponential curve.

Given that this variable shows a non-negative skewed trend and approximately assumes an exponential form, it may be reasonable to regress over $Y = log(y + 1)$ then inverse the transformation with $y_{pred} = exp(Y_{pred}) - 1$. This would probably result in a more stabilized variance and minimized skewness; in other words the population would be automatically symmetrized and outliers would be addressed as well.

Based on the fact that all values in the dataset are even, I considered to divide the variable by two before executing the logarithm. This would not impact much with a generic learning model, but would improve visualizations readability on the human side, which may help in extracting further insights to drive the workflow.

4.2.13. Missing Value Pattern Analysis

The observations over missing values in training data reported in this chapter have been made throughout previous feature analyses.

- Approximately 1/1000 of data (508 entries) has missing `city_id`. All these entries have also missing `content_score`, `n_images`, `distance_to_center`, `avg_rating`, `stars` and `n_reviews`.
- Apart from entries described in the first point of this list, approximately 1/2400 of data (165 entries) has missing both `avg_price` and `avg_saving_percent`; there is no entry in which only one of these two values is missing. This pattern may be reasonable with hotels that never supplied any room; the absence of an average price entails the absence of any price at all: as a consequence no average saving percent can be computed.
- All entries mentioned in the previous bullet have also `n_clicks` $= 0$.

- Apart from entries described in the first point of this list and the only entry in which `n_images` is missing, approximately 1/20000 of data (20 entries) has missing both `distance_to_center` and `n_reviews`; there is no entry in which only one of the two values is missing.

- All entries mentioned in the previous bullet have also missing `avg_rating` and `stars`.

Once I had analyzed in-depth all features, I executed again my EDA routine by dropping all entries I dropped throughout the process at the beginning of routine execution, so as to achieve a feature analysis that is independent from the order in which I examined each feature.

If I had the chance, I would investigate about whether these patterns are not due to randomness and the data collection mechanisms which caused them. Uncovering such information may provide us with smart strategies for missing data imputation.

4.2.14. Dealing with Extreme and Strong Outliers

I realized throughout my feature analyses that in this business case it is needed to deal with both strong and extreme outliers.

At this point of my workflow I need a baseline strategy with regard to extreme outliers, which are mandatory to take into account. I could think of three ways to address the problem:

- remove such entries if their relative amount is negligible with respect to training data size;

- clip or transform extreme values of such entries to reasonable values which would negligibly impact on the rescaling process;

- use regressors able to accommodate extreme outliers.

The problem of extreme outliers involve `n_images` (ranging in $[-1, 2.4{*}10^6]$), `distance_to_center` (ranging in $[0, 1.8 * 10^7]$) and `n_reviews` (ranging in $[0, 2.8 * 10^5]$).

First of all, I evaluated the simplest solution, i.e. removing extreme outlier entries. Despite the proportion of entries that such action would remove from data, I do not like this solution, because it would be acceptable only by assuming that such outliers are erroneous, and that the remaining part of the population keeps the original representativeness of the underlying phenomenon even without them. I recommend to adopt a similar strategy only based on well-defined prior knowledge about the true domain limits of a feature, which I do not have for the above-mentioned features.

A good alternative is given by Tukey in his book on EDA (J. Tukey, Exploratory Data Analysis, 1977): transforming the data by applying a logarithm expression would symmetrize the residuals and automatically convert extreme outliers into strong outliers; according to Tukey, this solution would be acceptable if outliers have a strongly positively skewed distribution, which is our case.

I consequently decided to preprocess the above mentioned features in the following way:

$$n\_images = log(2 + n\_images) \tag{4}$$

$$distance\_to\_center = log(1 + distance\_to\_center) \tag{5}$$

$$n\_reviews = log(1 + n\_reviews) \tag{6}$$

With regard to strong outlier identification, I planned to possibly consider further iterations in my workflow focused on their treatment.

4.2.15. Bonus Question #3 – Univariate & Bivariate Feature Analysis

Pairplots and Bivariate KDE representations are valuable visual tools to understand more in-depth the impact of a set of predictors on the dependent variable. In order to leverage them in the best way, data should be opportunely prepared and conditioned for visualization.

As a first observation, extremely skewed distributions have been transformed with a logarithmic expression so as to overcome visual limits due to events happening at different scales and enhance plot readability. This involved not only `n_clicks`, but also `n_images`, `distance_to_center` and `n_reviews`. Moreover, a subsampling procedure without replacement has been applied on the available data so as to avoid populating plots with too many samples.

For what concerns Univariate Feature Analysis, I decided to visualize the dependent variable not only as a continuous quantity, but also by discretizing it on a logarithmic scale. In particular, I leveraged Bivariate KDE plots applied on a continuous logarithmic expression of the dependent variable, and multiple Univariate KDE plots applied on a logarithmic discretization of the dependent variable, as described by the following transformations:

$$n\_clicks_{Bivariate\_KDE} = log(1 + n\_clicks), \tag{7}$$

$$n\_clicks_{Multiple\_KDE} = clip(ceil(log(1 + n\_clicks)), 0, 5). \tag{8}$$

The second expression can be interpreted in the following way: as a first operation, the ceiling of the natural logarithm maps an integer in the input ranges $[0, 0]$, $[1, 2]$, $[3, 7]$, $[8, 19]$, $[20, 53]$, $[54, 147]$ respectively into 0, 1, 2, 3, 4, 5; the clipping operation is then applied to group all outlier samples having 148+ clicks into the last set, 5: this is needed to provide us with balanced enough distributions in each group, as well as to avoid dealing with too many levels (the more the levels, the harder is to visually discriminate related color nuances).

The first representation is provided in its own set of Bivariate KDE plots, whereas the second one is part of the given set of Pairplots, arranged along the diagonal of the plot matrix. Both representations are contained in `\visualizations`).

For what concerns Pairplots, the logarithmic representation explained above has been globally applied to them. I believe it emphasizes well how much a single predictor conditions each qualitative class of possible hotel instances; its related multiple KDE plot basically tells us how a predictor behaves with hotel instances having respectively 0 clicks, 1 click, few clicks, etc.. In addition, the complementing colored scatter plots provide us with similar information to the former case, but based on the joint action of each pair of predictors.

By leveraging univariate representations, I made the following inferences.

- The `content_score` is highly correlated with `n_clicks`: the related multiple KDE plot shows that the density of "brighter" distributions increases with the score, whereas "darker" distributions have larger and larger densities on low score values, in a perfectly proportional fashion.
- The highest-performing hotels (those belonging to class 5) typically have very large `n_images`.
- For what concerns `distance_to_center`, the majority of hotels located in city centers are the least clicked ones.
- The baseline imputation strategy applied to `avg_rating` masked any possible insight we could have had from the plot. It may be possible to leverage it only after a proper imputation regression procedure.
- The lowest number of clicks are generally taken by aparthouses and hostels, or more in general to hotel instances having 0 stars; on the other hand, the majority of the highest-clicked hotels have 4 stars.
- An even more powerful predictor than `n_reviews` is given by `content_score`: it shows a pattern analogous to the former variable, but more emphasized. In addition, it shows a gap in the range $(0, 4]$, which means that in all the training data `n_reviews` probably does not assume any value in the range $[0, 50]$. This hypothesis has then been confirmed on the whole training data.
- Outlier prices (hotels with extraordinarily expensive rooms) are never clicked. Given that a very small amount of data is associated with this behavior, it may be due to randomness or the abundance of the never clicked hotel instances with respect to the other classes.
- `avg_saving_percent` grows with `n_clicks`.
- The absence of reviews (`n_reviews=0`) generally entails `n_clicks=0`

Below are reported my insights extracted from bivariate representations.

- Based on the slight data orientation inferable from their scatter plot, `content_score` and `n_reviews` seem to play well with each other when they deal with samples having the largest number of clicks.
- A very interesting presumable relationship is given by the slight correlation between pair (`stars`, `avg_rank`) and the target, as well as pair (`stars`, `avg_saving_percent`) and the target: it seems that by increasing `n_stars`, `avg_rank` and `avg_saving_percent` become more and more determining in yielding a large number of clicks.

For what concerns additional variables I would include in order to reduce the error further, I would start with already available information that still needs to be feature-engineered or leveraged: for instance, I refer to additional boolean columns indicating whether `n_images==-1` or `stars==0`.

I believe that a reasonable, task-oriented customer segmentation may provide us with further categorical variables to leverage, useful for many tasks including the problem at hand. For instance, asking the customer at the most promising moments of the transaction with trivago whether he needs a room for leisure or business reasons, would allow us to better structure and model the overall customer/advertiser connection. The same can be done with advertisers: it is possible to ask them a categorical description on qualitative aspects of the hotel, such as building style (old-fashioned, futuristic, etc.)

Another informative predictor may be the distance of the hotel to the nearer point of interest. This would be especially useful for the task at hand.

5. Data Science Workflow: Production Phase

All steps of the Data Science workflow occurring after EDA–which I refer to as the Production Phase–require a clear, analytically defined high-level target to tackle. The challenge description defines it together with a proper performance metric, thus it is possible to proceed with the workflow; conversely, it would be possible to derive a goal based on more general business demands and EDA findings.

5.1. Preprocessing

This stage of the workflow prepares, cleans and conditions the available data for the task based on EDA findings. This routine is meant to safely provide subsequent procedures of the workflow with correct data (proper columns, formats, domains, …), and should generalize as much as possible on input data. All the work of this step is coded in `preprocess.py` and `visualize.py`.

First, the training dataset has been wholly typed as `float32`.

The `hotel_id` column has been then dropped.

Entries having `NaN`s on any feature excepting for `avg_rating` and `stars` have been dropped.

Entries having out-of-domain values for any feature with a known domain have been dropped.

An imputation strategy is applied on missing values of the `avg_rating` feature. As a baseline, I considered a simple rounded mean strategy.

Two boolean features have been consequently added to the dataframe, indicating entries containing `avg_rating` and `stars` values which have been imputed.

`n_images`, `distance_to_center` and `n_reviews` have been transformed with a logarithmic expression so as to address extreme outliers.

At this execution point I acquired Pairplots and Bivariate KDE plots included in my project.

As a final preprocessing step, features having known domain have been rescaled by a predefined scalar factor: `content_score` and `avg_rating` have been divided by 100, `stars` has been divided by 5 whereas `avg_saving_percent` has been rescaled by 99.

Finally provided with finalized data, I decided to represent it in a 3D scatter plot after the application of a dimensionality reduction technique, so as to assess by visual inspection whether the data exhibits meaningful structure for the task at hand. I do not consider this as a necessary step, but sometimes it may help in better understanding available data.

In particular, I applied a simple PCA (Principal Component Analysis) technique so as to transform features into a 3D space, then I assigned the dependent variable to the hue based on a colormap, so as to possibly identify visual correlations among spatial directions and colors. I represented the target variable as $log(1 + n\_clicks)$, so as to get an higher symmetry and homogeneity on the overall distribution and improve color visibility. It can be noticed that the dimensionality-reduced data shows relevant structure along two dimensions: I could clearly distinguish two clusters of points respectively characterized by low and high `n_clicks`, which spread in two different directions.

I consider this approach as a baseline for data visualization: PCA is variance-based and able to deal with linear relationships only, but there are other alternative dimensionality reduction techniques which may result in new visual patterns.

5.1.1. Updates

Given that I did not achieve significant results in my first iterations, I iterated over this step first by reconsidering `stars` as a categorical feature, then by removing `city_id`.

The imputation strategy I used for `stars` should have not been used as a baseline: I realized later that it is more reasonable to start by considering such feature as a categorical predictor, then proceed until a significant model is obtained and finally by starting with these kind of optimizations.

I should have reasoned similarly for the problematic `city_id` feature as well.

5.2. Train/Dev/Test Splitting

Given our large dataset size (more than 300 thousand entries) and assuming an homogeneous label balancing, I am confident that a 90/10 split between training+validation set and hold-out test set is a fair choice. Moreover, I need to evaluate the model in view of its productionalization. I must assume that all phenomenons that ultimately impact on the dependent variable vary over time: what is needed here is a model that is able to keep up with them; possibly, a model implementing online learning. Nonetheless, in this case there is no time reference available, so I extracted the test set by random sampling, as also planned for training/validation splitting. Based on twentile and distribution plots, I observed a balanced partitioning of data with respect to the target. In case it was not balanced, I would have given more priority to the inclusion in my workflow of a step in which I would have applied a stratified sampling technique, or anything that may rebalance the task at an acceptable degree.

5.3. Preprocessing

Before any preprocessing step, I should assess the presence of multicollinearity so as to possibly remove redundant columns. The multicollinearity phenomenon may affect the final performance based on the task type and the used model, but it should be assessed regardless of its direct impact: reducing the number of features leaves unvaried or possibly improves the performance, if we can assume that excluded features contain redundant information. I will add this step to the pipeline after a first, simplistic baseline evaluation.

However, it is prioritary to get a reasonably working baseline regressor first, i.e. a model able to provide us with significant results. I won't explore any dimensionality reduction approach in this iteration.

5.4. Training, Validation & Test – Bonus Question #3

I generated and saved as a figure the scatter plot associated with the results of each iteration. Given that first iterations did not provide me with significant results, I included just one of them as the visual "baseline" to consider.

I applied the non-parametric Wilcoxon signed-rank test with a threshold of 0.05 to identify a statistically significant difference between the results of two iterations.

In all of my experiments I applied a 10-fold cross validation over the `train_dev` set, and finally scored my model on the `test` set; I implemented the custom WMSE score provided in the challenge description.

5.4.1. Iteration 1

In this first iteration, I applied a simple LinearRegression model over the `n_clicks` target as it is, without transforming it. An interesting advantage of LinearRegression is the interpretability of its results, which may come in handy during regression analysis; it is also a good regression baseline to start with. In terms of preprocessing, I used a MinMaxScaler for numerical features and a OneHotEncoder followed by a dimensionality reduction method (TruncatedSVD). This latter categorical strategy has been adopted so as to convert the huge feature space output by OneHotEncoder applied on `city_id` into a space with a more reasonable number of dimensions. I initially planned to use PCA so that I could have had the possibility to put a minimum threshold on the explained variance ratio associated with the final space; I would have started by applying a threshold of 0.99, which typically removes a fair amount of noise without touching relevant information and typically provides us with a consistently reduced space.

Unfortunately, the sparseness of the resulting feature matrix is not suitable for using PCA, and so I attempted to make it work with TruncatedSVD. I started with a maximum reasonable value for `n_components` (1000, i.e. 1/10 of the original feature space size), which resulted providing the output space with a 0.42 explained variance ratio.

I achieved a validation/test WMSE of 0.00005/2.26144: my baseline model basically overfits on the training&validation data, but fails when applied to the hold-out test set. This can be explained by the fact that the model's inherently linear learning mechanisms are not robust enough to generalize over a variable having an exponential behavior.

Let us consider that the standard approach to assess whether results achieved by our model are significant is given by comparing it with the baseline random estimator. In a regression task, this consists in an estimator always returning the population mean. Based on the test data, the random estimator yields a WMSE of 2.23621.

As it can be seen from the regression results plot, the predictions of this model have a random distribution over the output domain.

5.4.2. Iteration 2

In order to address my hypothetical explanation to overfitting phenomenon involving linear regression, I tackled the problem from the other hand: I applied logarithmic transformations on the target before providing my linear regressor with it, then any prediction has been inverse-transformed exponentially. By doing so, the resulting scatter plots of the target became smoother, as well as its KDE plot; still, let us not forget that skewness keeps happening in the transformed space as well, and that it involves unknown nonlinear relationships to capture.

Once again, prediction results showed random behavior over the whole distribution.

5.4.3. Iteration 3

At this point I thought that evaluating the application of some shrinkage would be the prioritary action to make: I considered an ElasticNet model, cross-validated with a grid search encompassing a relevant set of `alpha` and `l1_ratio` values.

Unfortunately, the model was taking too long and so I thought that it may help to apply a change which would generally speed up computation but also probably improve the performance. I observed that after the application of one-hot encoding many new columns contain too few 1s to be considered relevant. In order to get rid of this additional, expensive and negligible categories I implemented a transformer for inclusion in my pipeline, which basically discards categorical columns having a number of 1s lower than the given parameter (I applied 5 and obtained a 70% reduction in the number of features).

5.4.4. Iteration 4

Finally I succeeded in completing my ElasticNet execution in a reasonable time frame. My grid has been emphasized on `alpha` and `l1_ratio` combinations. Given that results kept being randomly distributed, I did not consider GridSearchCV results to be significant.

5.4.5. Iterations 5-6-7

Then, I decided to experiment with PolynomialFeatures. I applied all possible basis functions at the 2^{nd} and 3^{rd} orders, without achieving significant results. I kept on going by exploring also linear and kernelized SVR models, without success. At this point I realized that I've probably been too optimistic in including all features and processing them based on my starting assumptions, and that I should have addressed this more carefully. I then decided to keep on going with my workflow but by removing the `city_id` column and by refactoring `stars` as a categorical feature.

5.4.6. Iteration 8

At this point I ran a grid search over a Random Forest model, so as to evaluate whether it is able to capture nonlinearities and in the meanwhile decorrelate trees, which is fundamental to properly address overfitting and multicollinearity. I finally achieved a slightly significant improvement with a Validation/Test WMSE of 0.0008/2.20104. The grid search explored different values for the number of estimators and the maximum number of features considered by each estimator. The addition of the `city_id` feature to the model did not improve nor worsen results; I planned to evaluate its impact in the next iterations.

5.4.7. Iteration 9

I leveraged more in-depth Random Forest capabilities by running a Randomized Search over a larger set of hyperparameters, but obtained no statistically significant improvement with respect to the previous iteration results.

5.4.8. Iteration 10

I achieved even better results with Random Forest using a weighted oversampling technique. I observed that high-valued samples are very scarce and that the WMSE gives more importance to them: I hypothesized that the important training samples were too scarce to be properly learned by the model. I devised and implemented the following oversampling strategy: the training population is doubled in size, and the new half of the population is sampled with replacement from the original population, where each original data point has a probability of being sampled which is proportional to its WMSE weight. In this way the new population contains the same information of the original one, with more emphasis (more duplicates) given to important samples. The application of this strategy resulted in a Validation/Test WMSE of 0.70885/2.12419 with Random Forest Regressor.

5.4.9. Iteration 11

I then applied XGBoost Regressor and achieved a Validation/Test WMSE of 0.00002/ 1.65574.

Gradient Boosted Decision Trees identify a class of complex supervised models based on Gradient Boosting. One specific implementation of this model, XGBoost, is very popular in the data science environment especially because it dominated Kaggle competitions over the last two years. I decided to apply this model to our task because of its good generalization capabilities, its relative insensitivity to tuning thanks to its strong regularization mechanism, its online learning capability and its sensitivity to outliers, which may be advantageous with our task. In particular, I applied a grid search exploring learning rate and maximum depth, and an early stopping autoregularizing strategy based on the WMSE score.

In addition, I proved the usefulness of my oversampling strategy by verifying a statistically significant improvement in using it even with this model.

5.4.10. Iteration 12

In the next iteration I removed all preprocessing steps involving logaritmic transformations on `n_clicks`, `n_reviews` and the other extremely skewed features. Results did

not change when I removed transformations over predictors, but consistently improved when I did it on the dependent variable: the model yielded a Validation/Test WMSE of 0.07226/1.21553. I verified once again that the weighted oversampling technique yields significantly better results even with this powerful model.

5.4.11. Iteration 13

I finally implemented an improved splitting procedure involving a stratified sampling strategy with respect to `city_id`: it yielded a Validation/Test WMSE of 0.0692/0.90849. I recognize the importance of this action, and that I should have done it at the beginning of my workflow: it is more reasonable to let the model deal with a balanced set of samples for each city between training and test sets, which also results in a fairer evaluation.

6. Conclusions

The goal of this case study is to implement a regressor especially good at predicting instances with large `n_clicks`, which have shown to be scarce in the available data.

My pipeline involved stratified splitting, one-hot encoding, normalization, grid/randomized search, linear and polynomial regression with shrinkage, SVR, random forest and XGBoost with early stopping. The process I applied allowed me to achieve significant results with respect to the naïve baseline ($WMSE = 2.25$), and yielded a progressive series of statistically significant improvements which resulted in a final $WMSE$ of 0.91. One reason of the effectiveness of Random Forest and XGBoost in providing us with relevant results as compared with previous models is probably given by the sensitivity of these models to outliers, i.e. the "important" samples in this case study; this hypothesis is further endorsed by the statistically significant impact of my weighted oversampling strategy on the performance of all relevant models.

The given dataset has shown to include many artifacts, probably due to post-processing alterations applied to original data, but in general it has been straightforward to understand or feature-engineer it.

I have been able to uncover the meaning of the unknown `stars` feature and conduct univariate and bivariate feature analysis by means of adequate visualizations; in particular, log-discretized pairplots proved to be a more effective visualization tool, as compared with bivariate KDE or traditional pairplots.

The `city_id` feature has shown to be as important as problematic: its large number of categorical levels made it difficult to properly feed it to my models. I observed that tree models based on node splitting may yield relevant results in similar cases: even if IDs are uncorrelated in their order, a structured series of node splits may significantly discriminate across them. In addition, the statistically significant improvement associated with the application of a stratified splitting strategy with respect to this feature proved once again its relevance.

There are still several explorations, adjustments and improvements that can be made on this business case, but unfortunately I did not have enough time available till expected submission time.

The next actions I would pursue at this point would be to apply regression imputation on `avg_rating` and `stars`, provide my XGBoost model with the WMSE objective function, attempt a clusterization over city ids and then carefully inspect representative sets of samples that my best models misclassified, so as to possibly identify and address generalizations and patterns over the sources of error.