



HUMBOLDT-UNIVERSITÄT ZU BERLIN

SCHOOL OF BUSINESS AND ECONOMICS

LADISLAUS VON BORTKIEWICZ CHAIR OF STATISTICS

STATISTICAL PROGRAMMING LANGUAGES

SEMINAR PAPER

# Exploratory Data Analysis of airbnb listings in Berlin

*Silvia Ventoruzzo*  
(592252)

submitted to

Alla PETUKHINA

March 12, 2019

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Data Preparation</b>	<b>2</b>
2.1	Berlin neighbourhoods and districts . . . . .	2
2.2	Berlin VBB Zones . . . . .	4
2.3	Airbnb listings' attributes . . . . .	7
<b>3</b>	<b>Exploratory Data Analysis</b>	<b>9</b>
3.1	Descriptive statistics . . . . .	10
3.2	Distribution plots . . . . .	12
<b>4</b>	<b>Price analysis</b>	<b>14</b>
4.1	Correlation with price . . . . .	14
4.2	Linear regression on price . . . . .	15
<b>5</b>	<b>Clustering</b>	<b>15</b>
<b>6</b>	<b>ShinyApp</b>	<b>19</b>
<b>7</b>	<b>Results and Conclusions</b>	<b>21</b>
	<b>References</b>	<b>22</b>

# 1 Introduction

- What is the subject of the study? Describe the economic/econometric problem.

Airbnb (airbnb.com) is now a famous website allowing private people and commercial entities to rent out part of their spaces. It all started in 2007 when two 27-year-olds decided to sublet their living room in their San Francisco apartment during a conference to help pay their rent (Surname, 2012). Now, .....

- What is the purpose of the study (working hypothesis)?

Many studies have already been conducted on price determinants for the hotel industry, like the ones named in Wang and Nicolau (2017), and for Airbnb properties (.....), but none so far specific for the city of Berlin. We therefore attempt an exploratory analysis of Airbnb listings in Berlin and, in particular, of their price. For this purpose also the properties were also clustered and....

- What do we already know about the subject (literature

As in Wang and Nicolau (2017) summarized, in the case of hotels, hotel prices are negatively influenced by the hotel's location, since shorter distance from the city center, the main attractions and/or important transportation points leads to higher prices. On the other side positive influences on price are hotels' star rating and online customer rating, the services and amenities provided, and by the "presense of car parks and fitness centers". But price determinants could be different in case of different types of accomodations, such as the ones offered on Airbnb. That's why Wang and Nicolau (2017) also derived the drivers of price for Airbnb listings from 33 cities.

- Provide an overview of your results.

- Outline of the paper:

*The paper is organized as follows. The next section describes the model under investigation. Section ?? describes the data set and Section ?? presents the results. Finally, Section 7 concludes.*

- The introduction should not be longer than 4 pages.

## 2 Data Preparation

For the analysis explained in this paper data was downloaded for a website independent from airbnb itself. [insideairbnb \(?\)](#) scrapes airbnb to get its data, posts it online for the public to use on own analysis, while also providing some analysis of its own.

The data is divided according to cities and for each there is general information about the city's properties and their availability for the next year. For this analysis not all variables are being kept, the focus is indeed on the ones that might have the most affect on price. Moreover, feature engineering will also be performed to extract even more useful information. More details in subsection 2.1.

Since we are dealing with data with coordinates, spatial data is also needed to produce a map of the location. For this purpose further data has been downloaded from the Statistics Office of Berlin-Brandenburg [\(?\)](#) and Geofabrik [\(?\)](#) and further processed, as explained in subsections 2.1 and 2.2.

To handle data will be make use of functions from the different packages in the **tidyverse**, which allows for easy manipulation of the data and flexible plotting, see Ross et al. (2017). For spatial data the package **sf** has been chosen, since it works well with the **tidyverse** packages and for all the other reasons listed in Pebesma (2018).

### 2.1 Berlin neighbourhoods and districts

Berlin consists of 96 neighbourhoods (Ortsteile), which are grouped into 12 districts (Bezirke). The data downloaded from [\(?\)](#) contain one polygon for each neighbourhood and additional information about them. The important variables for this analysis are showed in table 1.

Name	BEZNAME	geometry
Buckow : 2	Treptow-Köpenick :15	POLYGON :97
Adlershof : 1	Pankow :13	epsg:4326 : 0
Alt-Hohenschönhausen: 1	Reinickendorf :11	+proj=long...: 0
Alt-Treptow : 1	Lichtenberg :10	
Altglienicke : 1	Spandau : 9	
Baumschulenweg : 1	Charlottenburg-Wilmersdorf: 7	
(Other) :90	(Other) :32	

**Table 1:** Berlin's polygons data

The polygons have been extracted from the relative shapefile with the function `st_read` from the `sf` package. For the neighbourhoods we simply rename the variables and keep the ones of interest.

Listing 1: |berlin\_\_districts\_\_neighbourhoods.R|

```
1 # Load shapefiles
2 berlin = sf::st_read(file.path(getwd(), "Data", "Berlin-Ortsteile-polygon.
   shp", fsep="/"))
3
4 # Object with the neighbourhoods (and respective district)
5 berlin_neighbourhood_sf = berlin %>%
6   dplyr::rename(id = Name,
7                 group = BEZNAME) %>%
8   dplyr::select(id, group, geometry) %>%
9   dplyr::arrange(group)
```

However, as we can see from table 1 the neighbourhood Buckow is composed of two separate polygons, which we therefore need to unite according to the neighbourhoods' names. Thanks to the flexibility of the `sf` objects, this is done simply with a `summarize` from the package `dyplr`.

Listing 2: |berlin\_\_districts\_\_neighbourhoods.R|

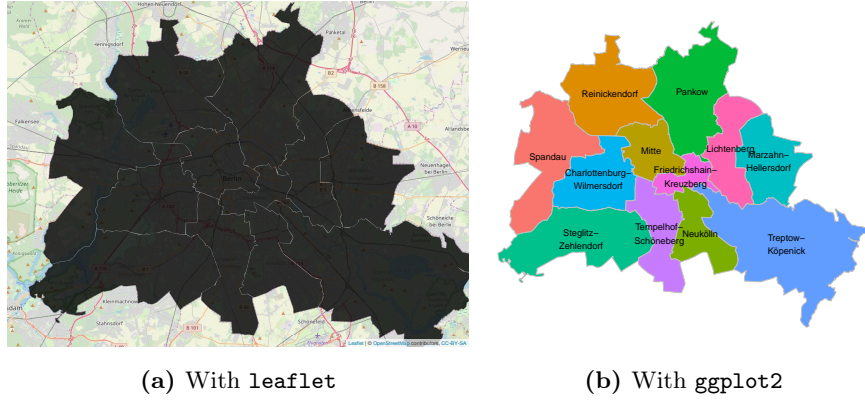
```
11 berlin_neighbourhood_singlebuckow_sf = berlin_neighbourhood_sf %>%
12   dplyr::group_by(id, group) %>%
13   dplyr::summarize(do_union = TRUE)
```

For the districts we perform the same procedure as above, but this time we unite the polygons only by their district, which are represented here by the group variable.

Listing 3: |berlin\_\_districts\_\_neighbourhoods.R|

```
19 berlin_district_sf = berlin_neighbourhood_sf %>%
20   dplyr::group_by(group) %>%
21   dplyr::summarize(do_union = TRUE) %>%
22   dplyr::mutate(id = group)
```

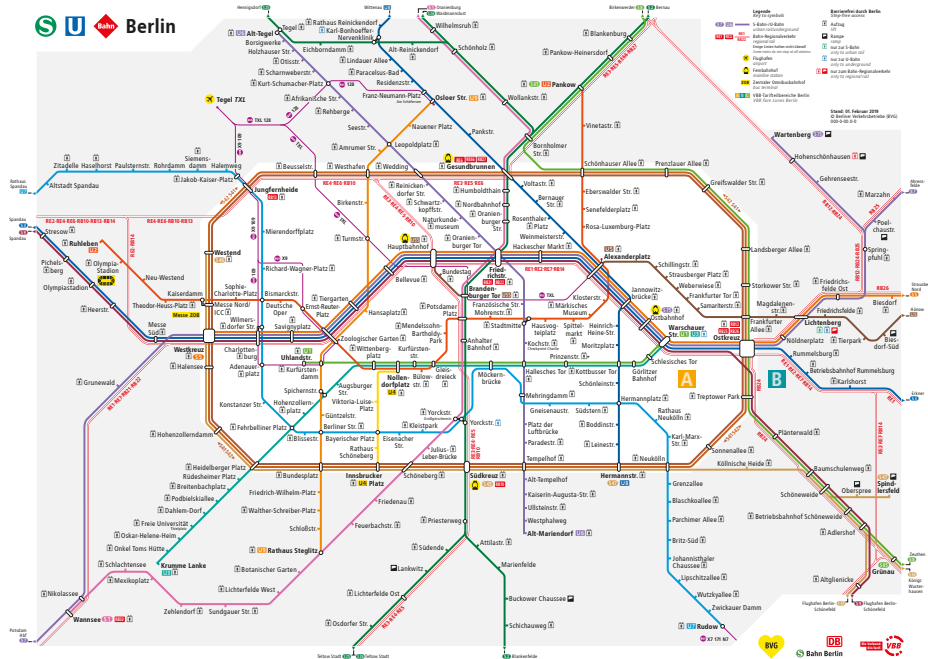
The produced polygons can be used to map Berlin using `ggplot2` or `leaflet`, as shown in figure 1. `leaflet` is more flexible and more suitable for plotting spatial data. Therefore, this package will be used for now on for mapping polygons and coordinate points.



**Figure 1:** Maps of the Berlin Districts  `berlin_districts_neighbourhoods_maps.R`

## 2.2 Berlin VBB Zones

The VBB (Verkehrsverbund Berlin-Brandenburg) is "the public transport authority covering the federal states of Berlin and Brandenburg" (CITATION: VBB Website). The city of Berlin, in particular, is divided in two fare areas: A, covering the center of Berlin up to the circular line (Ringbahn), and B, from the Ringbahn to the border with Brandenburg. After that there is also the area C, which however will not be covered here since we only consider the city of Berlin.



**Figure 2:** Network Map of Berlin Areas A and B (from VBB Website)

We tried to replicate these areas by also making use of the spatial points of the Berlin stations.

First of all, we need to create the polygon for area A, which is done by joining the points in the circular line and transforming this into a polygon.

Unfortunately the shapefile with the stations did not contain information about the line to which these stations correspond. Therefore we needed to filter the stations manually with the ones belonging to the circular line. This vector also contains the information in which order the points will be connected. You can notice that the first and last station are the same since the polygon needs to close.

**Listing 4: |berlin\_vbb\_zones.R|**

```
1 ringbahn_names_df = base::data.frame(
2   id = c("Südkreuz", "Schöneberg", "Innsbrucker Platz", "Bundesplatz",
3         "Heidelberger Platz", "Hohenzollerndamm", "Halensee", "Westkreuz",
4         "Messe Nord/ICC", "Westend", "Jungfernheide", "Beusselstraße",
5         "Westhafen", "Wedding", "Gesundbrunnen", "Schönhauser Allee",
6         "Prenzlauer Allee", "Greifswalder Straße", "Landsberger Allee",
7         "Storkower Straße", "Frankfurter Allee", "Ostkreuz", "Treptower
8         Park",
9         "Sonnenallee", "Neukölln", "Hermannstraße",
10        "Tempelhof", "Südkreuz"),
11   stringsAsFactors = FALSE) %>%
12   tibble::rownames_to_column(var = "order") %>%
13   dplyr::mutate(order = as.numeric(order))
```

Since some stations appear multiple times, we firstly filter railway stations, which include both subway and lightrail, and then we calculate the middle point for each station among the ones having the same name. We then join this with the dataframe containing the names of the stations in the circular line, thus filtering the stations to the ones we are interested in. After performing some preparation steps, the function `st_polygon` from the `sf` package was used to create a polygon out of a list of points.

**Listing 5: |berlin\_vbb\_zones.R|**

```
13 berlin_vbb_A_sf = stations %>%
14   dplyr::filter(fclass %like% "railway") %>%
15   dplyr::rename(id = name) %>%
16   dplyr::mutate(id = gsub("Berlin ", "", id),
17                id = gsub("Berlin-", "", id),
18                id = gsub(" *\\(.*?\\) *", "", id),
```

```

19         id = gsub("S ", "", id),
20         id = gsub("U ", "", id)) %>%
21     points_midpoint() %>%
22     dplyr::right_join(ringbahn_names_df, by = "id") %>%
23     dplyr::arrange(order) %>%
24     dplyr::select(long, lat) %>%
25     base::as.matrix() %>%
26     base::list() %>%
27     sf::st_polygon() %>%
28     sf::st_sfc() %>%
29     sf::st_sf(crs = sf::st_crs(berlin))

```

Second of all, in order to create the polygon for zone B, we need to produce the polygon for entire Berlin. This is done with a procedure already explained in subsection 2.1. In this case we don't perform any grouping, thus uniting all neighbourhoods.

**Listing 6: |berlin\_vbb\_zones.R|**

```

31 berlin_sf = berlin %>%
32     dplyr::summarize(do_union = TRUE)

```

Finally, we bind the two objects by row and calculate their intersections thanks to the function `st_intersection` from the package `sf`. We then define the area names according to how many times the two previous polygons intersect:

- Area A: where polygons intersect (n. overlaps  $> 1$ )
- Area B: where polygons do not intersect (n. overlaps  $\leq 1$ )

**Listing 7: |berlin\_vbb\_zones.R|**

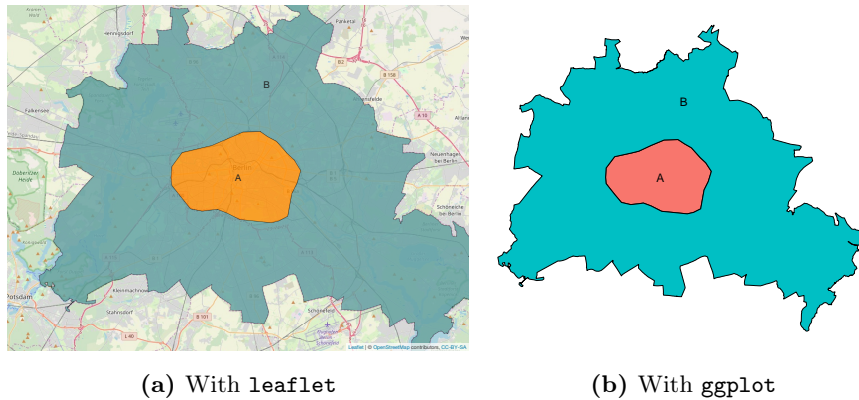
```

34 berlin_vbb_AB_sf = berlin_vbb_A_sf %>%
35     base::rbind(berlin_sf) %>%
36     sf::st_intersection() %>%
37     dplyr::mutate(id = ifelse(n.overlaps > 1, "A", "B")) %>%
38     dplyr::select(-n.overlaps, -origins) %>%
39     dplyr::arrange(desc(id))

```

As in subsection 2.1 the `sf` object can be used to map the VBB zones using `leaflet` or `ggplot2`.





**Figure 3:** Maps of the Berlin VBB Zones berlin\_vbb\_zones\_maps.R

## 2.3 Airbnb listings' attributes

The first part of cleaning the airbnb data consists in joining the two datasets containing general information according to their common variables and correcting some string values. Secondly, we proceed in checking for missing values and deriving that information from other correlated variables. Thirdly, we move on to feature engineering.

We firstly derive the areas where the properties are located thanks to the spatial polygons created before and the function `point_in_polygons`. This function loops through the polygons in the `sf` object and check which points are contained in which polygons. In the end it writes the id associated to the polygon, in our case the area id, in the summary column, which can be named as preferred.

**Listing 8:** |point\_in\_polygons.R|

```

1 point_in_polygons <- function(points_df, polys_sf,
2                               var_name, join_var = "id") {
3   # Create empty dataframe
4   is_in <- data.frame(matrix(ncol = nrow(polys_sf), nrow = nrow(points_df)))
5   is_in[,var_name] <- NA
6   # Extract coordinates of the polygons
7   coordinates <- as.data.frame(st_coordinates(polys_sf))
8   # Extract names of the polygons
9   name <- as.character(polys_sf$id)
10  # For all polygons check if the points are inside of them
11  for (k in 1:nrow(polys_sf)) {
12    is_in[,k] <- sp::point.in.polygon(point.x = points_df$long,
13                                      point.y = points_df$lat,
14                                      pol.x = coordinates$X

```

```

15         [coordinates$L2 == k],
16         pol.y = coordinates$Y
17         [coordinates$L2 == k])
18     # Get the names of the polygons where the points are in one column
19     is_in[,var_name][is_in[,k] == 1] <- name[k]
20 }
21 # Keep only summary column and add points' names
22 is_in <- is_in %>%
23     dplyr::select(var_name) %>%
24     dplyr::mutate(id = points_df$id)
25 # Add the summary column to the points dataframe
26 points_df <- dplyr::full_join(points_df, is_in, by = join_var)
27 return(points_df)
28 }

```

Secondly, for railway stations and tourist attractions we calculate the amount inside a range and the distance to the nearest point using the function `distance_count`. In particular, the following parameters will be used:

- Railway stations: distance = 1000 (1 km)
- (Top 10) attractions: distance = 2000 (2 km)

This function firstly calculates the distance between all properties and all reference points using the Haversine Formula, which "gives minimum distance between any two points on spherical body by using latitude and longitude" (Ingole and Nichat, 2013) according to the following formula:

$$d = 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_2) \cos(\phi_1) \sin^2 \left( \frac{\psi_2 - \psi_1}{2} \right)} \right) \quad (1)$$

Then it calculates how many reference points are within the set distance and how much is the distance to the nearest reference point. Finally this information is joined into the main dataframe.

**Listing 9: |distance\_count.R|**

```

1 distance_count = function(main, reference, var_name, distance) {
2     # Create variable names
3     var_name_count = paste(var_name, "count", sep = "_")
4     var_name_dist = paste(var_name, "dist", sep = "_")
5     # Calculate distance for each listing to each station

```

```

6   point_distance = geosphere::distm(x = main %>%
7                                   dplyr::select(long, lat),
8                                   y = reference %>%
9                                   dplyr::select(long, lat),
10                                  fun = distHaversine) %>%
11
12   as.data.frame() %>%
13   data.table::setnames(as.character(reference$id))
14   # Calculate how many "reference" are within "distance"
15   point_distance[, var_name_count] = rowSums(point_distance <= distance)
16   %>%
17
18   as.factor()
19
20   # Calculate the distance to the nearest "reference"
21   point_distance[, var_name_dist] = apply(point_distance[, -ncol(point_
22   distance)],
23
24   MARGIN = 1,
25   FUN = min) %>%
26   round(0)
27
28   # Insert this information into the main DF
29   main = point_distance %>%
30     dplyr::mutate(id = main$id) %>%
31     dplyr::select(id, var_name_count, var_name_dist) %>%
32     dplyr::right_join(main, by = "id")
33
34   return(main)
35 }

```

In the end we use the calendar dataframe to calculate availability of each property in the different seasons.

**Listing 10: |data\_preparation.R|**

### 3 Exploratory Data Analysis

The exploratory data analysis will be divided in two parts: subsection 3.1 will show the descriptive statistics calculated either numeric or categorical variables; subsection 3.2 will then display the distribution plots. Correlation will also be calculated, but only with respect to price. This subject will be dealt with in subsection 4.1.

### 3.1 Descriptive statistics

Descriptive statistics make the interpretation of the data easier by giving grouping it and thus providing a shorter representation of it (Ibe, 2014).

As in Ibe (2014) explained there are three general types of descriptive statistics:

1. Measures of central tendency
2. Measures of spread
3. Graphical displays

This subsection will focus on the first two, while subsection 3.2 will deal with the third.

Most of these statistics are usually applied to continuous data, sometimes even to numerical discrete data, but not to categorical variables. Therefore the function to calculate descriptive statistics has been split in two.


The first part calculate both measures of central tendency and spread of all numerical variables thanks to the function `apply`. The function `apply(X, MARGIN, FUN, ...)` calculates the function in `FUN` for all rows (`MARGIN = 1`) or columns (`MARGIN = 2`) for the data in `X`. One can add additional arguments, like the quantile probabilities in our case.

Listing 11: |descriptive\_statistics.R|

```
1 descriptive_statistics = function(df) {  
2  
3   # Descriptive statistics for numeric variables  
4   if (unique(apply(df, 2, function(x) is.numeric(x)))) {  
5  
6     summary = data.frame(  
7       variable = names(df),  
8       min      = apply(df, 2, min),  
9       '1Q'     = apply(df, 2, quantile, probs = 0.25),  
10      median   = apply(df, 2, median),  
11      '3Q'     = apply(df, 2, quantile, probs = 0.75),  
12      max      = apply(df, 2, max),  
13      iqr      = apply(df, 2, IQR),  
14      mean     = apply(df, 2, mean),  
15      sd       = apply(df, 2, sd),  
16      check.names = FALSE) %>%  
17      dplyr::mutate_if(is.numeric, function(x) round(x, 4))
```

Part of the results can be seen in table 2.

variable	min	1Q	median	3Q	max	iqr	mean	sd
price	0.00	30.00	47.00	70.00	9000.00	40.00	67.69	210.53
review_scores_rating	0.00	84.00	95.00	100.00	100.00	16.00	77.26	37.09

**Table 2:** Sample of descriptive table for numeric variables  descriptive\_statistics.R

For categorical variables the above used statistics do not work, therefore frequencies and proportions of each factor were calculated. The mode is then simply the factor with the highest frequency.

**Listing 12:** |descriptive\_statistics.R|

```

19 } else if (unique(apply(df, 2, function(x) is.character(x) | is.factor(x)
20 | is.logical(x)))) {
21
22   # Frequency
23   frequency_list = apply(df, 2, table)
24
25   frequency_df = data.frame(frequency = unlist(frequency_list)) %>%
26     tibble::rownames_to_column(var = "var_fact")
27
28   freq_var = colsplit(string = frequency_df$var_fact, pattern = "\\.",
29     names = c("variable", "factor"))
30
31   frequency = freq_var %>%
32     cbind(frequency_df) %>%
33     dplyr::select(-var_fact)
34
35   # Proportion
36   proportion_list = apply(df, 2, function(x) prop.table(table(x)))
37
38   proportion_df = data.frame(proportion = unlist(proportion_list)) %>%
39     tibble::rownames_to_column(var = "var_fact")
40
41   prop_var = colsplit(string = proportion_df$var_fact, pattern = "\\.",
42     names = c("variable", "factor"))
43
44   proportion = prop_var %>%
45     cbind(proportion_df) %>%
46     dplyr::select(-var_fact) %>%
47     dplyr::mutate(proportion = round(proportion, 4)*100,

```

```

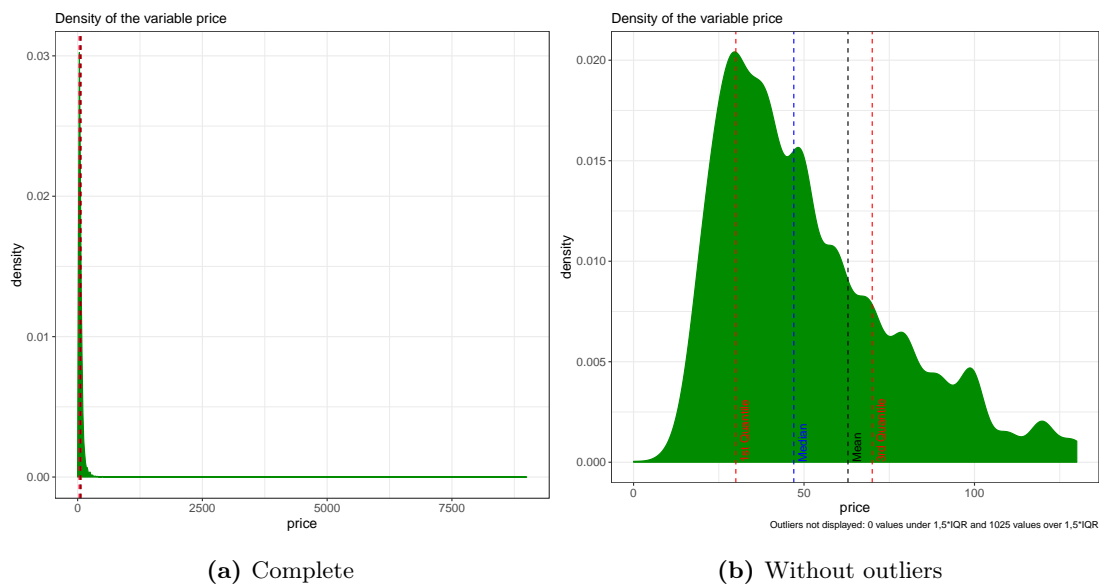
45         proportion = as.character(proportion) %>% paste("%")
46
47     summary = frequency %>%
48         dplyr::inner_join(proportion, by = c("variable", "factor")) %>%
49         dplyr::arrange(variable, desc(frequency))


```

## 3.2 Distribution plots

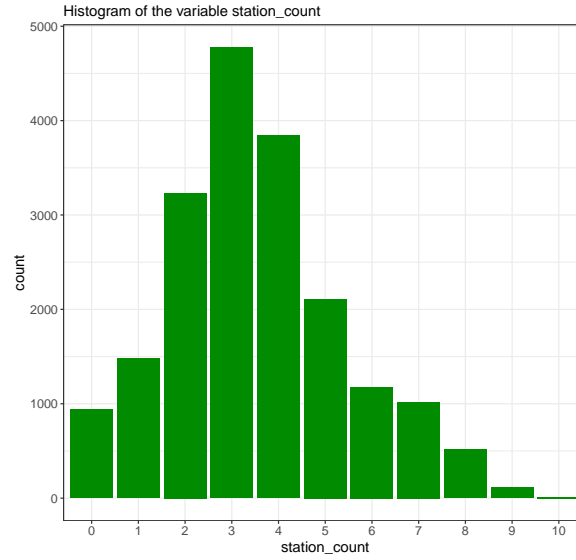
Also for the distribution plots we distinguish between numerical and categorical variables. In the former case a density plot has been chose, while in the latter a bar plot.


Unfortunately, many numeric variables present outliers, which, in the case of very skewed data, have been excluded for better visualization. An example can be seen in figure 4.



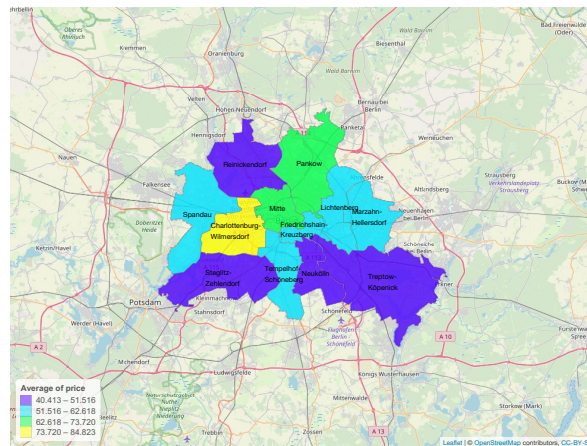
**Figure 4:** Distribution of the variable price  distribution\_plot.R

For the categorical variables a bar plot is more appropriate, since variable can only assume a limited amount of values, and usually only a few, like in the case displayed in figure 5.



**Figure 5:** Distribution of the variable station\_count  distribution\_plot.R

Having succesfully created spatial polygons for Berlin, we can also map the distribution of the variables across the city. For example, in figure 6 one can see how the average of price in each neighbourhood is distributed across neighbourhoods.



**Figure 6:** Distribution of the average of price across Berlin's districts

## 4 Price analysis

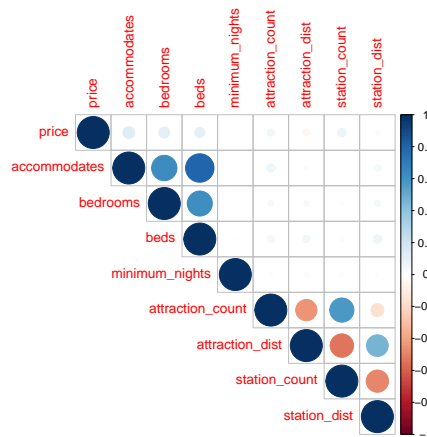
One of the core factors in the choice of the property to book is its price, being one of main drivers of customers' behaviors (Liang et al., 2018). Therefore one may want to try and see what properties' attributes influence its value.

We start in subsection 4.1 by calculating the correlation of all the other variables with respect to price. Then we try in subsection 4.2 to run a linear regression on price to look what variables are statistically relevant and how much they affect the price.

### 4.1 Correlation with price

Since we are only interested in the correlation with price, a classic correlation plot like the one in figure 7 may not be the most easily readable in this case, especially because of the large number of variables.

– UNDERSTAND HOW TO PUT IMAGE embedded in text



**Figure 7:** Correlation plot with the function `corrplot` from the package `corrplot`

In fact, categorical variables first need to be transformed into many dummy variables in order to calculate the correlation. The function `from_row_to_col` will be used to achieve this result. It creates dummy variables for all factor levels by creating a column of 1s, spreading it across so many columns as factor levels and filling the empty rows in the columns with 0s.

**Listing 13:** `|from_row_to_col.R|`

```
1 from_row_to_col = function(df, variable) {  
2   df = df %>%  
3     dplyr::mutate(yesno = 1) %>%
```



```

4     dplyr::distinct() %>%
5     tidyr::spread(variable, yesno, fill = 0)
6     return(df)
7 }

```

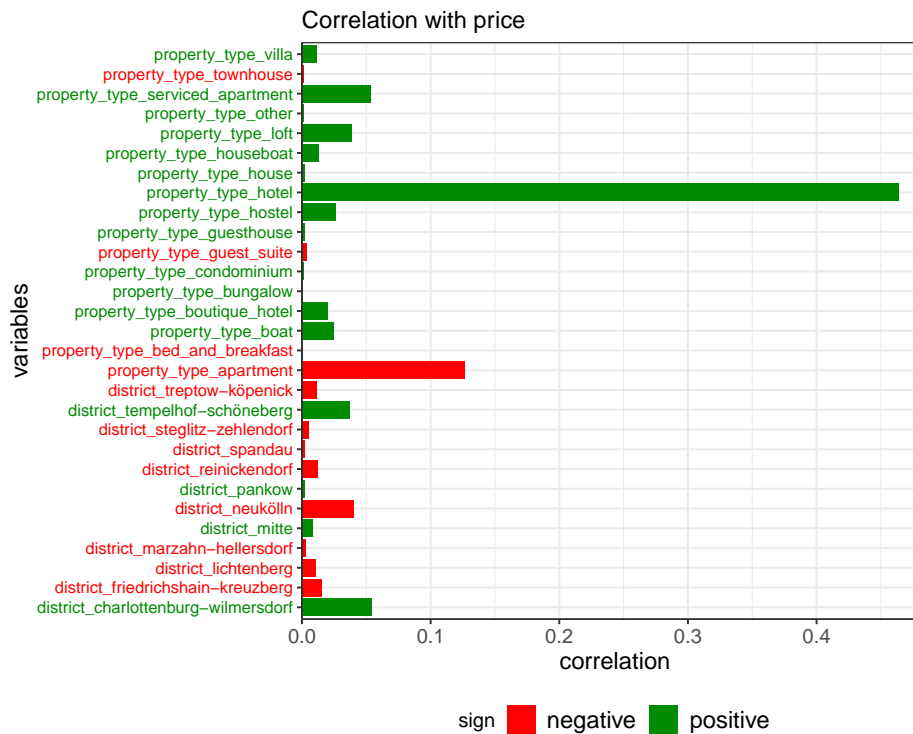


Figure 8: Sample of plot of correlation with price

## 4.2 Linear regression on price

As pointed out in Wang and Nicolau (2017) linear regression is used to describe possible linear relationships between a dependent variable and one or multiple independent variables.

In this case we will look for the relationship that the properties' attributes have on their price.

## 5 Clustering

A further step in data analysis is clustering, that is, trying to find groups in the data where they are similar to each other but different than data in other groups (Kaufman and Rousseeuw, 2009).

In this study we tried to see if Airbnb properties in Berlin can be split into clusters sharing

common features. For this purpose we will use k-means clustering, one of the most famous and used partitioning methods.

The basic ideas of this method were developed at the beginning of the second half of the 20<sup>th</sup> century (Bock, 2007). This approach partitions the data to its closest centroid in one of the user-specified  $k$  number of clusters. The process works according to the following steps (Tan, 2018):

1.  $k$  points are randomly selected to be centroids.
2. The rest of the data points are assigned to the respectively closest centroid, i.e. the one which minimizes the sum of the squared error (SSE)

$$SSE = \sum_{i=1}^K \sum_{\mathbf{x} \in C_i} \text{dist}(\mathbf{c}_i, \mathbf{x})^2 \quad (2)$$

where  $\text{dist}(\mathbf{c}_i, \mathbf{x})$  is the Euclidean distance between the centroid of the  $i^{\text{th}}$  cluster and a point  $\mathbf{x}$ .

3. New centroids are selected, being the mean of all points in the respective cluster.
4. Repeat points 2. and 3. until the centroids stop changing or the maximum number of iterations has been reached.

Subsection ?? will deal with the process of choosing the appropriate number of clusters to be used in the clustering algorithm, while subsection ?? will handle the clustering itself.

## 5.1 Choosing the number of clusters

According to procedure explained above, the first thing to do is to choose the number of clusters  $k$ . As illustrated in Kodinariya and Makwana (2013), there are multiple approaches to selecting the appropriate value of  $k$ . In this study the elbow method was used, since it can be calculated using the k-means algorithm and it does require too much computational power.

This method is a visual rule of thumb implemented in the function `number_of_clusters` and is Madhulatha (2012). It firstly requires the user to cluster the data multiple times with increasing values of  $k$ , starting at  $k = 2$ .

Listing 14: `|number_of_clusters.R|`

```
1 number_of_clusters = function(scaled_df, min = 2, max,
2                               iter_max = 10, plot_breaks) {
```

```

3  # Set seed for reproducibility
4  set.seed(900114)
5  # Values to test
6  k_values = min:max
7  # Empty dataframe for the total variance explained
8  tve = data.frame(clusters = k_values,
9                  tve      = rep(NA, length(k_values)))
10 # Empty list for the kmeans objects
11 clk = list()
12 # Loop through the possible values of k
13 for (k in k_values) {
14   # Calculate k-means for each k
15   clk[[k-1]] = kmeans(scaled_df, centers = k, iter.max = iter_max)

```

Secondly one calculates for each run the percentage of the total variance explained.

Listing 15: |number\_of\_clusters.R|

```

17  # Save the number of clusters
18  names(clk)[k-1] = paste(k, "clusters", sep = " ")
19  # Calculate percentage of total variance explained
20  tve$tve[k-1] = 1-clk[[k-1]]$tot.withinss/clk[[k-1]]$totss
21  # Print process
22  print(paste("k-means with", k, "clusters done", sep = " "))
23  }

```

One then plot the total variance explained against the number of clusters and chooses the value of  $k$  where adding another cluster does not add sufficient information (Madhulatha, 2012).

Listing 16: |number\_of\_clusters.R|

```

25
26  # Plot tve against k values
27  plot = ggplot(data = tve, aes(x = clusters, y = tve)) +
28    geom_line(color = "grey") +
29    geom_point(color = "red") +
30    scale_x_continuous(breaks = plot_breaks) +
31    labs(x      = "number of clusters",
32         y      = "% of tve",
33         title = paste("Plot of total variance explained for k from", min, "
34                       to", max, sep = " ")) +
35    theme_bw() +
36    theme(axis.text.x = element_text(size = rel(1.2)),

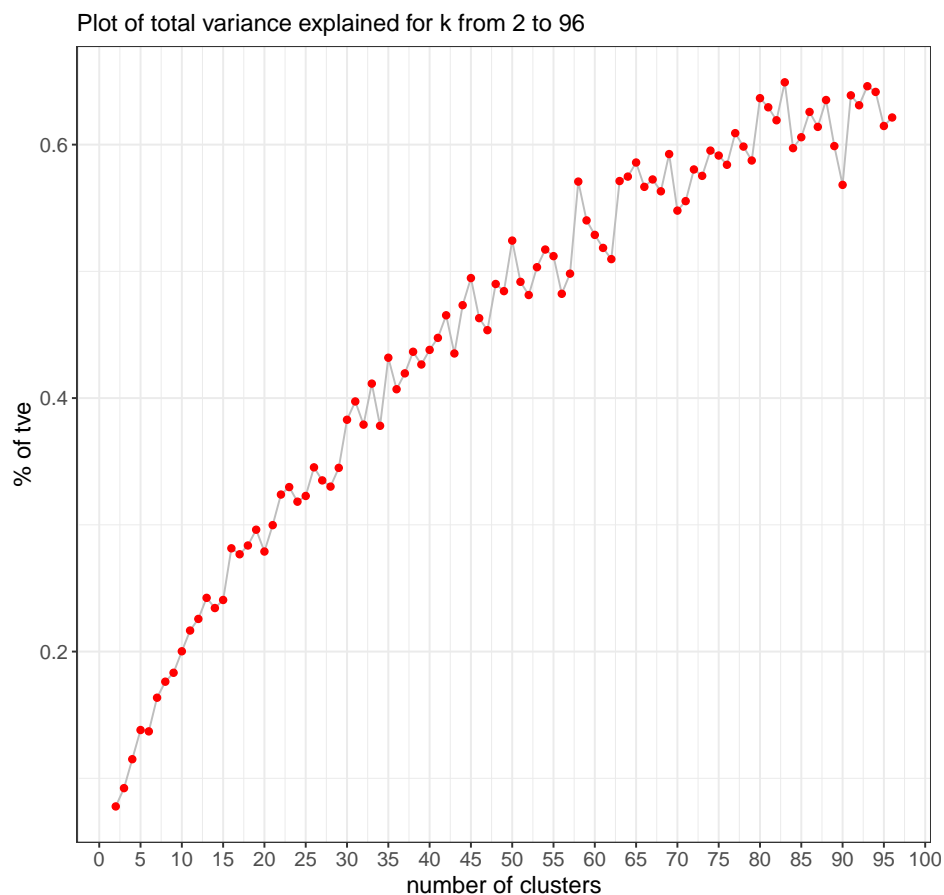
```


```

36     axis.text.y = element_text(size = rel(1.2)),
37     axis.title.x = element_text(size = rel(1.2)),
38     axis.title.y = element_text(size = rel(1.2)))
39
40     return(plot)
41 }

```

The resulting plot can be seen in 9. In this case there is no general best, since the value of *tve* is not continually increasing.



**Figure 9:** Plot of the total variance explained against different values of  $k$   number\_of\_clusters.R

## 5.2 k-means clustering

One can now use this number inside the function `kmeans` and look at the cluster assignments of the different points.

Having multiple variables, plotting the clustered points with respect to all variables is

not feasible, one need therefore to find alternatives. Since we have coordinate points, one possibility is to plot the points with respect to their longitude and latitude, like in figure .... Another one can be to plot them with respect to the first two principal components.

## 6 ShinyApp

`shiny` is an R package that allows the user to write interactive web applications. These are especially helpful when delivering information to people with no coding experience in a very user-friendly way.

As described in the official site of `shiny`, in its most basic version an App is contained in a single script called `app.R`. As the Apps become more and more complicated one can write the code in two scripts, `ui.R` and `server.R`, or even further split these into thematic scripts.

The App structure is divided into two main elements:

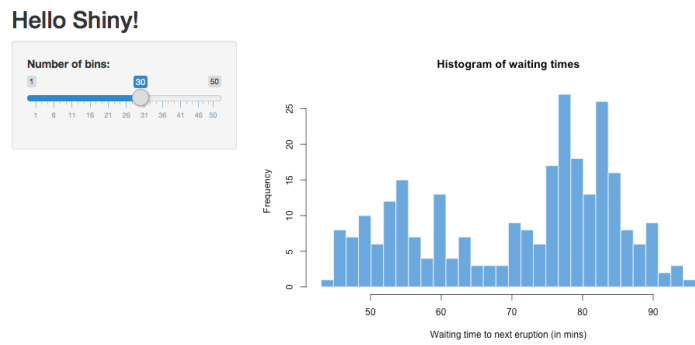
- *ui*: The user interface object determines the appearance of the App itself. Here the possible inputs and the outputs will be defined.
- *server*: The server function uses the input values chosen in the App to produce the outputs indicated in the user interface.

Finally the App will be called using the function `shinyApp(ui, server)`.

A simple example from the `shiny` website shows this in action.

```
1 ui = fluidPage(  
2   titlePanel("Hello Shiny!"),  
3   sidebarLayout(  
4     sidebarPanel(sliderInput(inputId = "bins",  
5                             label = "Number of bins:",  
6                             min = 1, max = 50, value = 30)),  
7     mainPanel(plotOutput(outputId = "distPlot"))  
8   )  
9  
10 server = function(input, output) {  
11   output$distPlot = renderPlot({  
12     x = faithful$waiting  
13     bins = seq(min(x), max(x), length.out = input$bins + 1)  
14     hist(x, breaks = bins, col = "#75AADB", border = "white",  
15          xlab = "Waiting time to next eruption (in mins)",  
16          main = "Histogram of waiting times"))  
17   }  
18  
19 shinyApp(ui = ui, server = server)
```

This simple App produces the result in figure 10.



**Figure 10:** Example of simple ShinyApp from website

Because of its interactivensess and flexibility a ShinyApp was built for this project in order to enable the final user a chance to a first hand analysis of the data. The App is reachable at this link (INSERT SHINY APP LINK).

## 7 Results and Conclusions

- Give a short summary of what has been done and what has been found.
- Expose results concisely.
- Draw conclusions about the problem studied. What are the implications of your findings?
- Point out some limitations of study (assist reader in judging validity of findings).
- Suggest issues for future research.



## References

- BOCK, H.-H. (2007): “Clustering methods: a history of k-means algorithms,” in *Selected contributions in data analysis and classification*, Springer, 161–172.
- IBE, O. (2014): *Fundamentals of applied probability and random processes*. 2nd ed, Academic Press, chap. 8.
- INGOLE, P. AND M. M. K. NICHAT (2013): “Landmark based shortest path detection by using Dijkstra Algorithm and Haversine Formula,” *International Journal of Engineering Research and Applications*, 3, 162–165.
- KAUFMAN, L. AND P. J. ROUSSEEUW (2009): *Finding groups in data: an introduction to cluster analysis*, vol. 344, John Wiley & Sons.
- KODINARIYA, T. M. AND P. R. MAKWANA (2013): “Review on determining number of Cluster in K-Means Clustering,” *International Journal*, 1, 90–95.
- LIANG, L. J., H. C. CHOI, AND M. JOPPE (2018): “Understanding repurchase intention of Airbnb consumers: perceived authenticity, electronic word-of-mouth, and price sensitivity,” *Journal of Travel & Tourism Marketing*, 35, 73–89.
- MADHULATHA, T. S. (2012): “An overview on clustering methods,” *arXiv preprint arXiv:1205.1117*.
- PEBESMA, E. (2018): “Simple Features for R: Standardized Support for Spatial Vector Data,” *The R Journal*, 10, 439–446.
- ROSS, Z., H. WICKHAM, AND D. ROBINSON (2017): “Declutter your R workflow with tidy tools,” Tech. rep., PeerJ Preprints.
- SURNAME, A. (2012): “Airbnb: The story behind the 1.3bnroom – lettingwebsite,”.
- TAN, P.-N. (2018): *Introduction to data mining*, Pearson Education India, chap. 8.
- WANG, D. AND J. L. NICOLAU (2017): “Price determinants of sharing economy based accommodation rental: A study of listings from 33 cities on Airbnb. com,” *International Journal of Hospitality Management*, 62, 120–131.

## **Declaration of Authorship**

I hereby confirm that I have authored this seminar paper independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

Berlin, March 12, 2019

Silvia Ventrizzo